UNIVERSIDADE DE PERNAMBUCO

Faculdade de Ciências e Tecnologia de Caruaru

Bacharelado em Sistemas de Informação

VICTOR LAERTE DE OLIVEIRA

TV DIGITAL NO BRASIL: UMA METODOLOGIA PRÁTICA PARA O DESENVOLVIMENTO DE APLICAÇÕES INTERATIVAS UTILIZANDO GINGA-NCL

ORIENTADOR: FERNANDO FERREIRA DE CARVALHO

CARUARU

2011

VICTOR LAERTE DE OLIVEIRA

TV DIGITAL NO BRASIL: UMA METODOLOGIA PRÁTICA PARA O DESENVOLVIMENTO DE APLICAÇÕES INTERATIVAS UTILIZANDO GINGA-NCL

Monografia apresentada como requisito parcial para a obtenção do diploma de Bacharel em Sistemas de Informação pela Faculdade de Ciência e Tecnologia de Caruaru - Universidade de Pernambuco.

CARUARU

2011

Monografia de Graduação apresentada por **Victor Laerte de Oliveira** do Curso de Graduação em Sistemas de Informação da Faculdade de Ciência e Tecnologia de Caruaru – Universidade de Pernambuco, sob o título **"TV Digital no Brasil: Uma Metodologia Prática para o Desenvolvimento de Aplicações Interativas Utilizando Ginga-NCL"**, orientada pelo Prof. **Fernando Ferreira de Carvalho** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Alexandre Magno A. Maciel
Departamento de Sistemas de Informação / UPE

Prof. Fernando Ferreira de Carvalho
Departamento de Sistemas de Informação / UPE

Visto e permitida a impressão.
Caruaru, 21 de dezembro de 2011.

Prof. Fernando Ferreira de Carvalho
Coordenador do Curso de Bacharelado em Sistemas de Informação da
Faculdade de Ciência e Tecnologia de Caruaru – Universidade de Pernambuco.

“Sofremos muito com o pouco que nos falta e gozamos pouco o muito que temos”

(William Shakespeare)

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por ter me dado saúde, sabedoria e perseverança na longa caminhada da graduação, por ter me guiado e iluminado nas horas mais difíceis e por ter me presenteado com a melhor das famílias. Ó Deus dos meus antepassados, eu te agradeço e louvo, pois me deste sabedoria e poder (Daniel 2:23).

À minha família, especialmente aos meus pais Sebastião Severino e Grace Monica, por todo amor e apoio que me deram, e sem o qual eu não estaria aqui.

À minha irmã Priscila Monica, pelo coração extremamente generoso que muitas vezes me incentivou nesta caminhada e pelo simples fato de existir e fazer parte de minha vida.

Aos meus avós maternos Domingos e Maria, pelas inúmeras vezes que me aconselharam a estudar e procurar sempre o meu melhor.

Aos meus tios, em especial meu Tio Glauber e sua esposa Taciana que torcem por mim como torceriam pelo seus próprios filhos, e a minha Tia Shirley e seu esposo Ed Carlos pelos conselhos e palavras de incentivo.

À minha namorada e companheira Marianna Melo, que me auxiliou na revisão gramatical de todo o trabalho e que com certeza foi uma das que mais sofreu com minha falta de tempo neste fim de processo.

Aos companheiros de jornada Waldeildo, Bartholomeu, Elisson, Diógenes, Nathália, Laura, João, Fábio, Bruno Augusto, Bruno Freitas e a todos os outros colegas da UPE (são muitos).

Aos meus amigos pessoais, Bruno (Cabeça), César, Thiago (meu futuro patrão), Paulo, Felipe Neiva (Mago), Thiago (Vacão), Ricardo (amigo que namora e some) e a Edinaldo (Amigo de longas datas) e Walter (Meu grande irmão) que me auxiliaram na arte de alguns componentes do trabalho.

A todos vocês que contribuíram direta ou indiretamente com a minha formação, meus sinceros agradecimentos.

