UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE BACHARELADO DE SISTEMAS DE INFORMAÇÃO

AMBIENTE DE *GADGETS* PARA TV DIGITAL

André Lima Rocha Campos

Florianópolis, outubro de 2010

AMBIENTE DE *GADGETS* PARA TV DIGITAL

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Bacharel, do curso de Sistemas de Informação da Universidade Federal de Santa Catarina.

Professor Orientador:

Fernando Ostuni Gauthier

Florianópolis, 2010.

André Lima Rocha Campos

AMBIENTE DE *GADGETS* PARA TV DIGITAL

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Bacharel, do curso de Sistemas de Informação da Universidade Federal de Santa Catarina.

Aprovado em:

ORIENTADORES:

_____
Prof. Fernando Ostuni Gauthier

_____
Prof. José Leomar Todesco

BANCA EXAMINADORA:

_____
Airton Zancanaro

_____
Greicy Kelli Spanhol Lenzi

Agradecimentos

Agradeço a toda minha família pelo apoio e em especial, à Rafaella Machado, que me ajudou na elaboração deste trabalho.

Resumo

Em virtude de não se ter um padrão ou modelo de desenvolvimento que trate a forma com que são criadas aplicações que se agregam a TV digital, como um utilitário ou serviço, neste trabalho foi proposto um modelo conceitual de um ambiente para incorporar essas aplicações. Para que fosse possível criar uma proposta de *software* por meio de padrões, foi realizado um estudo sobre a TV digital brasileira. Com base no modelo desenvolvido, pôde-se implementar uma aplicação que atendeu a requisitos definidos previamente. Essa aplicação fez uso das linguagens de programação NCL e Lua que fazem parte da especificação do *middleware* Ginga e foi denominada *TV Digital Deck*. Mesmo com algumas limitações, obteve-se um resultado satisfatório, que pode incentivar novos trabalhos de especificação e também desenvolvimento.

Palavras-chave: TV Digital. Ginga. *Middleware*. NCL. Lua.

Abstract

By virtue of not having a standard or model of development that addresses the way applications are created that join the digital TV as a utility or service, this paper proposed a conceptual model of an environment to incorporate these applications. To enable it to create a proposal for software through standards, a study was conducted on the Brazilian digital TV. Based on the model developed, we could implement an application that met the requirements defined in advance. This application made use of programming languages NCL and Lua that are part of the specification of the middleware and was named TV Digital Deck. Even with some limitations, we obtained a satisfactory result, which may encourage further work to specification and also development.

Keywords: Digital TV. Ginga. *Middleware*. NCL. Lua

SUMÁRIO

LISTA DE SIGLAS E ABREVIATURAS

SBTVD-T – Sistema Brasileiro de Televisão Digital Terrestre

NCL – *Nested Context Language*

XML – *Extensible Markup Language*

PUC-Rio – Pontifícia Universidade Católica do Rio de Janeiro

ISDB-T – *Integrated Services Digital Broadcasting Terrestrial*

ATSC – *Advanced Television Systems Committee*

UFPB – Universidade Federal da Paraiba

UFMA – Universidade Federal do Maranhão

MPEG – *Moving Picture Experts Group*

AVC – *Advanced Video Coding*

LAViD – Laboratório de Aplicações de Vídeo Digital

GPL – *General Public License*

API – *Application Programming Interface*

NCM – *Nested Context Model*

HTML – *HyperText Markup Language*

URI – *Uniform Resource Identifier*

IDE – *Integrated Development Environment*

SSH – *Secure Shell*

ASC – *American Standard Code*

PNG – *Portable Network Graphics*

DOM – *Document Object Model*

RSS – Really Simple Syndication

LISTA DE QUADROS

LISTA DE FIGURAS

1 INTRODUÇÃO

1.1 Considerações iniciais

A TV digital é uma nova tecnologia de transmissão de sinais de televisão, que proporcionará ao telespectador melhor qualidade de imagens, sons e uma série de novos benefícios tais como interatividade com os programas, também a possibilidade de utilização em dispositivos móveis.

Em virtude de não se ter um padrão ou modelo de desenvolvimento que trate a forma com que são criadas aplicações que se agregam a TV como um utilitário ou serviço, foi proposto um modelo conceitual de um ambiente para incorporar essas aplicações que foram denominadas *gadgets*.

Gadget (em inglês: geringonça, dispositivo) na área de tecnologia da informação é o nome que se dá a algum pequeno *software*, ferramenta ou serviço que pode ser agregado a uma aplicação ou ambiente maior (MELLO, 2009).

No contexto da TV Digital, qualquer aplicação que ofereça algum tipo serviço pode ser considerada como um *gadget*, nenhum modelo ou pré-requisito é necessário para que um utilitário desenvolvido para TV Digital possa receber o título de *gadget*. Com isso também foi proposto um modelo de desenvolvimento de *gadgets* para a TV Digital de forma que eles possam ser utilizados pela aplicação proposta.

Com base em toda proposição de modelos e padrões tanto para o software como para os *gadgets*, foi feita a implementação de um ambiente de incorporação chamado de *TV Digital Deck* seguindo as especificações contidas no modelo.

A tradução em português da palavra *deck* é *convés*. Em um navio, o convés é a estrutura horizontal que reforça o casco e serve como superfície principal de trabalho. Essa expressão é também usada em jogos de baralho, "*deck* de cartas", que consiste em uma seleção com as melhores cartas que servem de base para o jogo. Ou seja, a escolha do nome *TV Digital Deck* foi feita pensando em uma superfície de trabalho, na qual conste a seleção dos melhores elementos que estão disponíveis para uso, fazendo alusão ao ambiente de *gadgets*, com os *gadgets* escolhidos pelo usuário.

Para que essa implementação fosse posta em prática, foi utilizada uma camada de *software* intermediário presente na especificação do padrão brasileiro de TV digital terrestre chamada Ginga. Segundo informações retiradas do site *Ginga Digital TV Middleware Specification*, o Ginga é o *middleware* desenvolvido pela PUC-Rio e UFPB, de especificação

aberta adotado pelo Sistema Brasileiro de TV Digital Terrestre (SBTVD-T) integrado em conversores (*set-top boxes*) externos e embarcado em televisores. Esse sistema tem duas funções principais: uma é tornar as aplicações independentes do sistema operacional da plataforma de hardware utilizados; a outra é oferecer melhor suporte ao desenvolvimento de aplicações. Ou seja, o Ginga será o responsável por dar suporte à interatividade. O Ginga possui dois sistemas interligados: Ginga-NCL e o Ginga-J e com isso suporta dois paradigmas de programação (SOARES, 2009).

Além do *middleware*, a linguagem NCL foi utilizada para o desenvolvimento das aplicações junto com a linguagem de *script* Lua.

NCL é uma linguagem declarativa, um XML que segue o padrão NCM (Nested Contex Model), que especifica documentos de hipermídia onde é possível descrever objetos de mídia e seu comportamento temporal e espacial (SOARES, 2006)

Lua é uma linguagem de script desenvolvida pela PUC-Rio e que foi escolhida para atuar junto ao NCL complementando a gama de possibilidades de criação de aplicações para a TV Digital (The Programming Language Lua, 2010).

1.2 Identificação do problema

Atualmente as aplicações desenvolvidas para a TV digital ainda são poucas, porém com o avanço dessa nova tecnologia, diversos sistemas surgirão, cada um oferecendo novas funcionalidades, interatividade, informação e entretenimento, um leque de possibilidades se abrindo e gerando *softwares*.

Não foram encontradas informações que definissem se todas essas novas aplicações deverão prover certo controle a ponto de ser possível definir seu posicionamento na tela, até mesmo se estarão presentes em determinado momento. O acionamento das aplicações deve ser de responsabilidade e controle do usuário assim como a possibilidade de utilizar mais de uma aplicação simultaneamente.

Os *gadgets* devem ser independentes ao canal sintonizado e podem ser adquiridos de várias fontes, incluindo o canal de retorno.

Existem normas da ABNT, como a ABNT NBR 15606-2 (2007), que especificam o SBTVD-T e definem o que é possível ser implementado no *middleware*, porém, não foi encontrado, nesse estudo, nenhum padrão ou especificação de arquitetura das aplicações, deixando margens a um desenvolvimento independente que pode dificultar a criação de softwares interativos para a TV digital.

Diante desse contexto fica evidente a falta de um padrão de desenvolvimento e a falta de criação de aplicações que oferecem interatividade no universo televisivo. Mecanismos de controle e gerenciamento ainda não fazem parte da realidade das especificações de criação de softwares, porém, mostram-se necessários. Para isso, propostas de formas, modelos e padrões são bem vindos em quantidade e intensidade para que sejam avaliados, testados, aprovados ou não, amadurecendo o ambiente e de certa forma facilitando o desenvolvimento.

1.3 Objetivos

1.3.1 Objetivo Geral

Propor e desenvolver um sistema que incorpore e gerencie novas aplicações desenvolvidas (*gadgets*) para a TV Digital.

1.3.2 Objetivos Específicos

- Realizar um estudo sobre a TV digital brasileira, sobre seus padrões, especificações de desenvolvimento, plataformas e *middleware*;
- Expor as linguagens de programação suportadas pelo *middleware* Ginga: NCL e Lua;
- Especificar por meio de um modelo conceitual o software proposto;
- Desenvolver a aplicação segundo a especificação e modelagem conceitual criada.

1.4 Justificativa

Diante desta nova tecnologia de sistema televisivo, surgem inúmeras possibilidades de se promover interatividade entre a TV Digital e o telespectador. A informação e o entretenimento podem ser trazidos à tela na forma de *softwares* desenvolvidos, seguindo um determinado padrão (SOARES; BARBOSA, 2009). A TV Digital abrangendo a área de desenvolvimento de *software* abre novos caminhos e possibilidades, com isso surgiu o interesse por essa nova tecnologia, com o propósito de abordar algo inovador, que pudesse render novos estudos e não apenas reproduzir estudos já feitos.

Foi pensado na criação de um modelo de gerenciador que incorpore principalmente aplicações do tipo *gadgets*, com o objetivo de trazer organização, personalização e melhor aproveitamento da interatividade, assim como um modelo de desenvolvimento dos próprios *gadgets* e as funções na qual se propõem dentro do ambiente da TV digital.

Tendo em vista que não foram encontrados ao longo do estudo, padrões de gerenciamento dessas aplicações, identificou-se a necessidade de criação e implementação de um modelo que definisse uma estrutura de desenvolvimento. Essa estrutura garantiria a existência desses padrões e a especificação de como criar tanto um *software* de gerenciamento quanto os utilitários a serem gerenciados.

1.5 Limitações

Pela dificuldade em obter um ambiente de rádio difusão de sinal digital televisivo que transmitisse sinal digital no padrão brasileiro ISDB-TB, não foi possível criar a situação de transmissão de um canal de televisão e fluxos de dados para a execução do *software* proposto. Em conseqüência dessa dificuldade, foi utilizada uma ferramenta de simulação que faz a execução das aplicações sem a transmissão de sinal digital. Com isso, não faz parte da aplicação proposta o tratamento para torná-la independente do canal sintonizado, tornando-a uma aplicação residente dentro do *middleware* Ginga.

Não foi utilizada a linguagem Java, que é especificada para o Ginga-J, porém não suportada pelo Ginga-NCL Virtual STB, ambiente no qual puderam ser feitos os testes e a execução do *software*.

Na implementação do TV Digital Deck não foi tratado posicionamento dos *gadgets* nas regiões pré-definidas da tela da TV.

1.6 Organização do trabalho

O capítulo 1 apresenta objetivos, justificativas e limitações da proposta de modelo e desenvolvimento de uma aplicação para a TV Digital, bem como pequena definição do *middleware* Ginga, da linguagem NCL, e dos *gadgets* que foram utilizados para a execução da aplicação. O capítulo 2 aborda um pequeno estudo sobre a TV Digital, sua definição e tecnologias envolvidas, tratando também sobre o Ginga, uma inovação nacional que será o *middleware* do Sistema Brasileiro de TV Digital Terrestre (SBTVD). O capítulo 3 trata de definições e exemplos práticos das linguagens de programação NCL e Lua, que estão dentro da especificação do *middleware* Ginga. O capítulo 4 consiste na proposta de um modelo conceitual de desenvolvimento de uma aplicação para a TV Digital que incorpore *gadgets*. O capítulo 5 mostra o desenvolvimento e as limitações da aplicação proposta, feito em NCL e Lua. Nos capítulos 6 e 7 são feitas as considerações finais, sugestões para trabalhos futuros e, por fim, as referências consultadas.

2 TV DIGITAL

2.1 Definição

TV Digital é um sistema que vai substituir a TV Analógica oferecendo melhor qualidade de imagem e som. Diferente da versão analógica, a TV Digital não terá “chuviscos” na imagem causados por eventuais ruídos na transmissão. Os ruídos são interferências no sinal, que não vão deixar de existir, porém sua influência na transmissão digital é tratada por códigos corretores de erro especificados nos padrões de transmissão. Essa correção depende de um limiar onde o código corretor é capaz de retificar todos os erros abaixo desse limite, se a quantidade de erros ultrapassar essa taxa, o código transmitido ainda conterá erros inviabilizando a recepção. A partir disso: ou se tem um sinal perfeito ou nenhum sinal (TELECO, 2010).

A qualidade do áudio transmitido também é percebida com a possibilidade de se reproduzir áudio no padrão 5.1 (multicanal) propiciando sensação de imersão na cena transmitida. Com a tecnologia de transmissão e recepção digital, maior quantidade de conteúdo pode ser transmitida em uma mesma freqüência, sendo assim, além de vídeo e áudio, dados também são transmitidos, expandindo as possibilidades e funções desse sistema, criando uma capacidade computacional e surgimento de serviços por aplicações construídas (SOARES; BARBOSA, 2009).

2.2 Tecnologias envolvidas

O sistema de televisão digital possui padrões, com definições e especificações técnicas para melhor implantação do sistema. Alguns padrões definidos para a TV Digital são: Norte Americano/ATSC, Europeu/DVB-T, Japonês/ISDB-T. O padrão conhecido como nipo-brasileiro é resultado da adoção do padrão japonês com a incorporação de tecnologias desenvolvidas por pesquisadores brasileiros deixando ainda mais moderno. Neste caso passou a se chamar de SBTVD-T ou ISDB-TB.

Essas tecnologias nacionais são frutos de projetos desenvolvidos por Universidades como a Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), a Universidade Federal da Paraíba (UFPB) e outras instituições.

Esse padrão especifica os métodos que serão utilizados para compressão de áudio, vídeo, fluxo de transporte, modulação, aplicativos interativos e do *middleware*. O ISDB-TB possui como características também a multiprogramação, na qual, num mesmo canal é possível transmitir um ou mais programas em alta resolução (SOARES; BARBOSA, 2009).

Permite, também, a recepção em dispositivos móveis, como celulares, e ainda propõe um sistema de interatividade através do *middleware* Ginga que é 100% nacional. Toda essa tecnologia envolvida age desde a transmissão do sinal televisivo até sua recepção em aparelhos prontos para a TV Digital ou conversores que vão fazer os tratamentos necessários e oferecer os mais diversos recursos proporcionados por esse novo sistema.

2.2.1 Codificação de áudio e vídeo

Com o intuito de se obter o máximo de ganho na transmissão e assim reduzir a quantidade de bits gerados em um sinal digital, medidas de compressão da informação são adotadas. Essa compressão é feita eliminando bits do sinal, quando essa eliminação gera perdas de informação, diz-se que é uma *compressão com perdas* (SOARES; BARBOSA, 2009). Em um sistema de TV Digital são empregados no áudio técnicas de compressão que visam reduzir ao máximo essas perdas de informação, adotando como parâmetro a capacidade audível do ser humano, onde determinadas freqüências podem ser removidas e não afetarão a qualidade do áudio percebido, havendo então uma *compressão perceptualmente sem perdas* e como resultado um áudio de alta qualidade e baixa taxa de bits gerados (SOARES; BARBOSA, 2009).

O padrão de codificação de áudio adotado pelo sistema brasileiro de TV Digital é o MPEG-4 que absorve muitas funcionalidades do MPEG-1 e MPEG-2 e possui uma taxa de amostragem de 48khz. No quadro 1 é apresentada a codificação de áudio do SBTVD-T. Na codificação do vídeo é usado o padrão MPEG-4 AVC (*Advanced Video Coding*) que elimina a redundância de bits gerados conforme a capacidade visual do ser humano. O MPEG-4 AVC age em quadros que compõem o vídeo, temporalmente (*interquadros*), baseado em quadros passados e futuros de forma preditiva, ou, diretamente (*intraquadros*) (SOARES; BARBOSA, 2009). No quadro 2 é apresentada a codificação de vídeo do SBTVD-T.

	Receptores Fixos e Móveis	Receptores Portáteis
Padrão	ISO/IEC 14496-3 (MPEG-4 AAC)	ISO/IEC 14496-3 (MPEG-4 AAC)
Nível e Perfil	AAC@L4 (para multicanal 5.1) HE-AAC v1@L4 (para estéreo)	HE-AAC v2@L3 (dois canais)
Taxa de amostragem	48khz	48khz

Quadro1: Codificação de áudio no sistema brasileiro de TV digital terrestre

Fonte: Adaptado de Soares; Barbosa (2009).

	Receptores Fixos e Móveis	Receptores Portáteis
Padrão	ITU-T H.264 (MPEG-4 AVC)	ITU-T H.264 (MPEG-4 AVC)
Nível e Perfil	HP@L4.0	BP@L1.3
Número de linhas do nível	48khz	48khz
Taxa de quadros	30 e 60 Hz	15 e 30 Hz

Quadro 2: Codificação de vídeo no sistema brasileiro de TV digital terrestre

Fonte: Adaptado de Soares; Barbosa (2009).

São utilizados dois perfis de compressão do vídeo, um alto para receptores fixos e um perfil base para os receptores portáteis.