RESUMO

Os conceitos a respeito das tecnologias envolvidas na TV digital aberta brasileira ainda carecem de fontes de informações e ferramentas de suporte a estudantes e profissionais da área da informática. Por esta razão, surge a necessidade da criação de um material de referência contemplando, além dos conceitos, uma metodologia prática para o desenvolvimento de aplicações interativas. Neste trabalho são definidas as principais arquiteturas e conceitos técnicos do Sistema Brasileiro de Televisão Digital(SBTVD). Além disso, são apresentadas as linguagens de programação e ambientes de desenvolvimentos, dando ênfase à preparação de um alicerce para a implementação de uma aplicação interativa utilizando o subsistema declarativo Ginga-NCL, que servirá para avaliar o conhecimento dos telespectadores de programas que exibam material didático por meio de um questionário sobre o tema envolvido.

Palavras-chave: TV Digital, NCL, Lua, Desenvolvimento de aplicações interativas

ABSTRACT

The concepts about the technologies involved in the Brazilian Digital TV still require open information sources and tools to support students and professionals in the field of information technology. For this reason, there is the need to create a reference material covering, and concepts, a practical methodology for developing interactive applications. This paper sets out the key technical concepts and architectures of the Brazilian Digital Television System (SBTVD). In addition, we present the programming languages and development environments, emphasizing the preparation of a foundation for the implementation of an interactive application using declarative subsystem Ginga-NCL, which will serve to assess the knowledge of the viewers of programs that display educational materials by means of a questionnaire on the topic involved.

Keywords: Digital TV, NCL, Lua, Development of interactive applications

FIGURAS

QUADROS

ALGORITMOS

ACRÔNIMOS E SIGLAS

ABERT/SET – Associação Brasileira de Emissoras de Rádio e Televisão / Sociedade Brasileira de Engenharia de TV e Telecomunicações

ABNT – Associação Brasileira de Normas Técnicas

API – Application Programming Interface

ATSC – Advanced Television System Committee

BST-OFDM – Band Segmented Orthogonal Frequency Division Multiplexing

CPU – Central Processing Unit

DVB-T – Digital Video Broadcasting Television

FTP – File Transfer Protocol

HDTV – High Definition Television

IBGE – Instituto Brasileiro de Geografia e Estatística

IDE – Integrated Development Environment

ISDB-T – Integrated Services Digital Broadcasting Terrestrial

NCL – Nested Context Language

NCM – Nested Context Model

SBTVD – Sistema Brasileiro de Televisão Digital

SDTV – Standard Definition Television

SSH – Secure Shell

TQTVD – TOTVS Quality para TV Digital

TVD – Televisão Digital

XML – Extensible Markup Language

SUMÁRIO

1. INTRODUÇÃO

Desde os tempos primitivos o homem sente necessidade de expressar de diversas formas suas emoções e experiências cotidianas. Segundo Valim (1998) a idéia de trabalhar com imagens data do inicio da civilização. Já nos tempos primitivos, o homem se expressava por meio de desenhos para que gerações posteriores pudessem aprender ou os reverenciar. Com a evolução das técnicas de pintura a humanidade passou a expressar suas idéias de forma mais fiel, algo que se concretizou ainda mais com o avanço tecnológico e com o surgimento da fotografia.

Com a idéia de projeção de imagens estáticas em sequência, o cinema deu vida aos quadros parados, e assim ficamos muito próximos da reprodução da realidade.

Com a invenção da televisão, e sua praticidade de estar dentro dos lares, a TV tornou-se o maior e mais poderoso meio de comunicação, transmissor de informações e idéias, do mundo.

Segundo pesquisa realizada pelo IBGE (IBGE, 2009), cerca de 95,1% da população brasileira possui um aparelho de TV, perdendo somente para o fogão como eletrodoméstico de maior prioridade para os brasileiros. Diante disso, nota-se que o brasileiro tem na televisão a maior fonte de contato com o mundo exterior e de informação, independente de classe social.

A TV digital (TVD) surge no Brasil como uma poderosa ferramenta de inclusão digital. O governo aposta que com o advento da interatividade na TV, brasileiros que não tem acesso a recursos tecnológicos como um computador ou internet, possam ser inseridos no contexto tecnológico global, aprendendo a interagir com novas ferramentas e aplicativos, até então desconhecidos.

O processo de transição da TV analógica para digital é um processo lento que ainda está ocorrendo e envolve uma série de esforços do governo, entre eles definir os inúmeros padrões necessários para transmissão do sinal e para criação de aplicativos interativos. O Sistema Brasileiro de Televisão Digital (SBTVD) é o padrão

técnico para teledifusão digital no Brasil e entrou em operação no dia 2 de dezembro de 2007.

No SBTVD o *middleware*, dispositivo responsável por fazer a mediação entre o software da TV e os aplicativos feitos pelos desenvolvedores, é o Ginga, desenvolvido totalmente no Brasil. No Ginga existem basicamente dois tipos de ambientes de desenvolvimento, declarativo e o não-declarativo e é considerado o melhor *middleware* para televisão digital existente no momento, permitindo o uso de aplicações bastante complexas de interatividade em TV (GINGA, 2010).

1.1 Problema de Pesquisa

Espera-se ao final deste trabalho ter adquirido subsídios suficientes para encontrar a resposta da grande questão inspiradora: Quais as etapas para o desenvolvimento de aplicações interativas na TV digital brasileira?

É com base nesta indagação que o trabalho foi proposto, e é nela que toda pesquisa irá se basear.

1.2 Objetivos

Os objetivos da pesquisa dividem-se em geral e específicos, os quais serão definidos a seguir.

1.2.1 Objetivo Geral

Desenvolver uma metodologia prática para a criação de aplicativos interativos para TV digital, utilizando o subsistema Ginga-NCL dentro dos padrões brasileiros.

1.2.2 Objetivos Específicos

Com finalidade de atingir o objetivo geral, são traçados os seguintes objetivos específicos:

- Apresentar o modelo de TV digital brasileira;
- Apresentar as linguagens e ambientes de programação disponíveis;
- Demonstrar as etapas necessárias para a criação de aplicações interativas;

- Desenvolver uma aplicação interativa que sirva para avaliar o conhecimento dos usuários de programas educacionais, por meio de questionários sobre os temas envolvidos.

1.3 Justificativa

A chegada da TV digital tem sido o assunto mais discutido no meio acadêmico na última década. Com a demorada e constante transição da TV analógica para a digital no Brasil, as fontes de informações e ferramentas de suporte a estudantes e profissionais da área da informática ainda são poucas.

Academicamente este estudo é relevante, pois irá explanar os conceitos da TV digital e apresentar de forma geral os ambientes de desenvolvimento de aplicações interativas.

Em sua contribuição prática esta pesquisa abordará a criação de uma metodologia prática para o posterior desenvolvimento de um aplicativo interativo declarativo, utilizando Ginga-NCL, que servirá para avaliar o conhecimento dos telespectadores de programas que exibam material didático por meio de um questionário sobre o tema envolvido.

A pesquisa se justifica pela carência e dispersão de informações sobre os padrões brasileiros e ambientes de desenvolvimento utilizados na criação de aplicações para TV digital. Os resultados poderiam servir como solução pela comunidade de desenvolvedores que desejam aderir à programação de aplicativos interativos e encontram dificuldades.

Portanto, o tema discutido é de extrema relevância para criação de um alicerce ao desenvolvimento de aplicações interativas para TV digital, necessário para estudos mais amplos visto que, o tema abordado é extremamente vasto.

1.4 Escopo Negativo

A proposta deste trabalho está inserida em um contexto mais amplo, portanto faz-se necessário tratar alguns aspectos que não estão relacionados no escopo deste trabalho.