2.2.2 Transporte

Todo o áudio, vídeo e dados devem ser multiplexados e transmitidos em um único fluxo. Esse fluxo utiliza o padrão de multiplexação MPEG-2 *System* que insere informações pertinentes relativas à sincronização do conteúdo. A multiplexação dos dados pode ocorrer de duas formas: transporte *síncrono* (ver figura 1) e *assíncrono* (ver figura 2) (SOARES; BARBOSA, 2009). No serviço de transporte *síncrono* os dados são sincronizados entre si e também com os fluxos de áudio e vídeo, através do paradigma de eixo do tempo (*timeline*), onde o contexto da exibição é determinante para que os dados que estão sendo enviados ganhem sentido. Já no serviço de transporte *assíncrono*, essa relação temporal não está presente, pois o tempo de sincronização é indeterminado. Para haver sincronismo entre os dados e o fluxo de áudio e vídeo no serviço *assíncrono*, exige-se uma linguagem de

programação que especifique o sincronismo e seja interpretada pelo receptor (SOARES; BARBOSA, 2009).

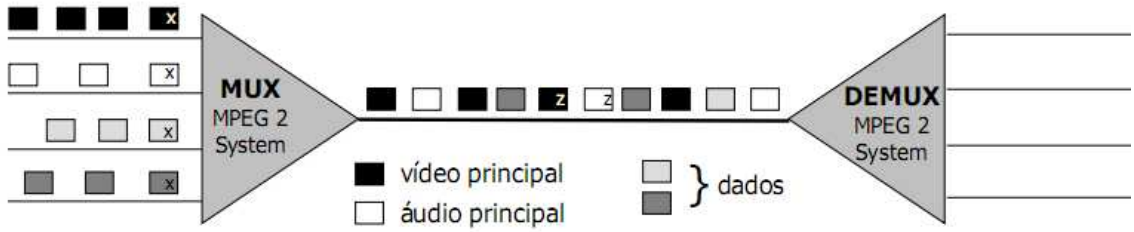

Figura 1: Fluxo de transporte *síncrono*

Fonte: Adaptado de Soares; Barbosa (2009).

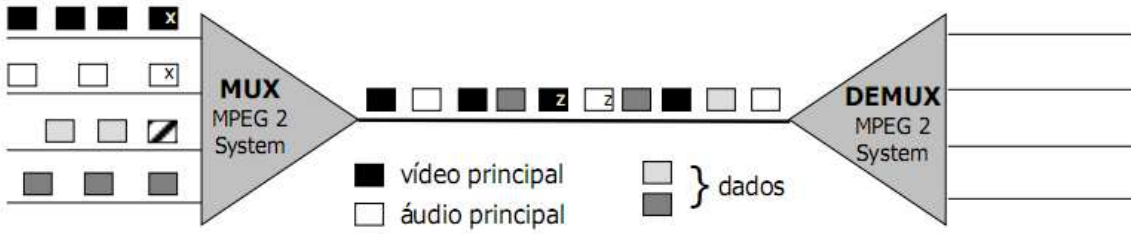

Figura 2: Fluxo de transporte *assíncrono*

Fonte: Adaptado de Soares; Barbosa (2009).

Os dados que não têm relação temporal devem ser enviados de forma cíclica se tornando independentes do momento de sintonização. A sintonização de um canal pode ocorrer a qualquer instante, num envio cíclico os dados tornam-se independentes do instante da sintonização. Esse envio cíclico de dados no sistema de TV Digital é feito através da utilização do protocolo denominado de carrossel de objetos.

> O Carrossel de objetos é de fato um protocolo de transmissão cíclica de dados. O resultado é um fluxo elementar de dados que contém o sistema de arquivos transmitido de forma cíclica. Assim, se um determinado receptor não recebeu um bloco de dados em particular (devido a uma falha na transmissão ou por ter sintonizado o canal após a transmissão desse bloco), basta esperar pela sua retransmissão correta. (SOARES; BARBOSA, 2009, p.17)

2.2.3 Canal de retorno

O receptor pode ter um canal de retorno que permite apenas o envio de dados (*unidirecional*), criando um nível de interatividade simples como a solicitação da compra de um determinado produto apresentado. Esse canal pode ser também *bidirecional*

possibilitando fazer carregamento de dados utilizados pelas aplicações. Há outro nível que permite a navegação na WEB e, por fim, um que possibilita o envio de dados em banda larga chamado de *interatividade plena*, no qual os receptores podem trocar dados entre si atuando como pequenas emissoras. Por meio dos seus padrões de referência, o sistema brasileiro de TV Digital possibilita, então, esses quatro níveis de interatividade (SOARES; BARBOSA, 2009).

2.2.4 *Middleware*

Middleware é um programa de computador que faz a mediação, entre o sistema operacional e a aplicação. Um *middleware* para a TV Digital interativa oferece uma gama de funções que permite o desenvolvimento facilitado de aplicações e suporte a múltiplos dispositivos de exibição “Do ponto de vista do software, podemos dizer, sem exagero, que ao definir o *middleware* estamos, de fato, definindo um sistema de televisão” (SOARES; BARBOSA, 2009, p.22). É uma nova camada que segue os padrões de referência da TV Digital fornecendo através de uma interface de programação, suporte às aplicações a serem executadas no ambiente de TV Digital. O *middleware* estará presente nos receptores digitais independentemente da plataforma de hardware e software.

2.2.4.1 Ginga

2.2.4.2 Histórico

Ginga é o *middleware* do Sistema Brasileiro de TV Digital Terrestre (SBTVD-T), desenvolvido pelos laboratórios Telemídia da PUC-Rio e LAViD da UFPB, é uma inovação brasileira avançada e a melhor solução para os requisitos do país.

2.2.4.3 Definição

Uma camada de software que está sendo instalada nos conversores e em televisores, funciona independentemente do sistema operacional da plataforma de hardware utilizado. O Ginga é uma especificação aberta que adota a licença GPLV2. O *middleware* oferecerá suporte ao desenvolvimento de aplicações interativas para a TV Digital com uma gama de funções que permitem o desenvolvimento facilitado de aplicações. Essas aplicações podem possibilitar, por exemplo, acesso à internet, operações bancárias, entre outras operações (GINGA DIGITAL TV MIDDLEWARE SPECIFICATION, 2009).

2.2.4.4 Composição

O Ginga é dividido em três subsistemas interligados: Ginga-CC, Ginga-NCL e Ginga-J (SOARES, 2009).

A. Ginga-CC (Ginga Common-Core)

Oferece suporte para os ambientes *declarativo* (Ginga-NCL) e *procedural* (Ginga-J) tratando da exibição dos objetos de mídia como JPEG, MPEG-4, MP3, GIF, entre outros formatos. Controla diversas camadas inerentes ao hardware bem como acesso à rede e ou canal de retorno (SOARES, 2009).

B. Ginga-NCL

Provê infra-estrutura para execução de aplicações desenvolvidas na linguagem declarativa NCL (Nested Context Language) e suporte para a linguagem de script Lua em conjunto com o NCL. Também pode referenciar aplicações desenvolvidas em Java que são executadas pelo Ginga-J através de uma ponte entre os dois subsistemas que permite tal comunicação (SOARES, 2009).

C. Ginga-J

Processa as aplicações desenvolvidas em Java e extensões especificamente voltadas ao ambiente de TV. As aplicações Java podem se relacionar com aplicações NCL de várias formas através de APIs. Essas APIs estão divididas em verdes, amarelas e vermelhas. As APIs verdes são responsáveis por manter o máximo possível a compatibilidade com os sistemas americano e europeu. APIs especificas do Ginga que podem ser exportadas para outros sistemas são as chamadas amarelas e o suporte para as necessidades de aplicações voltadas para o Brasil são endereçadas pela API vermelha (SOARES, 2009).

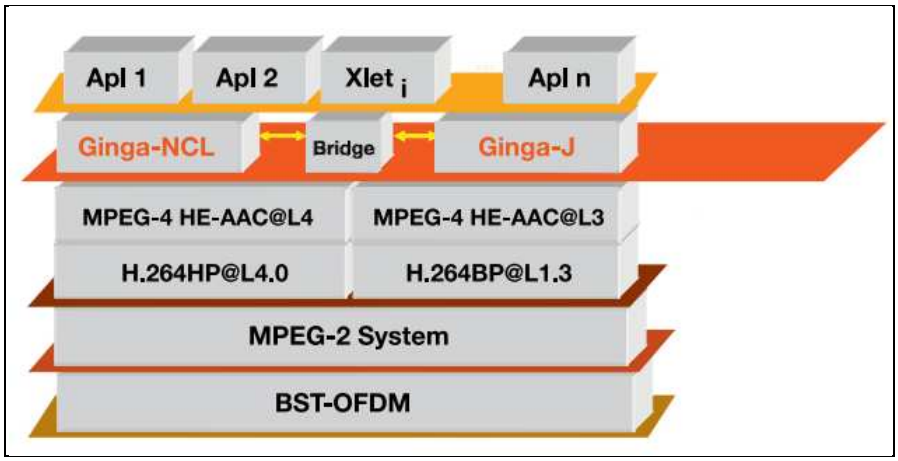

Figura 3: Modelo de referência do SBTVD-T

Fonte: adaptado de Soares (2009).

3 LINGUAGENS DE PROGRAMAÇÃO

O Sistema Brasileiro de TV Digital Terrestre (SBTVD-T) tem como uma das principais inovações o *middleware* Ginga. Ele provê suporte para o desenvolvimento de aplicações em NCL, Lua e Java. Nesse capítulo serão apresentados definições e exemplos práticos dessas linguagens com exceção de Java.

3.1 NCL

3.1.1 Histórico

A linguagem NCL foi desenvolvida pelo laboratório Telemídia da PUC-Rio para a criação de aplicativos de forma declarativa, fazendo parte da especificação Ginga-NCL juntamente com a linguagem Lua.

3.1.2 Definição

NCL (*Nested Context Language*) é uma linguagem declarativa, um XML seguindo o padrão NCM (*Nested Context Model*), que especifica documentos de hipermídia onde é possível descrever objetos de mídia e seu comportamento temporal e espacial. Por objeto de mídia entende-se texto, imagem, áudio, vídeo, objetos com código Lua, objetos com código Java, objetos com código HTML etc (SOARES, 2006). Existem algumas vantagens por ser uma linguagem declarativa como facilidade de criação não requerendo lógica de programação, alguns limites, porém, existem no que diz respeito, por exemplo, a um processamento matemático ou manipulação de textos. Para resolver essa limitação, adiciona-se outra linguagem como suporte imperativo, para ser usada quando necessária essa outra linguagem que supre as limitações do NCL é a de *script* Lua (SOARES, 2006).

NCL é uma poderosa linguagem de cola com suporte a relacionamentos e de fácil composição. O desenvolvimento de novas mídias escritas puramente em NCLua leva a aplicabilidade de NCL a um novo patamar de opções (SANT´ANNA; CERQUEIRA; SOARES, 2006).

3.1.3 Modelo Conceitual NCM

A NCL sendo uma linguagem declarativa é baseada em um modelo conceitual de dados, o NCM, que representa os conceitos estruturais, os relacionamentos, regras estruturais e operações.

Basicamente compõe-se de um *nó* NCM que possui um identificador, um conteúdo e um conjunto de âncoras. Uma âncora é um subconjunto das unidades de informação de um *nó* e são definidas separadamente do conteúdo de um *nó*. Um *nó de mídia* representa um *objeto de mídia* (SOARES; BARBOSA, 2009).

3.1.3.1 Estrutura de um documento NCL

Inicialmente e de caráter obrigatório, um documento NCL possui um cabeçalho de arquivo que especifica um documento XML com a *tag* introdutória e mais um único elemento raiz. A estrutura consiste em: um cabeçalho do programa, o corpo e a conclusão do documento, conforme figura 4 (SOARES; RODRIGUES; BARBOSA, 2006).

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!-- Generated by NCL Eclipse -->
3  <ncl id="exemplo" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
4      <head>
5
6      </head>
7
8      <body>
9
10     </body>
11 </ncl>
```

Figura 4: Modelo de documento NCL

Fonte: Desenvolvido pelo autor.

A. Cabeçalho

Dentro da *tag* <*head*></*head*> do documento NCL são definidas as bases de regiões, base de descritores e base de conectores.

B. Base de Regiões

Na base de regiões são definidas as regiões da tela onde os objetos de mídia serão apresentados. Essas regiões são declaradas inicialmente com a *tag* <*regionBase*></*regionBase*> e para cada nova região do documento a *tag* <*region*></*region*> definida como filha do elemento <*regionBase*>. O elemento <*region*> também pode ter outras *tags* <*region*></*region*> dentro dele. As regiões possuem identificadores e diversos atributos que definem a área de exibição, o tamanho e o posicionamento que podem ser em valores absolutos ou em porcentagens e sempre em relação

à região pai. Com exceção do atributo *zIndex* que define como ficará a sobreposição de regiões, onde o componente com maior valor do atributo *zIndex* sobrepõe o de menor valor.

Na figura 5 apresenta um exemplo onde existe uma região definida que cobre 100% da tela da televisão e uma segunda região filha da primeira e de menor tamanho.

```
6        <regionBase>
7            <region id="regiao1" width="100%" height="100%" zIndex="1">
8                <region id="regiao2" width="40" height="80" zIndex="2"/>
9            </region>
10       </regionBase>
```

Figura 5: Base de regiões

Fonte: Desenvolvido pelo autor.

C. Base de Descritores

Na base de descritores são criados os descritores para os objetos de mídia. O conjunto de descritores é definido como filho do elemento <*descriptorBase*> e escrito a partir da *tag* <*descriptor*></*descriptor*> possuindo um identificador e referenciando a região a ser utilizada, além das características iniciais, para a exibição do objeto de mídia, como: duração do objeto, possíveis efeitos de transição, entre outros. A figura 6 descreve um exemplo de descritores.

```
12       <descriptorBase>
13           <descriptor id="desc1" region="regiao1"/>
14           <descriptor id="desc2" region="regiao2"/>
15       </descriptorBase>
```

Figura 6: Base de descritores

Fonte: Desenvolvido pelo autor.

D. Base de conectores

Na base de conectores <*connectorBase*> são apresentados os conectores <*causalConnector*> ou pode-se referenciar um documento externo que deve ser importado através da *tag* <*importBase*></*importBase*> filha do elemento <*connectorBase*> que conterá os conectores que serão usados no documento. (Ver figura 7).

```
17        <connectorBase>
18            <importBase documentURI="causalConnBase.ncl" alias="conEx"/>
19        </connectorBase>
```

Figura 7: Importação de uma base de conectores

Fonte: Desenvolvido pelo autor.

Os conectores vão definir como os elos são ativados e o que eles disparam. Para declarar um conector <*causalConnector*> tem-se por obrigatoriedade declarar como filho ao menos uma *tag* de condição <*simpleCondition*> e uma *tag* de ação <*simpleAction*></*simpleAction*>. As condições podem depender de parâmetros e ser simples ou compostas de outras condições. As ações também podem depender de parâmetros e serem sequenciais ou conjuntas. O exemplo apresentado na figura 8 define que o conector ao ser iniciado executará a ação de *start* sob a condição de início (*onBegin*).

```
17        <connectorBase>
18            <causalConnector id="onBeginStartDelay">
19                <connectorParam name="delay"/>
20                <simpleCondition role="onBegin"/>
21                <simpleAction role="start" delay="$delay"/>
22            </causalConnector>
23        </connectorBase>
```

Figura 8: Base de conectores

Fonte: Desenvolvido pelo autor.

Ainda no cabeçalho do documento NCL, também podem ser definidas bases de efeitos de transição, metadados, importação de outros documentos e regras de alternativas para apresentação do documento.

E. Corpo

Dentro da *tag* <*body*></*body*> do documento NCL são definidos o ponto de entrada do programa e o conteúdo.

O ponto de entrada do programa é definido pela *tag* <*port*></*port*> que deve possuir um identificador e o atributo *component* referenciando o objeto de mídia ou o contexto a ser iniciado. No momento de execução do documento utiliza-se o ponto de entrada que aponta para o primeiro objeto a ser apresentado. A figura 9 mostra um exemplo de declaração de ponto de entrada apontando para um objeto de mídia.

```
110         <port id="entry" component="animation"/>
```

Figura 9: Ponto de entrada

Fonte: Desenvolvido pelo autor.

O conteúdo do programa pode ser definido por *tags* de objetos de mídia <*media*></*media*>**,** contextos <*context*></*context*> e elos <*link*></*link*>.

Os objetos de mídia podem ser áudio, vídeo, texto, imagem, objetos com código Lua, objetos com código Java, objetos com código HTML, etc. e são declarados através da *tag* <*media*></*media*> que deve possuir: um identificador, um atributo *src* com o URI (Uniform Resource Identifier) do objeto de mídia e um atributo *descriptor* referenciando o descritor a ser utilizado para esse objeto. (Conforme figura 10)

```
34         <media id="animation" src="media/animGar.mp4" descriptor="desc1"/>
```

Figura 10: Objeto de mídia

Fonte: Desenvolvido pelo autor.

Os elos <*link*> definem relacionamentos do conteúdo do documento. Atribui-se aos objetos de mídia uma relação dada pelo valor URI do atributo *xconnector* do elemento <*link*> que referencia um conector da base de conectores com suas condições e ações. Os elementos <*bind*> especificam o papel da relação através de seus atributos *role*, esse papel está especificado no conector podendo ser uma condição ou ação. Também selecionam os objetos a serem relacionados através do atributo *component*. A figura 11 mostra um exemplo de elo mapeando uma condição e ação e relacionando com objetos de mídia.

```
39          <link id="lMusic" xconnector="onBeginStartDelay">
40              <bind role="onBegin" component="animation"/>
41              <bind role="start" component="background">
42                  <bindParam name="delay" value="5s"/>
43              </bind>
44          </link>
```

Figura 11: Elo

Fonte: Desenvolvido pelo autor.

Os contextos <*context*> geralmente são usados para estruturar a aplicação, agrupando elementos do conteúdo. Assim como os objetos de mídia e elos, os contextos também devem possuir um identificador e todos os elementos a serem agrupados para o contexto devem ser

declarados como filhos desse elemento. A figura 12 mostra um exemplo de contexto agrupando alguns elementos.