Ressaltamos que os seguintes aspectos não fazem parte do escopo deste trabalho:

- Abordagem prática do *middleware* Ginga-J: mesmo sendo dada uma visão geral sobre o *middleware* Ginga-J o foco do trabalho se restringe a criação de uma metodologia para o *middleware* Ginga-NCL.
- Criação de *frameworks*: apesar do desenvolvimento de aplicações ser abordado, este trabalho não propõe a criação de nenhum framework.
- Abordagem detalhada de aspectos relacionados a Engenharia de Software: embora seja feita uma breve apresentação do requisitos essenciais do desenvolvimento de uma aplicação, não serão tratadas, com detalhes, os aspectos de especificações de Engenharia de Software.

2. REFERÊNCIAL TEÓRICO

2.1 Sistema de TV Digital

A vantagem inicial mais notável na transição da TV analógica para a digital foi a mudança significativa da qualidade de imagem e som. No sistema de transmissão de TV (digital ou analógico), o canal de transmissão, em sua maioria o ar, introduz diversas interferências e ruídos no sinal original transmitido.

> O ruído aleatório está presente em todo o espectro de freqüências e não pode ser evitado. Na transmissão analógica, ele provoca queda na qualidade do sinal recebido, causando o aparecimento de "chuviscos" na imagem. A queda da qualidade depende da relação entre as potências do sinal e do ruído (relação S/N). À medida que a relação diminui, e isso pode acontecer pela atenuação do sinal quando nos afastamos da fonte, diminui também a qualidade do sinal recebido. Nos sistemas digitais, o ruído pode modificar um nível digital do sinal transmitido a ponto que ele passa a ser confundido com outro nível, aumentando a probabilidade de erro de bit. (BARBOSA e SOARES, 2008, p. 106).

Para manter a qualidade da imagem nos sistemas digitais utiliza-se um código corretor, capaz de corrigir os erros causados pela interferência no canal. Se a taxa de erros estiver acima do limite de correção, o sistema não é capaz de corrigi-los levando à queda da recepção. Isso faz com que na transmissão digital ou temos uma imagem perfeita ou não temos nenhuma imagem.

Segundo Montez e Becker (MONTEZ e BECKER, 2004) existem duas modalidades mais conhecidas de TV digital. A SDTV (*Standard Definition Television*) com serviço de áudio e vídeo digitais, parecida com a TV analógica com aspecto 4:3 (largura:altura da imagem), cujos aparelhos receptores possuem 408 linhas, com 704 pontos em cada uma. E a HDTV (*High Definition Television*), cuja imagem possui formato 16:9, é recebida em aparelhos com 1080 linhas de definição e 1920 pontos.

Outra característica que faz a qualidade da imagem da TV digital ser superior a imagem da TV analógica é a compressão de dados. Comprimindo-se os sinais de áudio e vídeo é possível obter uma menor taxa de transmissão, possibilitando um maior tráfego de dados em uma única faixa de freqüência.

Assim tornou-se possível a multiprogramação, onde em um mesmo canal são transmitidos vários programas ao invés de apenas um. Em um jogo de futebol,

por exemplo, poderá ser transmitida a imagem da câmera principal e além dela, imagens de uma câmera atrás do gol ou na arquibancada, e o telespectador terá a possibilidade de escolher qual a melhor posição para assistir. Nesse caso o conteúdo das transmissões simultâneas está relacionado, todas são da mesma partida de futebol. Porém, é possível transmitir programas não relacionados, dando ao telespectador um vasto leque de opções de programação.

2.1.1 Visão Geral

Um programa de TV digital é composto de diversos dados transmitidos em conjunto. Esses dados podem ser vídeos, áudio, textos, imagens, etc. Com esse novo formato de transmissão o programa televisivo deixa de ser contínuo em sua concepção e começa a ter vários caminhos alternativos de exibição, dando a idéia de não-linearidade.

Para o processamento de todos esses dados é necessário uma capacidade computacional inexistente na TV Analógica. Os novos aparelhos de TV desenvolvidos para a transmissão digital já trazem esses sistemas computacionais integrados, porém para dar a possibilidade das TVs analógicas receberem um sinal digital foram criados sistemas de processamento, chamados de conversores digitais (ou *set-top box*).