```
34        <context id="contextoExemplo">
35            <port id="entry" component="animation"/>
36            <media id="background" src="media/background.png" descriptor="desc2"/>
37            <media id="animation" src="media/animGar.mp4" descriptor="desc1"/>
38            <link id="lMusic" xconnector="onBeginStartDelay">
39                <bind role="onBegin" component="animation"/>
40                <bind role="start" component="background">
41                    <bindParam name="delay" value="5s"/>
42                </bind>
43            </link>
44        </context>
```

Figura 12: Contexto

Fonte: Desenvolvido pelo autor.

Esses são os principais elementos da estrutura de um documento NCL, seus atributos obrigatórios, opcionais, porém ainda existem muitos outros atributos e elementos que enriquecem a estrutura e funcionalidade da linguagem.

3.1.4 Criação de um documento NCL

Para o desenvolvimento do documento foi utilizado o IDE (*Integrated Development Environment*) *Eclipse 3.5*, além desse, fez-se necessário a instalação do *plugin* desenvolvido pela UFMA (Universidade Federal do Maranhão) o NCL *Eclipse* que fornece o suporte para criação dos documentos NCL.

Para dar início a criação do documento NCL, foi elaborado um novo projeto no *Eclipse*. Dentro desse novo projeto, criou-se um *NCL document* nomeado *myVideoApp.ncl* onde foi desenvolvida toda a aplicação.

O *myVideoApp.ncl* consiste em uma aplicação desenvolvida em NCL, na qual exibe um vídeo na tela, que pode interromper a exibição pressionando o botão vermelho do controle remoto.

No momento da criação de um *NCL document* através do *Eclipse*, o arquivo gerado já possui a estrutura básica necessária para dar início ao desenvolvimento conforme figura 4 apresentada anteriormente.

3.1.4.1 Cabeçalho de myVideoApp.ncl

As primeiras informações a serem definidas são inseridas no cabeçalho do documento <*head*>. A base de regiões foi declarada com duas regiões, uma para o plano de fundo e outra

para o vídeo a ser exibido como pode ser visto na figura 13. A região do plano de fundo possui o atributo *zIndex* definido como 1 e na região do vídeo como 2, sendo assim quando o vídeo estiver em exibição irá sobrepor o plano de fundo em função do seu maior valor do atributo.

```
6       <regionBase>
7           <region id="backgroundReg" width="100%" height="100%" zIndex="1"/>
8           <region id="screenReg" width="100%" height="100%" zIndex="2"/>
9       </regionBase>
```

Figura 13: Base de regiões de *myVideoApp.ncl*

Fonte: Desenvolvido pelo autor.

Após definidas as regiões do documento, a base de descritores foi criada com dois descritores (ver figura 14), um para o plano de fundo referenciando a região definida e outro para o vídeo também referenciando sua região já definida.

```
11      <descriptorBase>
12          <descriptor id="backgroundDesc" region="backgroundReg"/>
13          <descriptor id="screenDesc" region="screenReg"/>
14      </descriptorBase>
```

Figura 14: Bases de descritores de *myVideoApp.ncl*

Fonte: Desenvolvido pelo autor.

Ao fim do cabeçalho foi acrescentada a base de conectores, com apenas dois conectores. Um dos conectores é responsável por definir que ao iniciar um objeto de mídia (*onBeginStartDelay*), um segundo objeto seja iniciado com um retardo de tempo definido através de um parâmetro, o *delay*. Já o segundo conector (*onEndStop*) é responsável por ao final da execução de um objeto, um segundo objeto também seja encerrado. A figura 15 apresenta como ficou a declaração dos conectores.

```
16        <connectorBase>
17            <causalConnector id="onBeginStartDelay">
18                <connectorParam name="delay"/>
19                <simpleCondition role="onBegin"/>
20                <simpleAction role="start" delay="$delay"/>
21            </causalConnector>
22            <causalConnector id="onEndStop">
23                <simpleCondition role="onEnd"/>
24                <simpleAction role="stop"/>
25            </causalConnector>
26        </connectorBase>
```

Figura 15: Base de conectores de *myVideoApp.ncl*

Fonte: Desenvolvido pelo autor.

O conector *onBeginStartDelay* possui o parâmetro *delay*. Ele possui também, uma única condição determinada como *onBegin* referente a quando a apresentação do objeto ligado a essa condição for iniciado. Por fim, uma única ação *start* para iniciar a apresentação associada a ação utilizando o atributo *delay* que recebe o valor contido no parâmetro indicando o tempo de retardo para a exibição do objeto associado a ação *start* em relação ao objeto ligado a condição *onBegin*.

O conector *onEndStop* é mais simples e possui apenas uma condição e uma ação. A condição indica que ao final da execução de um objeto de mídia, ocorre a ação *stop* que termina a execução de outro objeto de mídia.

3.1.4.2 Corpo de *myVideoApp.ncl*

No corpo do documento NCL, delimitado pela tag <*body*>, foi introduzido o ponto de entrada <*port*> referenciando por meio de seu atributo *component* o objeto de mídia que inicialmente deve ser apresentado ao executar o documento NCL em questão. (Ver figura 9). Na seqüência do corpo os objetos de mídia utilizados (ver figura 16), o plano de fundo nomeado *background* e o vídeo *animation* que será exibido.

```
30    <media id="background" src="media/background.png" descriptor="backgroundDesc"/>
31
32    <media id="animation" src="media/animGar.mp4" descriptor="screenDesc"/>
```

Figura 16: Objetos de mídia de *myVideoApp.ncl*

Fonte: Desenvolvido pelo autor.

Os objetos de mídia possuem como atributos o *id*, *src* e *descriptor*. No atributo *src* está indicado a URI do conteúdo do objeto, no caso do plano de fundo uma imagem no

formato PNG (*Portable Network Graphics*) e para o vídeo uma animação no formato MP4 (MPEG-4). Essas URI tomam como raiz o diretório onde se encontra o documento e podem indicar o caminho dos arquivos de forma relativa, no caso, partindo da raiz para o diretório *media* e por fim o arquivo propriamente dito.

Para finalizar o documento, foi criado um elo <*link*> que define o relacionamento entre o objeto de mídia *background* e *animation* da forma que ao iniciar o objeto *animation* o *background* será iniciado 5 segundos depois.

```
38        <link id="lBack" xconnector="onBeginStartDelay">
39            <bind role="onBegin" component="animation"/>
40            <bind role="start" component="background">
41                <bindParam name="delay" value="5s"/>
42            </bind>
43        </link>
44
45        <link id="lEnd" xconnector="onEndStop">
46            <bind role="onEnd" component="animation"/>
47            <bind role="stop" component="background"/>
48        </link>
```

Figura 17: Elos de *myVideoApp.ncl*

Fonte: Desenvolvido pelo autor.

Como pode ser visto na figura 17, o elo *lBack* faz uso do conector *onBeginStartDelay*, definido na base de conectores, mapeando a regra de condição através da tag <*bind*>, onde o atributo *role* define qual a regra, e o atributo *component* a qual objeto se refere. Em sequência é mapeada a ação com a regra *start* apontando para o objeto *background* e acrescentando o parâmetro *delay* necessário para completar a regra.

O elo *lEnd* (ver figura 17) utiliza o conector *onEndStop*, também presente na base de conectores, no qual, por meio do atributo *role* onEnd da tag <*bind*>, define a regra de condição *onEnd* para o objeto de mídia *animation* e mais a regra de ação *stop* para o objeto *background*, sendo assim ao témino do vídeo o fim da exibição do plano de fundo também ocorrerá.

A figura 18 mostra o documento completo pronto para ser executado.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Generated by NCL Eclipse -->
<ncl id="myVideoApp" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
    <head>

        <regionBase>
            <region id="backgroundReg" width="100%" height="100%" zIndex="1"/>
            <region id="screenReg" width="100%" height="100%" zIndex="2"/>
        </regionBase>

        <descriptorBase>
            <descriptor id="backgroundDesc" region="backgroundReg"/>
            <descriptor id="screenDesc" region="screenReg"/>
        </descriptorBase>

        <connectorBase>
            <causalConnector id="onBeginStartDelay">
                <connectorParam name="delay"/>
                <simpleCondition role="onBegin"/>
                <simpleAction role="start" delay="$delay"/>
            </causalConnector>
            <causalConnector id="onEndStop">
                <simpleCondition role="onEnd"/>
                <simpleAction role="stop"/>
            </causalConnector>
        </connectorBase>

    </head>

    <body>

        <port id="entry" component="animation"/>

        <media id="background" src="media/background.png" descriptor="backgroundDesc"/>

        <media id="animation" src="media/animGar.mp4" descriptor="screenDesc"/>

        <link id="lBack" xconnector="onBeginStartDelay">
            <bind role="onBegin" component="animation"/>
            <bind role="start" component="background">
                <bindParam name="delay" value="5s"/>
            </bind>
        </link>

        <link id="lEnd" xconnector="onEndStop">
            <bind role="onEnd" component="animation"/>
            <bind role="stop" component="background"/>
        </link>

    </body>
</ncl>
```

Figura 18: *myVideoApp.ncl*

Fonte: Desenvolvido pelo autor.

3.1.4.3 Executando myVideoApp.ncl

Para a execução do documento foram utilizadas as ferramentas *VMware*, um software de virtualização que permite a instalação de um sistema operacional dentro de outro, e o Ginga-NCL Virtual STB, uma máquina virtual Linux para VMware contendo o Ginga-NCL

C++ v. 0.11.2 rev.23 (ver figura 19) que com base em um *script* consegue executar documentos desenvolvidos na linguagem NCL.

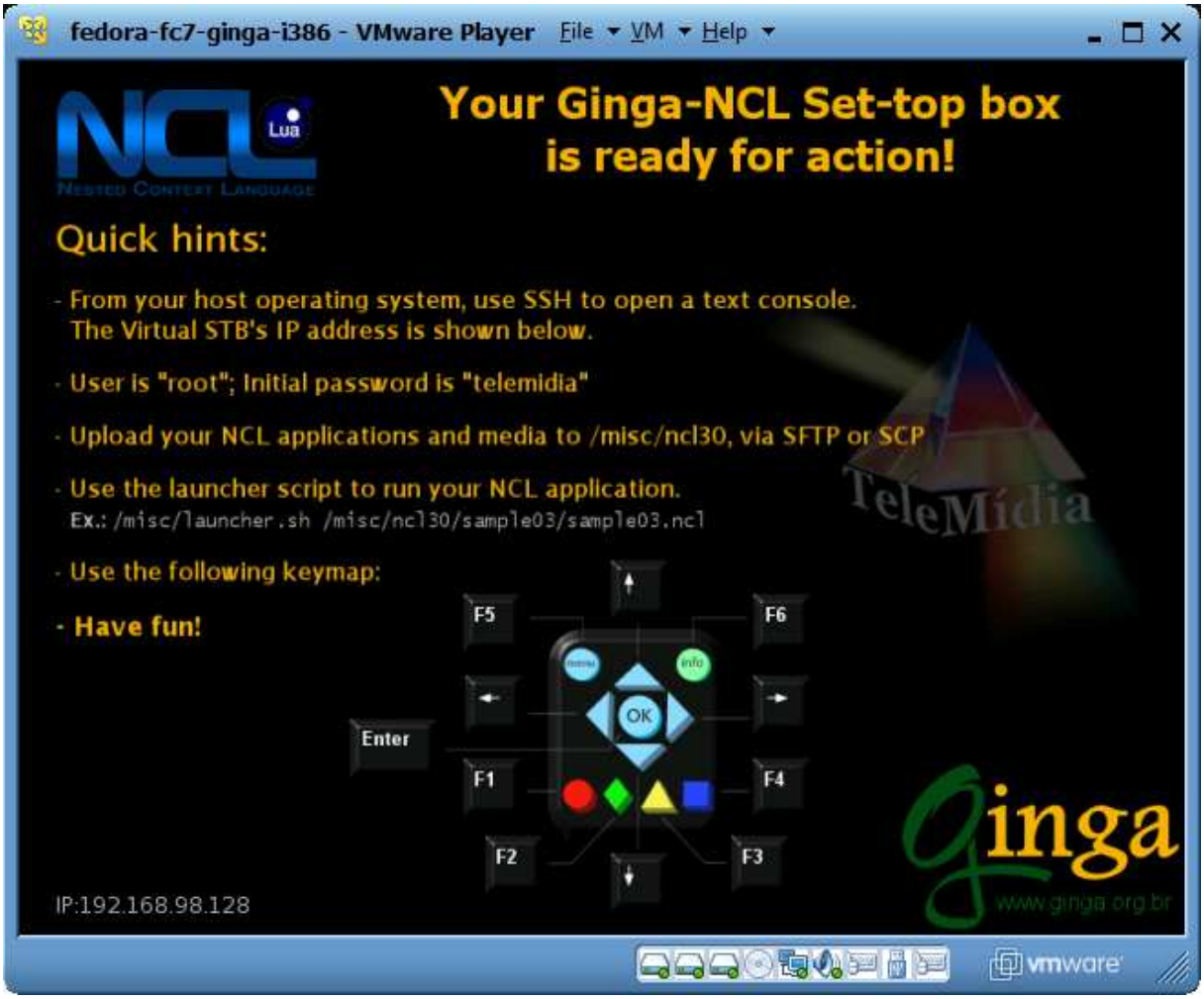

Figura 19: VMware com Ginga-NCL Virtual STB

Fonte: Desenvolvido pelo autor.

Utilizando um cliente de SSH (*Secure Shell*) o *WinSCP* foram transferidos para a máquina virtual os arquivos necessários para a execução do documento. Os arquivos foram: *myVideoApp.ncl*, *animGar.mp4*, *background.png* sendo os dois últimos colocados no diretório */misc/ncl30/media/* e o documento *myVideoApp.ncl* diretamente em */misc/ncl30/*. Conforme mostrado nas figuras 20 e 21.

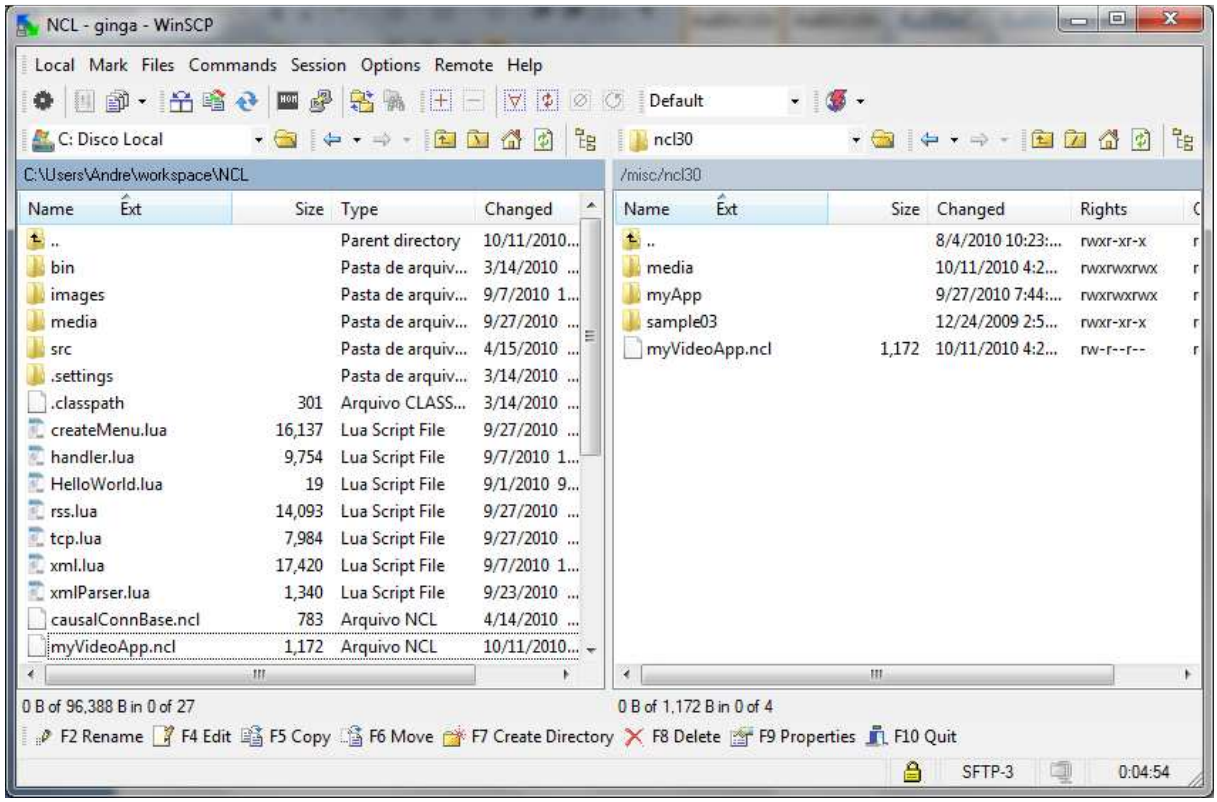

Figura 20: *WinSCP*: transferindo o documento *myVideoApp.ncl*

Fonte: Desenvolvido pelo autor.

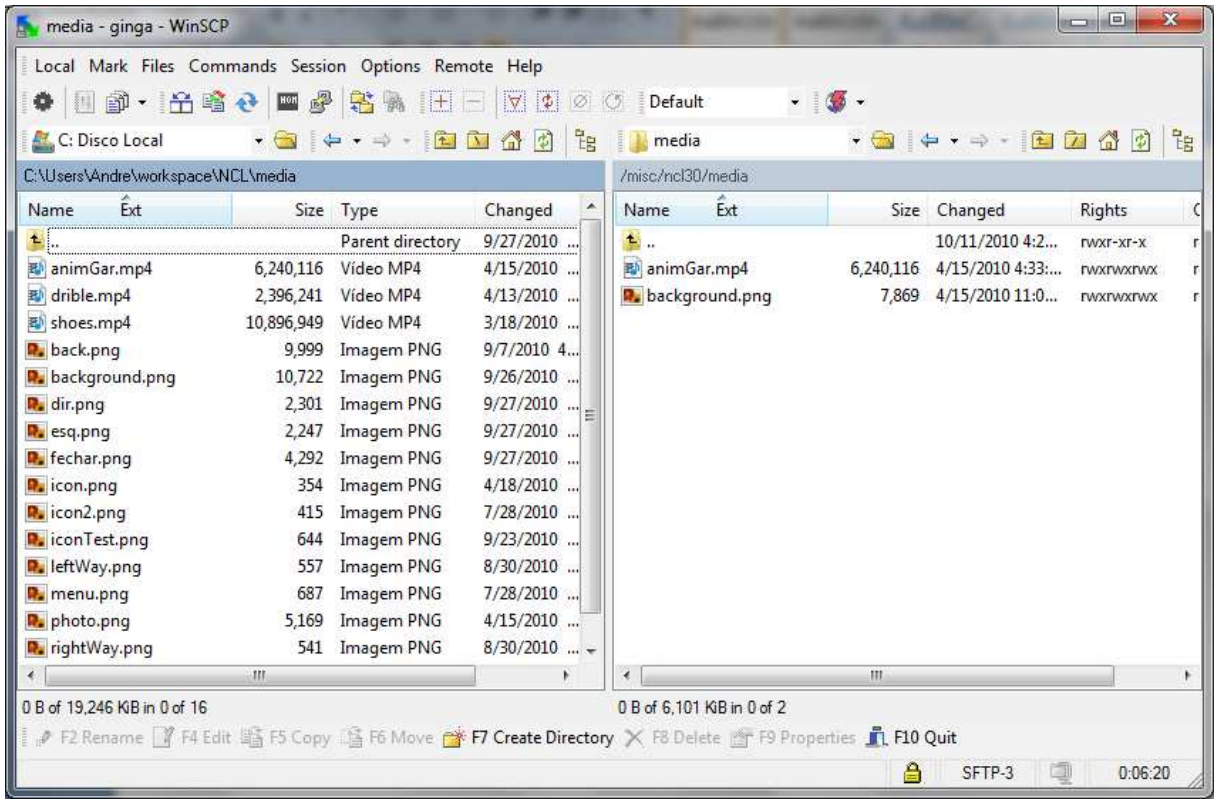

Figura 21: WinSCP: transferindo *animGar.mp4* e *background.png*

Fonte: Desenvolvido pelo autor.

Nessa máquina virtual, dentro do diretório */misc* existe um *script* chamado *launcher.sh* esse *script* é responsável por executar qualquer documento NCL passado como parâmetro.

Após a transferência dos arquivos para o Ginga Virtual STB, o ambiente está pronto para executar o documento NCL criado. Para poder executar o documento, foi feito um acesso via terminal à máquina virtual utilizando a ferramenta PuTTy, um outro cliente SSH que fornece acesso remoto. Aberta a sessão pelo PuTTy é solicitado usuário e senha onde foi utilizado *root* e *telemidia* respectivamente, as informações de usuário e senha estão disponíveis na tela de apresentação do Ginga-NCL Virtual STB conforme figura 19. Dado o acesso, fez-se uso de um comando para executar o documento criado conforme figura 22. Esse comando executa o *script launcher.sh* passando como parâmetro a URI do documento NCL a qual se deseja executar.

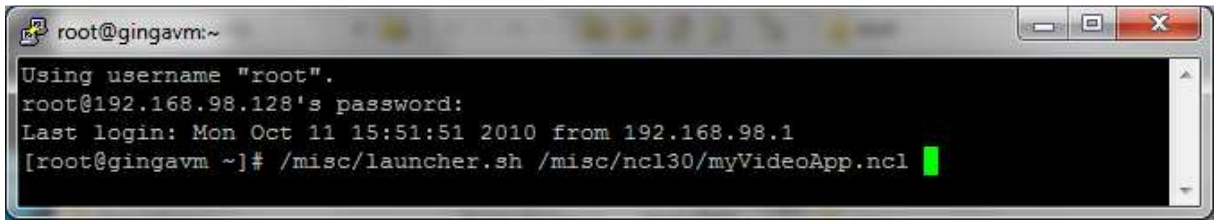

Figura 22: Executando o documento myVideoApp.ncl

Fonte: Desenvolvido pelo autor.

Como resultado da execução de *myVideoApp.ncl* teve a exibição do vídeo até seu final. (Ver figura 23).

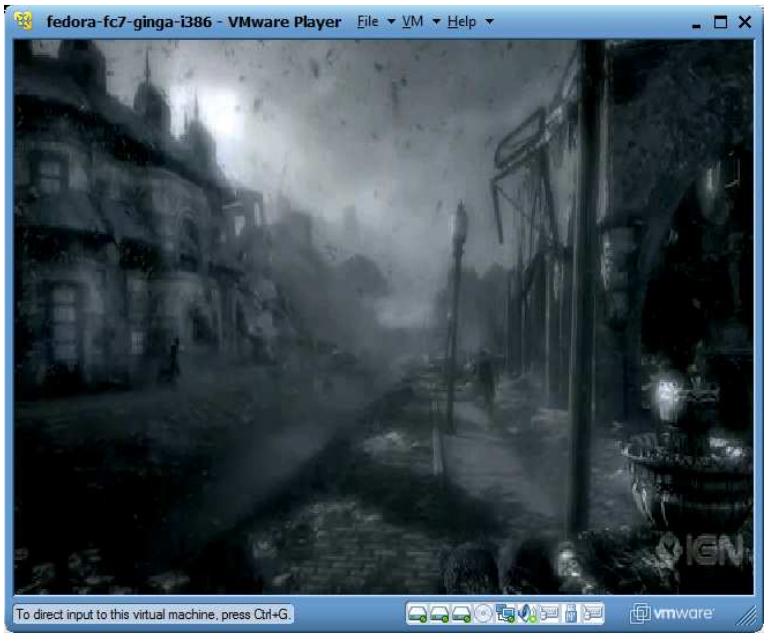

Figura 23: Vídeo animGar.mp4

Fonte: Desenvolvido pelo autor.

Durante toda a execução do documento, o terminal PuTTy exibiu um *log* de dados dos processos de registros dos eventos, conforme figura 24.

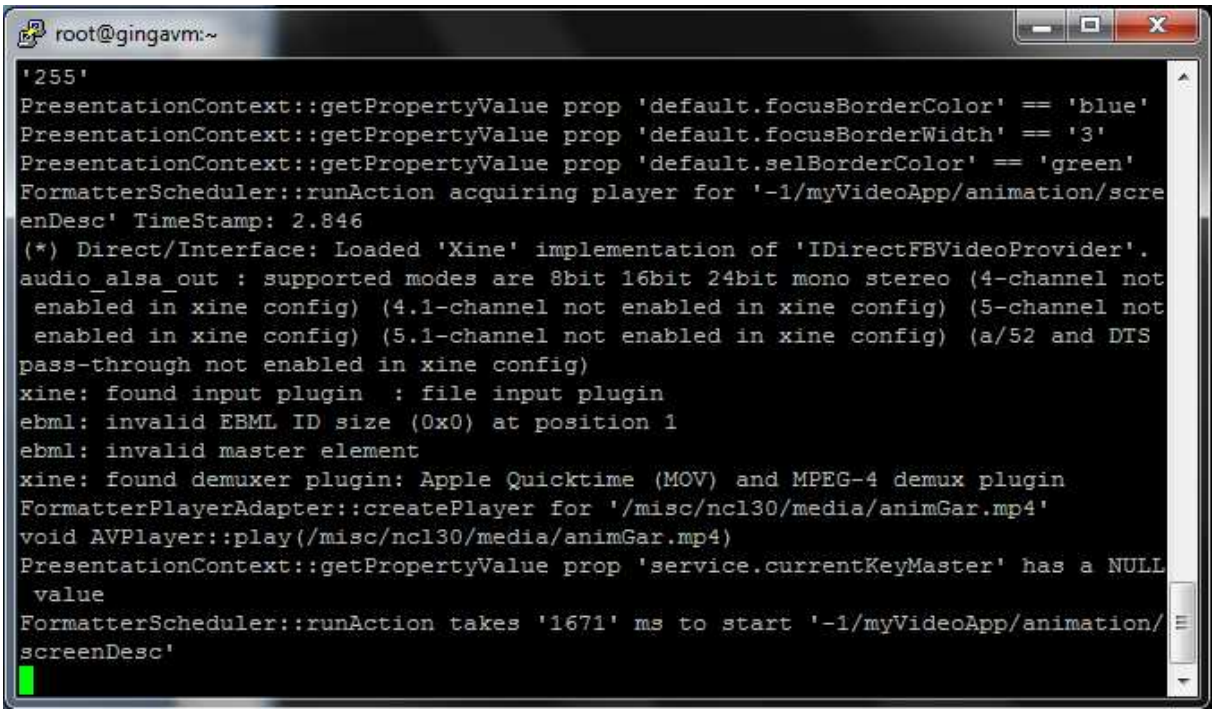

Figura 24: *log* de dados

Fonte: Desenvolvido pelo autor.

Um pouco trabalhoso de se executar uma aplicação no ambiente do Ginga-NCL Virtual STB, porém é a aplicação de simulação e testes mais próxima de uma solução embarcada do *middleware* Ginga que foi encontrada.

3.2 Lua

3.2.1 Definição

Lua é uma linguagem de programação desenvolvida para ser usada em conjunto com outras linguagens com o propósito de estender as funcionalidades da aplicação principal (THE PROGRAMMING LANGUAGE LUA, 2009). Uma linguagem de fácil aprendizado, desenvolvida pela PUC-Rio que tem como características: simplicidade, portabilidade, tipagem dinâmica, semântica extensível, interpretada a partir de *bytecodes* e possui gerenciamento automático de memória com coleta de lixo incremental. Lua é livre de código aberto podendo ser usada para quaisquer propósitos comerciais ou não.

Lua é usada em muitas aplicações industriais com ênfase em sistemas embutidos e jogos. Atualmente a linguagem de script mais usada em jogos (THE PROGRAMMING LANGUAGE LUA, 2009).

3.2.2 Exemplo de código Lua

A simplicidade em se escrever código Lua pode ser observada na figura 25, nas quais três formas diferentes do clássico exemplo de “Hello World” estão representadas.

```
2    -- Hello World
3
4    print "Hello World!"
5
6    print("Hello World!")
7
8    print("Hello World!");
9
```

Figura 25: “Hello World” em Lua

Fonte: Desenvolvido pelo autor.

Ao executar esse exemplo para cada instrução de *print* tem-se como resultado a mesma saída: “Hello World!”.

3.2.3 Tipos

Lua possui oito tipos de dados, são eles:

- Nil: *null* ou *void*;
- Boolean: *true* ou *false*;
- Integer: *float* Integers com precisão dupla;
- String: UTF-98 *strings*;
- Function: funções de primeira ordem;
- Table: tipo principal de Lua;
- Thread: usadas para corotinas;
- User data: ponteiros genéricos de C.

As *Strings* em Lua são imutáveis, não são objetos completos e podem ser construídos sintaticamente de várias formas: declaradas entre aspas simples, aspas duplas ou entre colchetes possibilitando *Strings* com quebra de linha conforme figura 26.

```
2   simples = 'Isso é uma string simples'
3
4   regular = "Isso é uma string normal sem quebra de linha"
5
6   completa = [[Isso é uma string grande.
7                Ela leva em consideração
8                as quebras de linhas.]]
```

Figura 26: *Strings* em Lua

Fonte: Desenvolvido pelo autor.

Strings também possuem funções adicionais através de uma API, entre elas: retornar o valor ASCII, retornar a *String* de um valor ASCII, retornar partes de uma *String*, retornar a ocorrência de determinado caractere, aplicar formatação. A sintaxe dessas funções pode ser observada na figura 27.

```
-- Funções String
string.byte("ABCDE", 2) -- Retorna o valor ASCII
-->   66
string.char(65,66,67,68,69) -- Retorna a String do ASCII
-->   ABCDE
string.find("hello Lua user", "Lua") -- Encontra a substring "Lua"
-->   7         9
string.find("hello Lua user", "l+") -- Encontra as ocorrências de 'l'
-->   3         4
string.format("%.7f", math.pi) -- Formata um número
-->     3.1415927
string.format("%8s", "Lua") -- Formata uma String
-->         Lua
```

Figura 27: Funções de *String*

Fonte: Desenvolvido pelo autor.

As funções podem receber qualquer número de parâmetros, retornarem qualquer quantidade de valores e até outra função. Um exemplo de *function* pode ser visto na figura 28.

```
-- Calcula fibonacci.
function fib(n)
   local a, b = 0, 1
   while n > 0 do
      a, b = b, a + b
      n = n - 1
   end
   return a
end

-- Executa fibonacci.
for i = 1,10 do
   print( fib( i ) )
end
```

Figura 28: *function* Lua

Fonte: Desenvolvido pelo autor.

As tabelas são as únicas estruturas de dados presentes em Lua e por conta disso são as mais importantes e complexas. São a base de todos os tipos criados.

São conjuntos de pares (chave e valor) de dados (também conhecido como *Hashed Heterogeneous Associative Array*), na qual os dados são referenciados por chaves. A chave (índice) pode ser de qualquer tipo de dado exceto *nil*.

Em Lua, tabelas são criadas usando a seguinte sintaxe. Ver figura 29.

```
2   -- Tabelas
3
4   table = {} -- cria uma tabela vazia
5
6   tabela = {"a", "b", "c"} -- cria uma tabela com as valores a, b e c
7
8   x = "chave" -- declaração da string x
9   array = {x = 10} -- cria uma tabela hash onde a string x é a chave e o inteiro 10 o valor
10  print(array[x]) -- vai imprimir o valor que possui a chave x, no caso o inteiro 10
11
```

Figura 29: *tables* em Lua

Fonte: Desenvolvido pelo autor.

Essa estrutura de dados também possui funções adicionais, entre elas: inserção, remoção, ordenação e concatenação. A sintaxe dessas funções pode ser observada na figura 30.

```
2   -- Funções de tables
3
4   t = { "a", "b", "c"} -- cria uma tabela com os valores a, b, c
5
6   table.insert(t, "d") -- insere na tabela o valor d
7
8   table.remove(t, 2) -- remove da tabela o valor contido no indice 2 (b)
9
10  table.sort(t, function(a,b) return a < b end) -- ordena a tabela atraves de uma função
11
12  table.concat(t, "\n") -- concatena uma quebra de linha para cada valor da tabela
13
```

Figura 30: Funções de *tables*

Fonte: Desenvolvido pelo autor.

Existem outros tipos mais complexos de tabelas em Lua, o que contribui para a sua potencialidade como linguagem de *script*.

3.3 Lua e NCL

Em função de suas características, Lua foi a linguagem de *script* escolhida para atuar junto à linguagem NCL complementando-a.

Além das API (Application Programming Interface) disponíveis da linguagem Lua, quatro novos módulos existem para os NCLua (objetos NCL com código Lua):

- *event*: permite a comunicação de objetos Lua com documentos NCL;
- *canvas*: agrega funções para desenho gráfico na região do NCLua;
- *settings*: oferece acesso às variáveis definidas no objetos *settings* do documento NCL;

- *persistent*: exporta uma tabela com variáveis persistentes entre objetos imperativos.

Segundo a norma ABNT NBR 15606-2 (2007), algumas funções são dependentes de plataforma e com isso não são de implementação obrigatória:

- No módulo *package*: *loadlib* é opcional;
- No módulo *io*: todas as funções são opcionais;
- No módulo *os*: *clock*, *execute*, *exit*, *getenv*, *remove*, *rename*, *tmpname* e *setlocale* são opcionais;
- No módulo *debug*: todas as funções são opcionais, a API C de *debug* também é opcional.

O ciclo de vida de um objeto Lua é controlado pelo documento NCL, porém, diferentemente das mídias, os *scripts* são guiados por eventos, que podem ser oriundos do NCL ou de fontes externas de interação como o controle remoto. O documento NCL pode definir em que momento o *script* vai ser executado, podendo encerrar no tempo determinado ou com algum evento de encerramento ou até durar por todo o período de execução vida da aplicação estando disponível caso seja requisitado novamente (SOARES; BARBOSA, 2009).

4 PROJETO DO SISTEMA

4.1 Proposta

Para possibilitar a incorporação de outras aplicações denominadas *gadgets*, foi proposta a criação de um *software* para a TV Digital, servindo como um ambiente personalizado de utilitários.

O *software* desenvolvido é do tipo residente, portanto, uma aplicação que independe do canal que está sintonizado, assim a seleção e uso dos *gadgets* se tornam dinâmicas, podendo ser utilizadas a qualquer momento. A figura 31 ilustra um esquema da estrutura do sistema televisivo com a aplicação proposta.

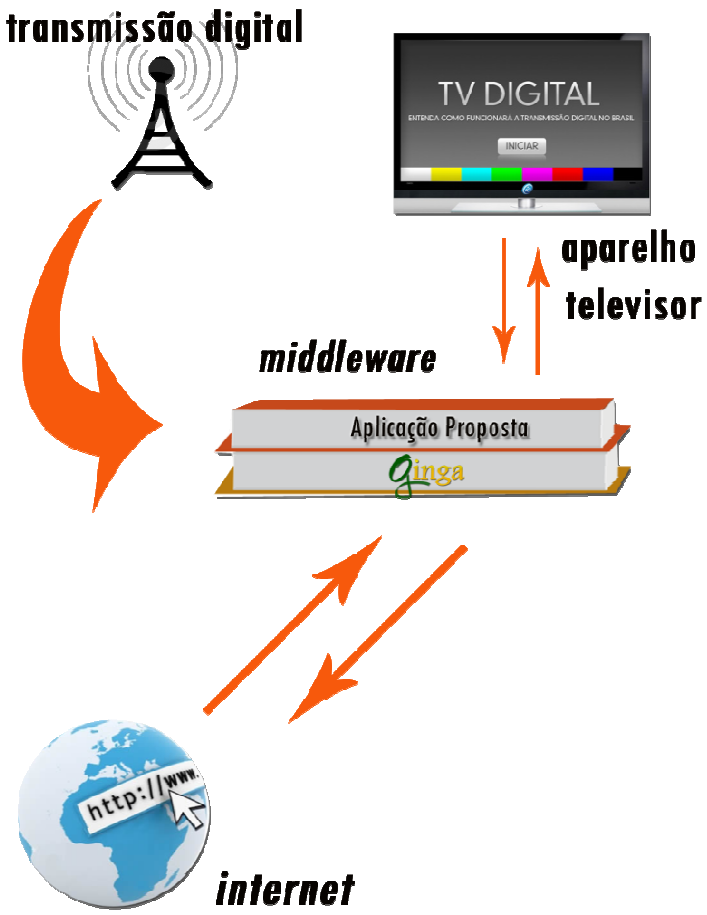

Figura 31: Esquema da aplicação

Fonte: Desenvolvido pelo autor.

Os *gadgets* poderiam ser adquiridos pelo sinal digital enviado pela emissora ou pela internet por meio do canal de retorno. E assim a aplicação faria uso dos *gadgets* que o próprio usuário escolheu.

4.2 Transmissão digital

As emissoras efetuam a difusão do sinal digital, enviando áudio, vídeo e dados. Os dados seriam pertinentes ao conteúdo interativo, podendo ser diversos aplicativos incluindo *gadgets*. Esses *softwares* poderão estar atrelados ao canal sintonizado, sendo que, mediante a uma troca de canal seus dados são perdidos, ou podem ser independentes, permanecendo disponíveis mesmo com a sintonização de novos canais.

Quaisquer aplicações, principalmente *gadgets*, em execução também poderiam receber atualizações de conteúdo por meio dessa mesma transmissão digital.

4.3 Ambiente de *gadgets*

A aplicação proposta executa de forma independente do canal e trata da disponibilização dos *gadgets* adquiridos para configuração e uso.

A disponibilização seria por meio de um menu listando todos os utilitários encontrados para que possam ser escolhidos com o auxílio do controle remoto. Feito a escolha de um *gadget* ele será executado em uma determinada região da tela.

Referente ao uso, o posicionamento do utilitário na tela poderá ser determinado pelo usuário, escolhendo entre regiões já pré-definidas onde seus *gadgets* poderão ser posicionados antes de entrar em execução. A figura 32 traz um modelo de regiões pré-definidas possíveis na tela da TV.

Figura 32: Regiões pré-definidas da tela da TV

Fonte: Desenvolvido pelo autor.

Após a escolha da região o *gadget* entra em execução, a partir desse momento, o foco está no utilitário escolhido, podendo fazer uso de todos os recursos que ele oferece.

4.4 Modelo conceitual

4.4.1 Modelo de aplicação

A aplicação funciona em dois momentos: no primeiro momento o *software* localiza todos os *gadgets* disponíveis em um diretório padrão dentro do Ginga, para cada utilitário encontrado, é feita uma leitura e coleta de dados relacionados para uso posterior. Com todas as informações necessárias será montada dinamicamente a segunda parte da aplicação, que consiste no ambiente para execução. Essa etapa de execução terá todas as referências aos *gadgets* lidos para assim poder executá-los. O ambiente de execução deverá conter um menu listando as opções para seleção e uso. Os *gadgets* serão listados e o usuário fará sua escolha por meio do controle remoto.

Com base nos dois momentos, a aplicação seria estruturada em dois módulos: carregamento e execução, conforme verifica-se na figura 33.

Figura 33: Estrutura do software proposto

Fonte: Desenvolvido pelo autor.

Os dois módulos representam exatamente os dois momentos do software, onde no primeiro o usuário vai carregar a aplicação e no segundo vai executá-la.

4.4.1.1 Módulos

A. Módulo de carregamento

Toda leitura do diretório e coleta de informações dos *gadgets* será feita nesse módulo, que atuará no primeiro momento da aplicação lendo todos os utilitários existentes, e cada um será coletado os dados necessários, montando o módulo de execução resultante. Assim, o *software* está pronto para ser executado pelo próprio usuário telespectador.

Esse carregamento é acionado pelo usuário, onde todos os *gadgets* que ele possuir dentro do diretório padrão estabelecido serão lidos e incorporados à aplicação, assim, o telespectador poderá manter no diretório apenas aplicações de seu interesse para ficarem disponíveis à utilização.

B. Módulo de execução

O módulo de execução será responsável por interpretar toda a estrutura da aplicação como o menu e o acionamento dos *gadgets* incorporados, bem como o resultado do carregamento feito no primeiro módulo. Este módulo terá o controle do menu, e toda usabilidade do sistema como, por exemplo, posicionamento dos *gadgets* nas regiões de tela pré-definidas, funcionando independentemente do canal sintonizado. Mesmo com o acionamento de algum *gadget*, o menu ainda se manterá disponível para utilização podendo selecionar mais utilitários para exibição.

4.4.2 Modelo de *gadgets*

Os *gadgets* deverão ser desenvolvidos seguindo um padrão que garanta a compatibilidade com o ambiente de gerenciamento. Para essa aplicação, foi criado um padrão definindo que meta dados deverão estar presentes nos utilitários, com informações pertinentes de uso dos recursos e configurações. Essas informações indicarão o nome do utilitário, localização do documento para execução e localização de um ícone que represente visualmente a aplicação e uma estrutura do *software,* que também foi proposto para garantir o correto funcionamento e integração.

4.4.2.1 Meta dados

Os meta dados são dados que fornecem informações sobre outros dados (SOARES; BARBOSA, 2009). Os meta dados facilitam o entendimento dos relacionamentos e a utilidade das informações. Esses dados devem estar presentes na aplicação, mas não influenciam no funcionamento do *gadget*. Não há restrições para o limite de meta-informação contida no utilitário, contanto que as estritamente necessárias à aplicação principal se façam presentes.

4.4.2.2 Estrutura dos *gadgets*

A construção do *gadget* deve ser feita tendo, obrigatoriamente, um módulo em NCL que contenha os meta dados necessários. Nesse módulo pode-se ter o desenvolvimento completo da aplicação ou ela pode estar segmentada em diversos outros módulos. Não há restrições quanto às linguagens e recursos que o *gadget* possa utilizar, desde que sejam suportadas pelo *middleware* Ginga.

4.5 Aparelho televisor

A aplicação, nos dois módulos, exibirá conteúdo na tela da TV de maneira que o usuário possa interagir desde o carregamento até a execução do ambiente de *gadgets*.

Essa interação se dará no módulo de carregamento para que o usuário, fazendo uso do controle remoto, acione o início da leitura dos utilitários e ao completar esse processo, acionará a saída da aplicação.

No módulo de execução poderá ser acionado um menu que conterá todos os *gadgets* lidos, selecionar o *gadget* desejado, definir posicionamento, executá-lo, e por fim esconder o menu.

4.6 Internet

O acesso à internet poderá ser feito pelo canal de retorno e com isso os *gadgets* poderão ser adquiridos também por meio da rede e ainda fazer uso dela para receber e enviar informações.

5 TV DIGITAL DECK

5.1 A aplicação

A aplicação implementada com base no modelo conceitual proposta foi chamada de *TV Digital Deck*, desenvolvida em NCL e Lua utilizando a ferramenta de desenvolvimento Eclipse com um *plugin* adicional para NCL e outro para Lua. O *TV Digital Deck* é composto de três arquivos: *proLoader.ncl*, *createMenu.lua* e *resultApp.ncl*. Além desses três arquivos desenvolvidos, a aplicação ainda faz uso de outras duas API´s de funções em Lua: *xml.lua*, que provê funções para fazer o *parser* de documentos XML e *handler.lua,* que atua junto ao *xml.lua* com funções de manipulação. Uma imagem de fundo no formato PNG e dois vídeos no formato MP4 também compõem a aplicação.

A execução do *TV Digital Deck* ocorre em dois momentos. O primeiro momento consiste em carregar e montar o documento resultante com base nas informações presentes em um diretório específico contendo os *gadgets* que serão utilizados. Os *gadgets* encontrados possuem meta dados que serão colhidos e utilizados para a criação desse documento resultante, que depois de pronto conterá um menu com referências para todos os *gadgets*, podendo executá-los conforme critério do usuário. O segundo momento é a execução do *resultApp.ncl* onde dois vídeos são reproduzidos simultaneamente, um deles “escondido”, simulando dois canais diferentes de televisão. Essa simulação se fez necessária, pois não existia a possibilidade de testes da aplicação em um ambiente de rádio difusão do sinal digital, sendo assim foi usado o Ginga-NCL Virtual STB que apenas executa documentos NCL. Ao acionar as teclas de seta para cima e para baixo do controle remoto ocorre a troca de exibição de um canal para outro, sem afetar o restante da aplicação, no caso, o menu dos *gadgets*. Inicialmente o menu não é apresentado, podendo ser acionado através do controle remoto, nele conterá atalhos para os *gadgets* carregados que poderão ser selecionados e executados.

5.1.1 Módulo de carregamento

Esse módulo foi implementado em um documento NCL chamado de *proLoader.ncl.* O *proLoader.ncl* é bastante simples, tendo somente dois objetos de mídia: o *background.png*, que é a imagem de fundo exibida durante a execução e o *createMenu.lua,* um objeto NCLua que fará a montagem do terceiro arquivo: o *resultApp.ncl*. Ao iniciar o documento, a porta de entrada é o objeto *background*, que já entra em exibição com base no descritor e região definidos. A figura 34 mostra a tela inicial.

Figura 34: Tela inicial do *proLoader.ncl*

Fonte: Desenvolvido pelo autor.

O plano de fundo exibido contém as instruções necessárias para que o usuário dê início ao carregamento e montagem do *resultApp.ncl*. Ao pressionar a tecla verde do controle remoto o objeto *createMenu.lua* é acionado. Para encerrar o documento basta apertar a tecla vermelha.

5.1.2 Criação do módulo de execução

O módulo de execução é criado dinamicamente pelo módulo de carregamento por meio do arquivo *createMenu.lua* que é responsável por procurar e armazenar em uma lista as URI dos *gadgets* encontrados no diretório *myGadgets*. Essa lista tem o nome de *sourceList.txt*, que é apenas um arquivo de texto com a relação de todos os endereços dos *gadgets* encontrados. Depois da lista montada, cada item é lido e utilizado para buscar o documento correspondente ao endereço. A busca é feita por meio da URI do documento NCL, que é mapeado para um XML DOM e depois transformado em uma tabela Lua. De posse

dessa tabela é possível extrair os meta-dados presentes no documento e com eles dar início à criação do *resultApp.ncl*.

As informações conseguidas por meio dos meta-dados serve para a criação de *tags* NCL, que são incorporadas no documento *resultApp.ncl*. As *tags* criadas dizem respeito às estruturas de importação do documento original, região e descritor dos ícones que representam o *gadget* no menu, contexto dos *gadgets* para poderem ser executados, objetos de mídia dos ícones de representação dos *gadgets* e elos de início e parada dos elementos, do menu e da aplicação como um todo.

5.1.2.1 Importação

A *tag* de importação de documento é utilizada para que o *resultApp.ncl* tenha uma referência ao *gadget* e poder executá-lo quando necessário. A figura 35 mostra um exemplo de como essa importação é declarada.

```
7⊖      <importedDocumentBase>
8           <importNCL documentURI="rssGadget.ncl" alias="rss"/>
9           <importNCL documentURI="testGadget.ncl" alias="test"/>
10      </importedDocumentBase>
```

Figura 35: Importação do *gadget*

Fonte: Desenvolvido pelo autor.

A *tag* de importação possui dois atributos, o *documentURI* onde será informado o caminho do documento e o *alias* que será um nome usado para referenciar o documento importado dentro do documento que esta importando.

5.1.2.2 Regiões e Descritores

A base de regiões e de descritores são preenchidas de maneiras distintas. A base de regiões é preenchida com sub-regiões para os ícones de representação dos *gadgets*, que ficam todas organizadas espacialmente dentro da área do menu, que já foi previamente definida no arquivo *resultApp.ncl*.

A base de descritores é preenchida com informações que referenciam essas regiões e ainda define, com base em índices, o foco da seleção dos ícones na aplicação. Quando selecionado o primeiro ícone presente no menu, por exemplo, ao pressionar a tecla da seta para a direta do controle remoto, o foco passa para o próximo ícone, essa funcionalidade

corresponde aos atributos *focusIndex*, *moveRight* e *moveLeft* e é definida no descritor de cada ícone como pode ser visto na figura 36.

```
16      <regionBase>
17          <region id="screenReg" width="100%" height="100%" zIndex="2">
18              <region id="frameMenu" top="80%" width="100%" height="18.5%" zIndex="3">
19                  <region id="rssGadgetReg" left="10%" top="10%" width="10%" height="80%" zIndex="4"/>
20                  <region id="testGadgetReg" left="25%" top="10%" width="10%" height="80%" zIndex="4"/>
21              </region>
22          </region>
23      </regionBase>
24
25      <descriptorBase>
26          <descriptor id="channelOneDesc" region="screenReg"/>
27
28          <descriptor id="channelTwoDesc" region="screenReg">
29              <descriptorParam name="visible" value="false"/>
30              <descriptorParam name="soundLevel" value="0"/>
31          </descriptor>
32
33          <descriptor id="menuDesc" region="frameMenu" transIn="trans">
34              <descriptorParam name="transparency" value="0.3"/>
35          </descriptor>
36
37          <descriptor id="rssGadgetDesc" region="rssGadgetReg" focusIndex="11" moveRight="12" moveLeft="10"/>
38          <descriptor id="testGadgetDesc" region="testGadgetReg" focusIndex="12" moveRight="13" moveLeft="11"/>
39      </descriptorBase>
```

Figura 36: Regiões e Descritores

Fonte: Desenvolvido pelo autor.

A quantidade de regiões e descritores declaradas vai variar de acordo como o número de *gadgets* carregados.

5.1.2.3 Contexto

O contexto é criado para que o documento que já foi feito a importação fique disponível por meio de um *alias* (atributo da *tag* de importação), e assim quando um ícone do menu for executado ele chamará o *gadget* correspondente por meio deste <*context*>.

5.1.2.4 Objetos de mídia

Os objetos de mídia montados pelo *createMenu.lua* são apenas os dos ícones dos *gadgets*, lembrando que essa informação é retirada da leitura dos metadados presentes nos documentos. Demais objetos como o menu e os vídeos já vem predefinidos no documento *resultApp.ncl*, e as referências para executar os *gadgets* estão declaradas nos <*context*> e não em objetos de mídia <*media*>.

5.1.2.5 Elos

Por fim os elos são criados, alguns já são predefinidos no *resultApp.ncl*. Os elos pré-definidos são para executar os vídeos, parar, resumir, esconder, tudo dentro da idéia de simular a troca de canal, e também para exibir ou não o menu. Os elos que de fato são criados

dinamicamente pelo objeto lua são os que cuidam da exibição dos ícones dos *gadgets* no menu, oferecem a execução do *gadget* selecionado e o encerramento da aplicação como um todo.

5.1.2.6 Estrutura do arquivo *createMenu.lua*

O código Lua do arquivo *createMenu.lua* foi estruturado da seguinte forma: no início do arquivo são declaradas todas as variáveis de uso global, depois todas as funções e por fim o código de execução.

Para cada tipo de *tag* criada com as informações dos *gadgets* lidos, existe uma função específica que recebe como parâmetro a tabela que contém os dados resultantes do *parser* do *gadget*, e retorna uma *string* com a *tag* definida. A figura 37 mostra um exemplo de função que monta o objeto de mídia do ícone do *gadget* lido.

```
114     --monta a media do icone do gadget no menu
115     --@return string com a media para inserir no ncl principal
116     function montaMediaIcon(tableXml)
117         local id = tableXml.root.ncl.head.meta[1]._attr.name --pega id
118         local uriIcon = tableXml.root.ncl.head.meta[4]._attr.name --pega uri do icone de representacao
119         local result = [[<media id="]] .. id .. [[Icon" src="]] .. uriIcon .. [[" descriptor="]] .. id .. [[Desc"/>
120         ]]
121 
122         return result
123     end
```

Figura 37: Função *montaMediaIcon*

Fonte: Desenvolvido pelo autor.

Além das funções para criação das *tags*, foram definidas também funções para leitura do diretório *myGadgets* e criação do arquivo *sourceList.txt*, criação do arquivo *resultApp.ncl* com inserção das *tags* criadas e uma função exclusiva para o *parser* do documento do *gadget*.

O código de execução localizado no final do arquivo faz as chamadas necessárias para as funções e termina a execução com uma mensagem de sucesso desenhada na tela. Ver figura 38.

Figura 38: Tela de sucesso do *proLoader.ncl*

Fonte: Desenvolvido pelo autor.

5.1.3 Módulo de execução

O arquivo resultante é inicialmente um documento NCL simples, sem regiões, descritores, objetos, apenas a estrutura básica que caracteriza um NCL: cabeçalho e corpo. Após a execução do código Lua *createMenu.lua* o documento *resultApp.ncl* é totalmente sobrescrito, resultando em um documento novo com toda estrutura proposta do *TV Digital Deck*, menu, *gadgets* e vídeos, pronto para ser executado

5.1.4 Estrutura de diretórios

A *TV Digital Deck* possui uma estrutura de diretórios padrão para garantir sua correta operação, a figura 39 mostra como ficou definida essa estrutura.

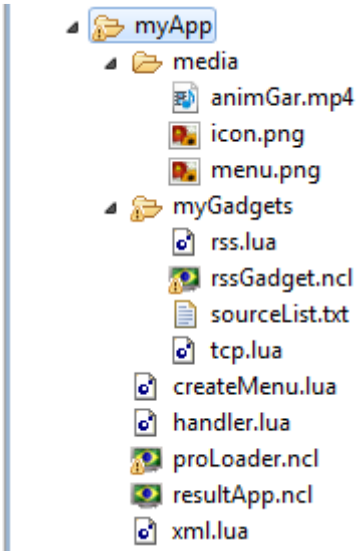

Figura 39: Estrutura de diretórios

Fonte: Desenvolvido pelo autor.

O diretório *myApp* é o principal, onde ficam os arquivos *proLoader.ncl*, *createMenu.lua* e seus dependentes: *xml.lua e handler.lua.* Nesse diretório também ficam: *resultApp.ncl*, o diretório *media* para as imagens e vídeos e o diretório *myGadgets* onde devem ser colocados os *gadgets* que se deseja utilizar para que possam ser lidos ao executar o *proLoader.ncl*. O arquivo *sourceList.txt* quando criado também fica localizado no diretório *myGadgets*.

5.1.5 Execução

Para executar a *TV Digital Deck*, toda sua estrutura de diretórios e arquivos foi transferida para a máquina virtual Ginga-NCL Virtual STB, precisamente dentro do diretório */misc/ncl30/*. No diretório *myGadgets* foram colocados os arquivos *rssGadget.ncl*, *rss.lua*, *tcp.lua* e as imagens *dir.png*, *esq.png* e *fechar.png* que compõe uma adaptação de um *gadget* para leitura de RSS na TV Digital, feito por Manoel Campos (2010). Feita a transferência dos arquivos, partiu-se para a execução do documento *proLoader.ncl* que como resultado gerou o arquivo *sourceList.txt* com a URI do aplicativo *rssGadget* e sobrescreveu o documento *resultApp.ncl*. Com o *resultApp.ncl* pronto pode-se executá-lo tendo como efeito a apresentação dos vídeos e o menu com o novo *gadget* pronto para uso. As figuras 40 e 41 mostram o menu e o *rssGadget* em execução respectivamente.

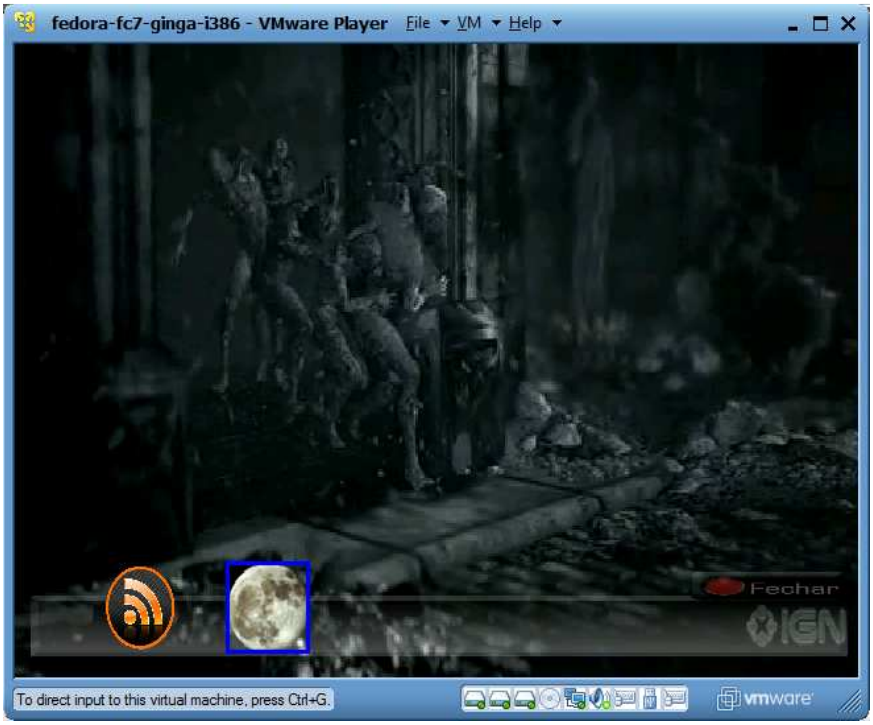

Figura 40: *TV Digital Deck*

Fonte: Desenvolvido pelo autor.

Figura 41: *rssGadget*

Fonte: Desenvolvido pelo autor.

5.2 Modelo de *gadgets*

O modelo proposto define que o *gadget* deve ser desenvolvido em NCL e possuir uma imagem que o represente, um ícone para ser usado dentro do menu da *TV Digital Deck*. O documento NCL deve conter meta dados com determinadas informações que servirão de base para seu gerenciamento.

5.2.1 Meta dados

Os meta dados necessários no modelo de *gadget* proposto são:

- *id*: nome do documento NCL do *gadget* (sem extensão);
- *alias*: nome atributo *component* da porta de entrada do documento;
- *uri*: URI do documento;
- *uriIcon*: URI da imagem que vai representar o *gadget* (ícone).

Essas informações devem ser preenchidas nessa ordem, pois o código Lua da *TV Digital Deck* que colhe esses dados usa por base seus índices que são numerados de forma crescente, começando pelo número um. Essa numeração é atribuída no momento do *parser* do XML. Outros meta dados podem ser inseridos, porém, sempre depois dos quatro propostos no modelo. A figura 42 mostra um exemplo de como ficam declarados os meta dados.

```
32          <meta content="id" name="rssGadget"/>
33          <meta content="alias" name="rss"/>
34          <meta content="uri" name="/misc/ncl30/myApp/myGadgets/rssGadget.ncl"/>
35          <meta content="uriIcon" name="/misc/ncl30/myApp/myGadgets/rss.png"/>
```

Figura 42: Declaração dos meta-dados

Fonte: Desenvolvido pelo autor.

Os meta dados devem ser declarados no cabeçalho do documento e devem ficar abaixo da declaração dos conectores.

Sem nenhum caráter de obrigatoriedade, são recomendáveis que os *gadgtes* desenvolvidos tenham determinados comportamentos que melhorem a usabilidade da aplicação. Opções de posicionamento do *gadget* na tela e saída da aplicação são bem vindas, tendo em vista as limitações de operação causadas pelo uso do controle remoto.

5.2.2 *Gadget* leitor de Rss

O *gadget* utilizado no *TV Digital Deck* é uma adaptação do Lua *Rss Reader* para TV Digital feito por Manoel Campos (2010), esse *gadget* foi chamado de *rssGadget* e tem por objetivo apresentar as notícias de determinado *feed*, um alimentador de RSS, que foi assinado. Assinando um *feed* tem-se acesso a constantes atualizações de novos RSS sobre informações das quais o *feed* assinado se propõe a trazer. Se foi assinado um *feed* de um *site* de tecnologia certamente as notícias, os RSS, serão sobre tecnologia.

No *rssGadge*t as notícias são buscadas na internet por meio do canal de retorno, usando o protocolo *tcp* implementado em uma API em Lua. Com o *gadget* em execução é possível ler as notícias, saber quantas existem, avançar para a próxima, voltar à anterior e sair da aplicação.

5.2.2.1 RSS

O RSS (*Really Simple Syndication*) é um formato de conteúdo simples e padronizado feito em XML com o objetivo de compartilhar informações novas (ALECRIM, 2008).

O arquivo RSS contém informações como: título, links, descrição da alteração, data, autor, etc, que são geralmente das últimas atualizações do *site* ao qual ele está agregado. Minuto a minuto o arquivo RSS é atualizado mostrando as alterações recentes.

5.2.2.2 Estrutura do *rssGadget*

A aplicação está dividida em 2 módulos: exibição e controle. Esse último necessita do arquivo *tcp.lua*, uma API com funções para trabalhar com conexões *tcp*. Além dessa estrutura são usadas também as imagens: *rssIcon.png*, *dir.png*, *esq.png* e *fechar.png* que são utilizadas na parte gráfica do *gadget*.

A. Módulo de exibição

Nesse módulo, escrito em NCL, são declarados: os meta dados necessários para integração com *TV Digital Deck*, o NCLua responsável por toda funcionalidade da aplicação e elos responsáveis pelo posicionamento do *gadget* na tela e por encerrar a aplicação ao pressionar o botão vermelho do controle remoto.

B. Módulo de controle

O módulo de controle foi implementado com um NCLua que vai buscar o RSS do *feed* assinado através de uma conexão *tcp*. O RSS sendo um XML é transformado em uma tabela

Lua por meio de um *parser*, e assim são coletadas as informações, usualmente notícias, e exibidas na tela. A Figura 41 mostra o *rssGadget* em funcionamento.

C. API de conexões *tcp*

A API *tcp.lua* foi desenvolvida pela PUC-Rio para prover funções utilizadas para conexões *tcp*. Essa API utiliza co-rotinas de Lua para simular *multi-thread*. A figura 43 mostra um exemplo de função presente em *tcp.lua*.

```
116 ---Conecta em um servidor por meio do protocolo TCP.
117 --A função só retorna quando a conexão for estabelecida.
118 --@param host Nome do host para conectar
119 --@param port Porta a ser usada para a conexão
120 function connect (host, port)
121     local t = {
122         host    = host,
123         port    = port,
124         waiting = 'connect'
125     }
126     CONNECTIONS[coroutine.running()] = t
127 
128     event.post {
129         class = 'tcp',
130         type  = 'connect',
131         host  = host,
132         port  = port,
133     }
134 
135     --Suspende a execução da co-rotina.
136     --A função atual (connect) só retorna quando
137     --a co-rotina for resumida, o que ocorre
138     --quando o evento connect é capturado
139     --pela função handler.
140     return coroutine.yield()
141 end
```

Figura 43: Função *connect*

Fonte: Desenvolvido pelo autor.

A função *connect* recebe dois parâmetros: *host* e *port*, que são o endereço a qual se deseja conectar e a porta, respectivamente. Os dois parâmetros recebidos são colocados em uma tabela junto com mais uma informação denominada *waiting* com o valor “connect”. Essa tabela é colocada dentro de outra como uma *thread*, tornando-se uma co-rotina que vai executar até atingir seu objetivo: obter uma conexão com o *host* especificado. Na função *connect* também é registrado um evento com as informações de conexão, que será tratado pela função *handler* da API *tcp.lua* onde realizará os procedimentos para adquirir a conexão. Por

fim, a função *connect* só retorna quando a co-rotina for resumida, o que ocorre quando o evento é capturado pelo *handler*.

6 CONCLUSÕES

Ainda que não totalmente definido o potencial de interatividade da TV Digital brasileira, já é possível desenvolver aplicações de todos os tipos fazendo uso de especificações de linguagens e outros recursos suportados pelo *middleware* Ginga. Dentro dessas possibilidades, pôde-se construir um modelo conceitual de desenvolvimento e colocá-lo em prática por meio de sua implementação. Mesmo com algumas limitações, obteve-se um resultado satisfatório, que pode incentivar novos trabalhos de especificação e também desenvolvimento.

Para que fosse possível criar uma proposta de *software* por meio de padrões e um modelo conceitual, foi realizado um estudo sobre a TV digital brasileira.

Com base no modelo desenvolvido, pôde-se implementar uma aplicação que atendeu a requisitos definidos previamente. Essa aplicação fez uso das linguagens de programação NCL e Lua que fazem parte da especificação do *middleware* Ginga.

As linguagens NCL e Lua foram analisadas de maneira que possibilitaram a extração de informações a respeito de formas para atender os requisitos do modelo, deixando evidente o potencial das duas linguagens.

Existiram algumas dificuldades ao longo da etapa de desenvolvimento como a impossibilidade de testar as aplicações interativas em ambientes reais. Essa dificuldade foi contornada com o uso de um ambiente virtual simulado, o que acarretou em outros potenciais problemas, como a necessidade de transferência dos arquivos da aplicação para esse ambiente usando uma ferramenta de comunicação em protocolo SSH. Para cada alteração no *software*, uma nova transferência teve que ser feita para possibilitar um novo teste. A decisão de desenvolver o aplicativo proposto em dois momentos de execução se deu por conseqüência de uma limitação da linguagem NCL no que diz respeito a um documento poder importar outro em tempo de execução.

A aplicação resultou num *software*, que incorpora outros *softwares* denominados *gadgets* e com isso organiza, gerencia e personaliza esses utilitários, contribuindo para a interatividade proposta da TV Digital brasileira.

6.1 Trabalhos futuros

Com base na proposta de um modelo conceitual de aplicação para TV Digital, que oferece suporte a outras aplicações denominadas *gadgets*, novos utilitários podem ser desenvolvidos seguindo esse ou uma adaptação do padrão apresentado. Melhorias no modelo

podem ser feitas, enriquecendo a especificação e dando continuidade a um desenvolvimento organizado dentro do ambiente da TV Digital.

A implementação criada a partir da estrutura conceitual apresentada servirá de ponte para utilização dos *gadgets*, que serão desenvolvidos seguindo a especificação e modelos de utilitários.

Para trabalhos futuros poderá ser modelado e implementado a possibilidade de obtenção dos *gadgets* por meio do sinal digital televisivo, e não apenas pelo canal de retorno. Essa implementação ampliaria a possibilidade de difusão desses utilitários, sendo mais uma fonte de obtenção de informação. Outra sugestão é criar o tratamento específico que deixará o *software* ser executado como uma aplicação residente do Ginga e assim se tornando independente do canal sintonizado. A possibilidade de definição do posicionamento dos *gadgets* em regiões da tela pré-definidas não fez parte da implementação do TV Digital Deck, ficando também como sugestão acrescentá-la ao software.

7 REFERÊNCIAS

ABNT NBR 15601 (2007). Associação Brasileira de Normas Técnicas, “Televisão digital terrestre – sistema de transmissão”. Sistema Brasileiro de TV Digital Terrestre, *NBR 15601*.

ABNT NBR 15602-1 (2007). Associação Brasileira de Normas Técnicas, “Televisão digital terrestre – codificação de vídeo, áudio e multiplexação, Parte 1: Codificação de vídeo”. Sistema Brasileiro de TV Digital Terrestre, *NBR 15602-1*.

ABNT NBR 15602-2 (2007). Associação Brasileira de Normas Técnicas, “Televisão digital terrestre – codificação de vídeo, áudio e multiplexação, Parte 2: Codificação de áudio”. Sistema Brasileiro de TV Digital Terrestre, *NBR 15602-2*.

ABNT NBR 15606-2 (2007). Associação Brasileira de Normas Técnicas, “Televisão digital terrestre – codificação de dados e especificações de transmissão para rádiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações”. Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ABNT NBR 15606-2 (2007). Associação Brasileira de Normas Técnicas, “Televisão digital terrestre – objetos procedurais Lua em apresentações NCL”. Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ALECRIM, Emerson. **O que é RSS?** Disponível em: <http://www.infowester.com/rss.php> Acesso em novembro de 2010.

GINGA DIGITAL TV MIDDLEWARE SPECIFICATION. Disponível em: <http://www.ginga.org.br/> Acesso em novembro de 2010.

MELLO, Laurentino. **Gadget. Saiba o que é...** Disponível em: <http://www.tinotec.com.br/blog/gadget-saiba-o-que-e/> Acesso em novembro de 2010.

SANT´ANNA, F.; CERQUEIRA, R.; SOARES, L.F.G. **NCLua – Objetos imperativos Lua na linguagem declarativa NCL**. Disponível em: <ftp://ftp.telemidia.puc-rio.br/~lfgs/docs/conferencepapers/2008_10_santanna.pdf> Acesso em 23/10/2010.

SOARES, Luiz Fernando Gomes. **TV interativa se faz com Ginga.** Disponível em: <http://www.gingancl.org.br/resources/Encarte-mod.pdf> Acesso em 23/10/2010.

SOARES, Luiz Fernando Gomes; BARBOSA, Simone Diniz Junqueira. **Programando em NCL 3.0: Desenvolvimento de aplicações para o middleware Ginga.** RJ: Elsevier, 2009.

SOARES, L.F.G.; RODRIGUES, F.R.; BARBOSA, J.D.S. **Manual de construção de programas audiovisuais interativos utilizando a NCL 2.3** – Perfil básico. Disponível em: <http://www.ncl.org.br/documentos/manualNCL2_3.pdf> Acesso em: 23/10/2010.

TELECO: **Inteligência em telecomunicações. TV Digital: O que é?** Disponível em: <http://www.teleco.com.br/tvdigital.asp> Acesso em 08/11/2010.

THE PROGRAMMING LANGUAGE LUA. Disponível em: <http://www.lua.org/portugues.html> Acesso em novembro de 2010.

8 APÊNDICE

8.1 CÓDIGO FONTE

proLoader.ncl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Modulo de Carregamento -->
<!-- Generated by NCL Eclipse -->
<!-- @author Andre Campos -->
<ncl id="proLoader" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
      <head>
            <regionBase>
                  <region id="screenReg" width="100%" height="100%"
zIndex="1">
                        <region id="backgroundReg" width="100%"
height="100%" zIndex="2">
                              <region id="luaReg" left="25%" top="70%"
width="40%" height="20%" zIndex="3"/>
                        </region>
                  </region>
            </regionBase>

            <descriptorBase>
                  <descriptor id="backgroundDesc" region="backgroundReg"/>
                  <descriptor id="luaDesc" region="luaReg"
focusIndex="createMenuLuaIdx"/>
            </descriptorBase>

            <connectorBase>
                  <causalConnector id="onKeySelectionStart">
                        <connectorParam name="keyCode"/>
                        <connectorParam name="delay"/>
                        <simpleCondition role="onSelection"
key="$keyCode"/>
                        <compoundAction operator="seq">
                              <simpleAction role="start" delay="$delay"
max="unbounded" qualifier="par"/>
                              <simpleAction role="stop" delay="$delay"
max="unbounded" qualifier="par"/>
                        </compoundAction>
                  </causalConnector>

                  <causalConnector id="onKeySelectionStop">
                        <connectorParam name="keyCode"/>
                        <simpleCondition role="onSelection"
key="$keyCode"/>
                        <simpleAction role="stop" max="unbounded"
qualifier="par"/>
                  </causalConnector>
            </connectorBase>
      </head>

      <body>
            <port id="entry" component="background"/>
```

```
            <media type="application/x-ginga-settings"
id="programSettings">
              <property name="service.currentKeyMaster"
value="createMenuLuaIdx"/>
            </media>

            <media id="background" src="media/background.png"
descriptor="backgroundDesc"/>

            <media id="sucess" src="media/sucess.png"
descriptor="luaDesc"/>
            <media id="loading" src="media/loading.png"
descriptor="luaDesc"/>

            <media id="luaProLoader" src="createMenu.lua"
descriptor="luaDesc"/>

            <link id="launch" xconnector="onKeySelectionStart">
                  <bind role="onSelection" component="background">
                        <bindParam name="keyCode" value="GREEN"/>
                  </bind>
                  <bind role="start" component="luaProLoader">
                        <bindParam name="delay" value="0"/>
                  </bind>
                  <bind role="start" component="loading">
                        <bindParam name="delay" value="0"/>
                  </bind>
                  <bind role="stop" component="loading">
                        <bindParam name="delay" value="5"/>
                  </bind>
                  <bind role="start" component="sucess">
                        <bindParam name="delay" value="5"/>
                  </bind>
            </link>

            <link id="end" xconnector="onKeySelectionStop">
                  <bind role="onSelection" component="background">
                        <bindParam name="keyCode" value="RED"/>
                  </bind>
                  <bind role="stop" component="background"/>
                  <bind role="stop" component="luaProLoader"/>
                  <bind role="stop" component="sucess"/>
                  <bind role="stop" component="loading"/>
            </link>
      </body>
</ncl>
```

resultApp.ncl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Modulo de Execucao -->
<!-- Generated by NCL Eclipse -->
<!-- @author Andre Campos -->
<ncl id="resultApp" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
      <head>
            <!-- inicialmente vazia, pois sera gerada dinamicamente atraves
do script em Lua -->
      </head>

      <body>

      </body>
</ncl>
```

createMenu.lua

```
---Modulo de Carregamento<br/>
---@author André Campos<br/>

dofile("/misc/ncl30/myApp/xml.lua")
dofile("/misc/ncl30/myApp/handler.lua")

      ---VARIAVEIS GLOBAIS

      --contador de gadgets lidos
      counterApp = 0

      --string onde ficarao todas as tags de import do documento ncl
      resultImports = ''

      --string onde ficarao todas as tags de regioes do documento ncl
      resultRegions = ''

      --string onde ficarao todas as tags de descritores do documento ncl
      resultDescriptors = ''

      --string onde ficarao todas as tags de contexto do documento ncl
      resultContexts = ''

      --string onde ficarao todas as tags de medias do documento ncl
      resultMedias = ''

      --string onde ficarao todas as tags de bind role para start do
documento ncl
      resultBindStart = ''

      --string onde ficarao todas as tags de bind role para stop do
documento ncl
      resultBindStop = ''

      --string onde ficarao todas as tags de bind role para stop dos
gadgets do documento ncl
      resultBindStopGadgets = ''
```

```
	--string onde ficarao todas as tags de link para execucao do gadget
do documento ncl
	resultLink = ''

	--string onde ficarao todas as tags de bind role para stop do
documento ncl
	resultBindStopGeral = ''

	---

	---FUNCOES

	--obtem a lista de aplicacoes existentes no diretorio gadgets
	--@return uma tabela contendo o endereco das aplicacoes localizadas
no diretorio gadgets.
	function findApps()
		--a funcao utiliza o modulo "os" de lua e executa
		--o comando find (uma aplicação Linux escrita em C)
		--para buscar as aplicacoes NCL no diretorio gadgets e gerar
	    --a saida em um arquivo de texto, contendo a lista
	    --de arquivo encontrados.
		os.execute("find /misc/ncl30/myApp/myGadgets/ -type f -name
'*.ncl' > /misc/ncl30/myApp/myGadgets/sourceList.txt")

	    --tabela que armazena o endereço de cada app localizada pela
função
	    local source = {}

	    --carrega o arquivo de texto criado e o percorre linha a linha,
	    --adicionando em uma tabela, os endereços das apps encontradas.
		for line in
io.lines("/misc/ncl30/myApp/myGadgets/sourceList.txt") do
		    print(#source .. " - " .. line)
		    table.insert(source, line) --insere na tabela o indice e o
endereco de cada app
		    counterApp = counterApp + 1
		end

	    return source
	end

	--faz o parse dos dados contidos no xml econtrado atraves da uri
passada como parametro
	--@return uma table lua com o xml embutido
	function parseXmlToTable(uri)
		---Instancia o objeto que e responsavel por
		---armazenar o XML em forma de uma table lua
		local tableXml = simpleTreeHandler()

		---nome do arquivo XML a ser lido
		local FILE_NAME = uri

		--abre o arquivo XML
		local file = io.open(FILE_NAME, "r") --somente leitura

		if (file) then
		  --passa o conteudo do arquivo para uma string
		  local xmltext = file:read("*all")
		      file:close()
```

```
            --instancia o objeto que faz a conversao de XML para uma
table lua
            xmlparser = xmlParser(tableXml)
            xmlparser:parse(xmltext)
            --agora a variavel tableXml eh um table lua com o xml
embutido
        end

        return tableXml
    end

    --monta a regiao de posicionamento do icone do gadget no menu
    --@return string com a regiao para inserir no ncl principal
    function montaRegGadget(tableXml, left) --pega id
        local id = tableXml.root.ncl.head.meta[1]._attr.name
        local result = [[<region id="]] .. id .. [[Reg" left="]] ..
left .. [[%" top="10%" width="10%" height="80%" zIndex="4"/>
                    ]]

        return result
    end

    --monta o descritor de posicionamento do icone do gadget no menu
    --@return string com o descritor para inserir no ncl principal
    function montaDescGadget(tableXml, index)
        local id = tableXml.root.ncl.head.meta[1]._attr.name --pega id
        local result = [[<descriptor id="]] .. id .. [[Desc" region="]]
.. id .. [[Reg" focusIndex="]] .. index .. [[" moveRight="]] .. index + 1
.. [[" moveLeft="]] .. index - 1 .. [["/>
                ]]

        return result
    end

    --monta a media do icone do gadget no menu
    --@return string com a media para inserir no ncl principal
    function montaMediaIcon(tableXml)
        local id = tableXml.root.ncl.head.meta[1]._attr.name --pega id
        local uriIcon = tableXml.root.ncl.head.meta[4]._attr.name --
pega uri do icone de representacao
        local result = [[<media id="]] .. id .. [[Icon" src="]] ..
uriIcon .. [[" descriptor="]] .. id .. [[Desc"/>
        ]]

        return result
    end

    --monta o contexto do gadget que sera importado
    --@return string com o contexto para inserir no ncl principal
    function montaContextoGadget(tableXml)
        local id = tableXml.root.ncl.head.meta[1]._attr.name --pega id
        local alias = tableXml.root.ncl.head.meta[2]._attr.name --pega
alias
        local result = [[<context id="]] .. id .. [[" refer="]] ..
alias .. [[#]] .. id .. [["/>
        ]]

        return result
    end

    --monta o import do gadget
```

```
	--@return string com o import para inserir no ncl principal
	function montaImportGadget(tableXml)
		local alias = tableXml.root.ncl.head.meta[2]._attr.name --pega
alias
		local uri = tableXml.root.ncl.head.meta[3]._attr.name --pega
uri do gadget
		local result = [[<importNCL documentURI="]] .. uri .. [["
alias="]] .. alias .. [["/>
			]]

		return result
	end

	--monta os binds de stop para o link de stop do menu e o link de end
da app
	--@return string com os binds para inserir no ncl principal
	function montaBindsStopIconMenu(tableXml)
		local id = tableXml.root.ncl.head.meta[1]._attr.name --pega id
		local result = [[<bind role="stop" component="]] .. id ..
[[Icon"/>
			]]

		return result
	end

	--monta os binds de stop para o link de end da app
	--@return string com os binds para inserir no ncl principal
	function montaBindsStopGadgets(tableXml)
		local id = tableXml.root.ncl.head.meta[1]._attr.name --pega id
		local result = [[<bind role="stop" component="]] .. id .. [["/>
			]]

		return result
	end

	--monta os binds de start para o link de start do menu
	--@return string com os binds para inserir no ncl principal
	function montaBindsStartIconMenu(tableXml)
		local id = tableXml.root.ncl.head.meta[1]._attr.name --pega id
		local result = [[<bind role="start" component="]] .. id ..
[[Icon">
				<bindParam name="delay" value="1s"/>
			</bind>
			]]

		return result
	end

	--monta link para executar um gadget do menu
	--@return string com o link de start do gadget para inserir no ncl
principal
	function montaLinkStartGadget(tableXml)
		local id = tableXml.root.ncl.head.meta[1]._attr.name --pega id
		local result = [[<link id="]] .. id .. [[Link"
xconnector="onKeySelectionStopStart">
			<bind role="onSelection" component="]] .. id .. [[Icon">
				<bindParam name="keyCode" value="VK_ENTER"/>
			</bind>
			<bind role="stop" component="menu"/>
			]]
			.. resultBindStopGeral ..
```

```
                  [[

                  <bind role="start" component="]] .. id .. [["/>
            </link>

            ]]

            return result
      end

      function montaTags()
            local left = 10

            for i = 1, counterApp do
                  --tabela com os metadados do gadget
                  tableApp = parseXmlToTable(appsUri[i])

                  --monta as tags que serao inseridas no documento ncl
chamando as respectivas funcoes de montagem
                  resultImports = resultImports ..
montaImportGadget(tableApp)
                  resultRegions = resultRegions .. montaRegGadget(tableApp,
left)
                  resultDescriptors = resultDescriptors ..
montaDescGadget(tableApp, i + 10)
                  resultContexts = resultContexts ..
montaContextoGadget(tableApp)
                  resultMedias = resultMedias .. montaMediaIcon(tableApp)
                  resultBindStop = resultBindStop ..
montaBindsStopIconMenu(tableApp)
                  resultBindStopGadgets = resultBindStopGadgets ..
montaBindsStopGadgets(tableApp)
                  resultBindStart = resultBindStart ..
montaBindsStartIconMenu(tableApp)

                  --espacamento entre os icones no menu aumenta de 15 em 15
                  left = left + 15
            end

            resultBindStopGeral = resultBindStopGeral .. resultBindStop

            for i = 1, counterApp do
                  --tabela com os metadados do gadget
                  tableApp = parseXmlToTable(appsUri[i])

                  --monta as tags de link que serao inseridas no documento
ncl chamando as respectivas funcoes de montagem
                  --estao num for separado para poder concatenar os stops
de todos os gadgets lidos no final
                  resultLink = resultLink .. montaLinkStartGadget(tableApp)
            end
      end

      --monta o ncl resultante da leitura dos gadgets envolvidos
      function montaArquivoResultado()
            result = [[<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Modulo de Execucao -->
<!-- Generated    by NCL Eclipse -->
<!-- Andre Campos -->
<ncl id="resultApp" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
      <head>
```

```
            <importedDocumentBase>
                  ]]
                  .. resultImports ..
                  [[

            </importedDocumentBase>

            <transitionBase>
                  <transition id="trans" type="barWipe"/>
            </transitionBase>

            <regionBase>
                  <region id="screenReg" width="100%" height="100%"
zIndex="2">
                        <region id="frameMenu" top="80%" width="100%"
height="18.5%" zIndex="3">
                              ]]
                              .. resultRegions ..
                              [[

                        </region>
                  </region>
            </regionBase>

            <descriptorBase>
                  <descriptor id="channelOneDesc" region="screenReg"/>

                  <descriptor id="channelTwoDesc" region="screenReg">
                        <descriptorParam name="visible" value="false"/>
                        <descriptorParam name="soundLevel" value="0"/>
                  </descriptor>

                  <descriptor id="menuDesc" region="frameMenu"
transIn="trans"/>

                  ]]
                  .. resultDescriptors ..
                  [[

            </descriptorBase>

            <connectorBase>
                  <causalConnector id="onBeginStart">
                        <simpleCondition role="onBegin"/>
                        <simpleAction role="start" max="unbounded"
qualifier="par"/>
                  </causalConnector>

                  <causalConnector id="onBeginStart">
                        <simpleCondition role="onBegin"/>
                        <simpleAction role="start" max="unbounded"
qualifier="par"/>
                  </causalConnector>

                  <causalConnector id="onKeySelectionSet">
                        <connectorParam name="keyCode"/>
                        <connectorParam name="var"/>
                        <simpleCondition role="onSelection"
key="$keyCode"/>
                        <compoundAction operator="seq">
```

```
                              <simpleAction role="set" value="$var"
max="unbounded" qualifier="par"/>
                        </compoundAction>
                  </causalConnector>

                  <causalConnector id="onKeySelectionStart">
                        <connectorParam name="keyCode"/>
                        <connectorParam name="delay"/>
                        <simpleCondition role="onSelection"
key="$keyCode"/>
                        <simpleAction role="start" delay="$delay"
max="unbounded" qualifier="par"/>
                  </causalConnector>

                  <causalConnector id="onKeySelectionStop">
                        <connectorParam name="keyCode"/>
                        <simpleCondition role="onSelection"
key="$keyCode"/>
                        <simpleAction role="stop" max="unbounded"
qualifier="par"/>
                  </causalConnector>

                  <causalConnector id="onKeySelectionStopStart">
                        <connectorParam name="keyCode"/>
                        <simpleCondition role="onSelection"
key="$keyCode"/>
                        <compoundAction operator="seq">
                              <simpleAction role="stop" max="unbounded"
qualifier="par"/>
                              <simpleAction role="start" max="unbounded"
qualifier="par"/>
                        </compoundAction>
                  </causalConnector>

                  <causalConnector id="onEndStop">
                        <simpleCondition role="onEnd"/>
                        <simpleAction role="stop" max="unbounded"
qualifier="par"/>
                  </causalConnector>
            </connectorBase>
      </head>

      <body>
            <port id="entry1" component="channelOne"/>

            ]]
            .. resultContexts ..
            [[


            <media id="channelOne"
src="/misc/ncl30/myApp/media/animGar.mp4" descriptor="channelOneDesc">
                  <property name="visible"/>
                  <property name="soundLevel"/>
            </media>

            <media id="channelTwo" src="/misc/ncl30/myApp/media/drible.mp4"
descriptor="channelTwoDesc">
                  <property name="soundLevel"/>
                  <property name="visible"/>
            </media>
```

```
            <media id="menu" src="/misc/ncl30/myApp/media/menu.png"
descriptor="menuDesc"/>

            ]]
            .. resultMedias ..
            [[

            <link id="lApp" xconnector="onBeginStart">
                  <bind role="onBegin" component="channelOne"/>
                  <bind role="start" component="channelTwo"/>
            </link>

            <link id="lChangeChannelToOne" xconnector="onKeySelectionSet">
                  <bind role="onSelection" component="channelTwo">
                        <bindParam name="keyCode" value="CURSOR_DOWN"/>
                  </bind>
                  <bind role="set" component="channelOne"
interface="visible">
                        <bindParam name="var" value="true"/>
                  </bind>
                  <bind role="set" component="channelOne"
interface="soundLevel">
                        <bindParam name="var" value="1"/>
                  </bind>
                  <bind role="set" component="channelTwo"
interface="visible">
                        <bindParam name="var" value="false"/>
                  </bind>
                  <bind role="set" component="channelTwo"
interface="soundLevel">
                        <bindParam name="var" value="0"/>
                  </bind>
            </link>

            <link id="lChangeChannelToTwo" xconnector="onKeySelectionSet">
                  <bind role="onSelection" component="channelOne">
                        <bindParam name="keyCode" value="CURSOR_UP"/>
                  </bind>
                  <bind role="set" component="channelTwo"
interface="visible">
                        <bindParam name="var" value="true"/>
                  </bind>
                  <bind role="set" component="channelTwo"
interface="soundLevel">
                        <bindParam name="var" value="1"/>
                  </bind>
                  <bind role="set" component="channelOne"
interface="visible">
                        <bindParam name="var" value="false"/>
                  </bind>
                  <bind role="set" component="channelOne"
interface="soundLevel">
                        <bindParam name="var" value="0"/>
                  </bind>
            </link>

            <link id="lMenuOn1" xconnector="onKeySelectionStart">
                  <bind role="onSelection" component="channelOne">
                        <bindParam name="keyCode" value="GREEN"/>
                  </bind>
```

```
      <bind role="start" component="menu">
            <bindParam name="delay" value="0s"/>
      </bind>
      ]]
      .. resultBindStart ..
      [[

</link>

<link id="lMenuOn2" xconnector="onKeySelectionStart">
      <bind role="onSelection" component="channelTwo">
            <bindParam name="keyCode" value="GREEN"/>
      </bind>
      <bind role="start" component="menu">
            <bindParam name="delay" value="0s"/>
      </bind>
      ]]
      .. resultBindStart ..
      [[

</link>

<link id="lMenuOff" xconnector="onKeySelectionStop">
      <bind role="onSelection" component="menu">
            <bindParam name="keyCode" value="RED"/>
      </bind>
      <bind role="stop" component="menu"/>
      ]]
      .. resultBindStop ..
      [[

</link>

<link id="lEnd" xconnector="onEndStop">
      <bind role="onEnd" component="channelOne"/>
      <bind role="stop" component="channelTwo"/>
      <bind role="stop" component="menu"/>
      ]]
      .. resultBindStop ..
      [[

      ]]
      .. resultBindStopGadgets ..
      [[

</link>

<link id="lKeyEndOne" xconnector="onKeySelectionStop">
      <bind role="onSelection" component="channelOne">
            <bindParam name="keyCode" value="EXIT"/>
      </bind>
      <bind role="stop" component="channelOne"/>
      <bind role="stop" component="channelTwo"/>
      <bind role="stop" component="menu"/>
      ]]
      .. resultBindStop ..
      [[

      ]]
      .. resultBindStopGadgets ..
      [[
```

```
            </link>

            <link id="lKeyEndTwo" xconnector="onKeySelectionStop">
                  <bind role="onSelection" component="channelTwo">
                        <bindParam name="keyCode" value="EXIT"/>
                  </bind>
                  <bind role="stop" component="channelOne"/>
                  <bind role="stop" component="channelTwo"/>
                  <bind role="stop" component="menu"/>
                  ]]
                  .. resultBindStop ..
                  [[

                  ]]
                  .. resultBindStopGadgets ..
                  [[

            </link>

            ]]
            .. resultLink ..
            [[

      </body>
</ncl>]]

            RESULT_FILE = "/misc/ncl30/myApp/resultApp.ncl"

            d = io.open(RESULT_FILE, "w") --modo escrita
            d:write(result)
            d:close()

      end

      function desenhaSucess()
            local regLarg, regAlt = canvas:attrSize()
            print('desenhando imagem fim')
            print('largura', regLarg)
            print('altura', regLarg)
            local imgSucess =
canvas:new("/misc/ncl30/myApp/media/sucess.png")
            canvas:compose(regLarg, regAlt, imgSucess)
            canvas:flush()
      end

      ---

      ---EXECUCAO
      function execute()
            --tabela que armazena a lista de endereços dos gadgets a serem
exbidos na aplicacao
            appsUri = findApps()

            --monta as tags que serao inseridas no ncl resultante
            montaTags()

            --mescla as tags geradas com o modelo e gera o arquivo ncl
resultante
            montaArquivoResultado()
```

```
	--desenha a msg de fim do carregamento
	regLarg, regAlt = canvas:attrSize()

	event.timer(5000, desenhaSucess)

end

function handler(evt)
	--desenha a msg de inicio do carregamento
	local regLarg, regAlt = canvas:attrSize()
	print('desenhando imagem inicio')
	print('largura', regLarg)
	print('altura', regLarg)
	local img = canvas:new("/misc/ncl30/myApp/media/loading.png")
	canvas:compose(regLarg, regAlt, img)
	canvas:flush()

	execute()
end

event.register(handler)

--- FIM
```

8.1 ARTIGO

AMBIENTE DE *GADGETS* PARA TV DIGITAL

André Lima Rocha Campos

Departamento de Informática e Estatística – INE
Universidade Federal de Santa Catarina – UFSC
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil

andrecamppos@inf.ufsc.br

Resumo. Este artigo apresenta o resultado do trabalho de conclusão de curso no qual é proposto um modelo conceitual de um ambiente para incorporar aplicações voltadas à TV digital, em virtude de não se ter um padrão ou modelo de desenvolvimento que trate a forma com que são criadas aplicações que se agregam a ela, como um utilitário ou serviço. Para que fosse possível criar uma proposta de *software* por meio de padrões, foi realizado um estudo sobre a TV digital brasileira.
Com base no modelo desenvolvido, pôde-se implementar uma aplicação que atendeu a requisitos definidos previamente. Essa aplicação fez uso das linguagens de programação NCL e Lua que fazem parte da especificação do *middleware* Ginga e foi denominada *TV Digital Deck*. Mesmo com algumas limitações, obteve-se um resultado satisfatório, que pode incentivar novos trabalhos de especificação e também desenvolvimento.

Palavras-chave: TV Digital. Ginga. *Middleware*. NCL. Lua.

Abstract. This article presents the result of completion of course work which proposed a conceptual model of an environment for applications directed to incorporate digital TV due to not having a standard or model of development that addresses the way applications are created that aggregate to it as a utility or service. To enable it to create a proposal for software through standards, a study was conducted on the Brazilian digital TV.
Based on the model developed, we could implement an application that met the requirements defined in advance. This application made use of programming languages NCL and Moon that are part of the specification of the middleware and was named Digital TV Deck. Even with some limitations, we obtained a satisfactory result, which may encourage further work to specification and also development.

Keywords: Digital TV. Ginga. *Middleware*. NCL. Lua

1 Introdução

A TV digital é uma nova tecnologia de transmissão de sinais de televisão, que proporcionará ao telespectador melhor qualidade de imagens, sons e uma série de novos benefícios tais como interatividade com os programas, também a possibilidade de utilização em dispositivos móveis.

Este artigo tem por objetivo apresentar o resultado do trabalho de conclusão de curso que propôs e desenvolveu um sistema para incorporação e gerenciamento de novas aplicações desenvolvidas (*gadgets*) para a TV Digital. A justificativa para a escolha do trabalho foi a falta de um padrão ou modelo de desenvolvimento que tratasse a forma com que são criadas aplicações que se agregam à TV como um utilitário ou serviço. No trabalho foi proposto um modelo conceitual de um ambiente para incorporar essas aplicações que foram denominadas *gadgets*.

Gadget (em inglês: geringonça, dispositivo) na área de tecnologia da informação é o nome que se dá a algum pequeno *software*, ferramenta ou serviço que pode ser agregado a uma aplicação ou ambiente maior (MELLO, 2009).

Com base em toda proposição de modelos e padrões tanto para o software como para os *gadgets*, o trabalho expõe uma implementação de um ambiente de incorporação chamado de *TV Digital Deck* seguindo as especificações contidas no modelo.

Foi apresentado como objetivo do trabalho a realização de um estudo sobre a TV digital brasileira, sobre seus padrões, especificações de desenvolvimento, plataformas e *middleware*. Expostas as linguagens de programação suportadas pelo *middleware* Ginga: NCL e Lua. Especificação por meio de um modelo conceitual o software proposto e o Desenvolvimento da aplicação segundo a especificação e modelagem conceitual criada. O artigo contemplará os principais pontos da pesquisa sendo que maiores detalhes poderão ser vistos no próprio trabalho.

2 Tecnologias e conceitos da TV Digital

Com a tecnologia de transmissão e recepção digital, maior quantidade de conteúdo pode ser transmitida em uma mesma freqüência, sendo assim, além de vídeo e áudio, dados também são transmitidos, expandindo as possibilidades e funções desse sistema, criando uma

capacidade computacional e surgimento de serviços por aplicações construídas (SOARES; BARBOSA, 2009).

A qualidade do áudio transmitido também é percebida com a possibilidade de se reproduzir áudio no padrão 5.1 (multicanal) propiciando sensação de imersão na cena transmitida.

O *Middleware* é um programa de computador que faz a mediação, entre o sistema operacional e a aplicação. Um *middleware* para a TV Digital interativa oferece uma gama de funções que permite o desenvolvimento facilitado de aplicações e suporte a múltiplos dispositivos de exibição “Do ponto de vista do software, podemos dizer, sem exagero, que ao definir o *middleware* estamos, de fato, definindo um sistema de televisão” (SOARES; BARBOSA, 2009, p.22). É uma nova camada que segue os padrões de referência da TV Digital fornecendo através de uma interface de programação, suporte às aplicações a serem executadas no ambiente de TV Digital.

Ginga é o nome do *middleware* desenvolvido para o sistema brasileiro de TV digital, é uma especificação aberta que adota a licença GPLV2. Uma camada de software que está sendo instalada nos conversores e em televisores funcionando independentemente do sistema operacional da plataforma de hardware utilizado. O *middleware* oferecerá suporte ao desenvolvimento de aplicações interativas para a TV Digital com uma gama de funções que permitem o desenvolvimento facilitado de aplicações. Essas aplicações podem possibilitar, por exemplo, acesso à internet, operações bancárias, entre outras operações (GINGA DIGITAL TV MIDDLEWARE SPECIFICATION, 2009).

3 Linguagens de Programação

O Sistema Brasileiro de TV Digital Terrestre (SBTVD-T) tem como uma das principais inovações o *middleware* Ginga. Ele provê suporte para o desenvolvimento de aplicações em NCL, Lua e Java. A linguagem Java não será abordada, pois ficou fora do escopo em função dê não ter sido encontrado um simulador do *middleware* Ginga que atendesse todas as expectativas e fornecesse suporte às aplicações em Java.

NCL (*Nested Context Language*) é uma linguagem declarativa, um XML seguindo o padrão NCM (*Nested Context Model*), que especifica documentos de hipermídia onde é possível descrever objetos de mídia e seu comportamento temporal e espacial. Por objeto de

mídia entende-se texto, imagem, áudio, vídeo, objetos com código Lua, objetos com código Java, objetos com código HTML etc (SOARES, 2006).

A NCL sendo uma linguagem declarativa é baseada em um modelo conceitual de dados, o NCM, que representa os conceitos estruturais, os relacionamentos, regras estruturais e operações.

Basicamente compõe-se de um *nó* NCM que possui um identificador, um conteúdo e um conjunto de âncoras. Uma âncora é um subconjunto das unidades de informação de um *nó* e são definidas separadamente do conteúdo de um *nó*. Um *nó de mídia* representa um *objeto de mídia* (SOARES; BARBOSA, 2009).

Lua é uma linguagem de programação desenvolvida para ser usada em conjunto com outras linguagens com o propósito de estender as funcionalidades da aplicação principal (THE PROGRAMMING LANGUAGE LUA, 2009). Uma linguagem de fácil aprendizado, desenvolvida pela PUC-Rio que tem como características: simplicidade, portabilidade, tipagem dinâmica, semântica extensível, interpretada a partir de *bytecodes* e possui gerenciamento automático de memória com coleta de lixo incremental. Lua é livre de código aberto podendo ser usada para quaisquer propósitos comerciais ou não.

Lua é usada em muitas aplicações industriais com ênfase em sistemas embutidos e jogos. Atualmente a linguagem de script mais usada em jogos (THE PROGRAMMING LANGUAGE LUA, 2009).

Em função de suas características, Lua foi a linguagem de *script* escolhida para atuar junto à linguagem NCL complementando-a.

O ciclo de vida de um objeto Lua é controlado pelo documento NCL, porém, diferentemente das mídias, os *scripts* são guiados por eventos, que podem ser oriundos do NCL ou de fontes externas de interação como o controle remoto. O documento NCL pode definir em que momento o *script* vai ser executado, podendo encerrar no tempo determinado ou com algum evento de encerramento ou até durar por todo o período de execução vida da aplicação estando disponível caso seja requisitado novamente (SOARES; BARBOSA, 2009).

4 Projeto do Sistema

Para possibilitar a incorporação de outras aplicações denominadas *gadgets*, foi proposta a criação de um *software* para a TV Digital, servindo como um ambiente personalizado de utilitários.

O *software* desenvolvido é do tipo residente, portanto, uma aplicação que independe do canal que está sintonizado, assim a seleção e uso dos *gadgets* se tornam dinâmicas, podendo ser utilizadas a qualquer momento.

Os *gadgets* poderiam ser adquiridos pelo sinal digital enviado pela emissora ou pela internet por meio do canal de retorno. E assim a aplicação faria uso dos *gadgets* que o próprio usuário escolheu.

A aplicação funciona em dois momentos: no primeiro momento o *software* localiza todos os *gadgets* disponíveis em um diretório padrão dentro do Ginga, para cada utilitário encontrado, é feita uma leitura e coleta de dados relacionados para uso posterior. Com todas as informações necessárias será montada dinamicamente a segunda parte da aplicação, que consiste no ambiente para execução. Essa etapa de execução terá todas as referências aos *gadgets* lidos para assim poder executá-los. O ambiente de execução deverá conter um menu listando as opções para seleção e uso. Os *gadgets* serão listados e o usuário fará sua escolha por meio do controle remoto.

Com base nos dois momentos, a aplicação seria estruturada em dois módulos: carregamento e execução.

Toda leitura do diretório e coleta de informações dos *gadgets* foi feita no módulo de carregamento, que atua no primeiro momento da aplicação lendo todos os utilitários existentes, e cada um será coletado os dados necessários, montando o módulo de execução resultante. Assim, o *software* fica pronto para ser executado pelo próprio usuário telespectador.

O módulo de execução é responsável por interpretar toda a estrutura da aplicação como o menu e o acionamento dos *gadgets* incorporados, bem como o resultado do carregamento feito no primeiro módulo.

Os *gadgets* são desenvolvidos seguindo um padrão que garanta a compatibilidade com o ambiente de gerenciamento. Para essa aplicação, foi criado um padrão definindo que meta dados deverão estar presentes nos utilitários, com informações pertinentes de uso dos recursos e configurações.

5 TV Digital Deck

A aplicação foi implementada com base no modelo conceitual proposto e foi chamada de *TV Digital Deck*, desenvolvida em NCL e Lua utilizando a ferramenta de desenvolvimento Eclipse com um *plugin* adicional para NCL e outro para Lua. O *TV Digital Deck* é composto de três arquivos: *proLoader.ncl*, *createMenu.lua* e *resultApp.ncl*. Além desses três arquivos desenvolvidos, a aplicação ainda fez uso de outras duas API´s de funções em Lua: *xml.lua*, que provê funções para fazer o *parser* de documentos XML e *handler.lua,* que atua junto ao *xml.lua* com funções de manipulação. Uma imagem de fundo no formato PNG e dois vídeos no formato MP4 também compõem a aplicação.

A execução do *TV Digital Deck* ocorre em dois momentos. O primeiro momento consiste em carregar e montar o documento resultante com base nas informações presentes em um diretório específico contendo os *gadgets* que serão utilizados. Os *gadgets* encontrados possuem meta dados que serão colhidos e utilizados para a criação desse documento resultante, que depois de pronto conterá um menu com referências para todos os *gadgets*, podendo executá-los conforme critério do usuário. O segundo momento é a execução do *resultApp.ncl* onde dois vídeos são reproduzidos simultaneamente, um deles “escondido”, simulando dois canais diferentes de televisão. Ao acionar as teclas de seta para cima e para baixo do controle remoto ocorre a troca de exibição de um canal para outro, sem afetar o restante da aplicação, no caso, o menu dos *gadgets*. Inicialmente o menu não é apresentado, podendo ser acionado através do controle remoto, nele conterá atalhos para os *gadgets* carregados que poderão ser selecionados e executados.

O módulo de carregamento foi implementado em um documento NCL chamado de *proLoader.ncl.* O *proLoader.ncl* é bastante simples, tendo somente dois objetos de mídia: o *background.png*, que é a imagem de fundo exibida durante a execução e o *createMenu.lua*, um objeto NCLua que fará a montagem do terceiro arquivo: o *resultApp.ncl*.

O módulo de execução fé criado dinamicamente pelo módulo de carregamento por meio do arquivo *createMenu.lua* que é responsável por procurar e armazenar em uma lista as URI dos *gadgets* encontrados no diretório *myGadgets*. Essa lista tem o nome de *sourceList.txt*, que é apenas um arquivo de texto com a relação de todos os endereços dos *gadgets* encontrados. Depois da lista montada, cada item é lido e utilizado para buscar o documento correspondente ao endereço. A busca é feita por meio da URI do documento NCL, que é mapeado para um XML DOM e depois transformado em uma tabela Lua. De posse

dessa tabela é possível extrair os meta dados presentes no documento e com eles dar início à criação do *resultApp.ncl*.

As informações conseguidas por meio dos meta dados servem para a criação de *tags* NCL, que são incorporadas no documento *resultApp.ncl*. As *tags* criadas dizem respeito às estruturas de importação do documento original, região e descritor dos ícones que representam o *gadget* no menu, contexto dos *gadgets* para poderem ser executados, objetos de mídia dos ícones de representação dos *gadgets* e elos de início e parada dos elementos, do menu e da aplicação como um todo.

O arquivo resultante é inicialmente um documento NCL simples, sem regiões, descritores, objetos, apenas a estrutura básica que caracteriza um NCL: cabeçalho e corpo. Após a execução do código Lua *createMenu.lua* o documento *resultApp.ncl* é totalmente sobrescrito, resultando em um documento novo com toda estrutura proposta do *TV Digital Deck*, menu, *gadgets* e vídeos, pronto para ser executado

6 Conclusões

Ainda que não totalmente definido o potencial de interatividade da TV Digital brasileira, já é possível desenvolver aplicações de todos os tipos fazendo uso de especificações de linguagens e outros recursos suportados pelo *middleware* Ginga. Dentro dessas possibilidades, pôde-se verificar na construção do modelo conceitual de desenvolvimento e em funcionamento por meio de sua implementação, que mesmo com algumas limitações, obteve-se um resultado satisfatório, que pode incentivar novos trabalhos de especificação e também desenvolvimento.

Para que fosse possível criar uma proposta de *software* por meio de padrões e um modelo conceitual, foi realizado um estudo sobre a TV digital brasileira.

Com base no modelo desenvolvido, pôde-se implementar uma aplicação que atendeu a requisitos definidos previamente. Essa aplicação fez uso das linguagens de programação NCL e Lua que fazem parte da especificação do *middleware* Ginga.

As linguagens NCL e Lua foram analisadas de maneira que possibilitaram a extração de informações a respeito de formas para atender os requisitos do modelo, deixando evidente o potencial das duas linguagens.

A aplicação resultou num *software,* que incorpora outros *softwares* denominados *gadgets* e com isso organiza, gerencia e personaliza esses utilitários, contribuindo para a interatividade proposta da TV Digital brasileira.

Melhorias no modelo podem ser feitas, enriquecendo a especificação e dando continuidade a um desenvolvimento organizado dentro do ambiente da TV Digital.

A implementação criada a partir da estrutura conceitual apresentada servirá de ponte para utilização dos *gadgets*, que serão desenvolvidos seguindo a especificação e modelos de utilitários.

Para trabalhos futuros poderá ser modelado e implementado a possibilidade de obtenção dos *gadgets* por meio do sinal digital televisivo, e não apenas pelo canal de retorno.

7 Referências

ABNT NBR 15606-2 (2007). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre – codificação de dados e especificações de transmissão para rádiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações". Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ABNT NBR 15606-2 (2007). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre – objetos procedurais Lua em apresentações NCL". Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

GINGA DIGITAL TV MIDDLEWARE SPECIFICATION. Disponível em: <http://www.ginga.org.br/> Acesso em novembro de 2010.

MELLO, Laurentino. **Gadget. Saiba o que é...** Disponível em: <http://www.tinotec.com.br/blog/gadget-saiba-o-que-e/> Acesso em novembro de 2010.

SOARES, Luiz Fernando Gomes. **TV interativa se faz com Ginga.** Disponível em: <http://www.gingancl.org.br/resources/Encarte-mod.pdf> Acesso em 23/10/2010.

SOARES, Luiz Fernando Gomes; BARBOSA, Simone Diniz Junqueira. **Programando em NCL 3.0: Desenvolvimento de aplicações para o middleware Ginga. RJ: Elsevier, 2009.**

SOARES, L.F.G.; RODRIGUES, F.R.; BARBOSA, J.D.S. **Manual de construção de programas audiovisuais interativos utilizando a NCL 2.3** – Perfil básico. Disponível em: <http://www.ncl.org.br/documentos/manualNCL2_3.pdf> Acesso em: 23/10/2010.

THE PROGRAMMING LANGUAGE LUA. Disponível em: <http://www.lua.org/portugues.html> Acesso em novembro de 2010.