> Conectada a um terminal de acesso (set-top-box), a televisão digital permitirá que os telespectadores interajam com diversos programas e acessem inúmeras informações e serviços. Através da interatividade, conteúdos e serviços não disponíveis atualmente e sequer imaginados poderão ser disponibilizados. (ZANCANARO et. al., 2009, p. 2)

O conversor digital pode ser embutido no aparelho ou não, porém sua função é a mesma. O sinal recebido, depois de demodulado (sintonizado e retirado do canal de frequência), é separado (demultiplexado) e entregue para os decodificadores de áudio e vídeo, para processamento em uma CPU. O receptor tem acesso a uma rede externa através da qual pode enviar ou receber dados. Este canal de retorno é também chamado de canal de interatividade.

2.1.2 Interatividade

A interatividade é a troca entre o usuário de um sistema informático e a máquina por meio de um terminal dotado de tela de visualização. (Koogan/Houaiss 1999).

Assim, podemos entender por interatividade, a forma de participar de algo modificando o seu conteúdo. É comum que muito dos programas de TV se autodenominem interativos, pois segundo Lemos (1997) o adjetivo interativo torna o produto a ser comercializado mais moderno aos olhos do consumidor. Porém, se analisarmos a interatividade com base em seus conceitos podemos notar que programas que se dizem interativos como os “reality shows” são na realidade programas reativos, pois os telespectadores apenas reagem a estímulos oferecidos pela emissora. Não tendo um papel ativo em relação à programação.

Para Lemos (1997) a interatividade na televisão se da em cinco níveis distintos:

- Nível 0 – Possibilidade apenas da troca de canal, ou a regulagem de volume, contraste, brilho e ligar ou desligar o aparelho de televisão. A transmissão ocorre em preto e branco, com a exibição de dois canais apenas.
- Nível 1 – Nascimento da televisão colorida, e outras emissoras. O aumento de canais traz consigo o controle remoto que vem suprir a demanda de conforto requerida pelos usuários, além da possibilidade de fazer ajustes na forma como a programação é assistida.
- Nível 2 – A TV passa a poder ser utilizado para outros fins, além de assistir os programas transmitidos pelas emissoras de televisão o usuário tem acesso a jogos eletrônicos, vídeo-cassete e câmeras portáteis que possibilitam que o telespectador utilizar a TV para jogar ou simplesmente assistir a filmagens previamente gravadas. O vídeo-cassete ainda possibilita o usuário gravar os programas exibidos e assisti-los quando bem desejar.
- Nível 3 – Surge os primeiros sinais de interatividade digital, o telespectador pode interferir no conteúdo na programação através de um canal alternativo de retorno como fax, telefone ou mensagens de correio eletrônico (*email*).

- Nível 4 – Neste nível surge a TV interativa, pois o telespectador pode utilizar o controle remoto e interferir na programação, escolhendo cenas ou ângulos de câmeras.

Becker e Montez (MONTEZ e BECKER, 2004) acrescentam ainda mais três níveis de interação:

- Nível 5 – O telespectador pode participar da programação, mandando vídeos de baixa qualidade. Agora se tem a necessidade de um canal de retorno ou canal de interação que tenha capacidade de dar os recursos para a transmissão do vídeo do telespectador para a emissora.
- Nível 6 – Nesse nível temos os mesmos recursos que no nível 5 e mais a permissão da transmissão de vídeos de alta qualidade. O canal de retorno ou canal de interatividade deve, por obrigatoriedade, dispor de banda superior à oferecida no nível anterior.
- Nível 7 – Agora o telespectador chega à interatividade plena, criação de conteúdo igual ao da emissora. “Cai por terra” o monopólio de produção e veiculação das redes de televisão, o usuário passa a atuar como se fosse um internauta na *Web,* com capacidade e recursos necessários à publicação de sites com o conteúdo que preferir.

Apesar de haver controvérsias entre diversos autores sobre os conceitos da TV interativa, uma característica é praticamente unânime: a TV deixa de ser unidirecional. Com a interatividade, o telespectador passará a se comunicar com a emissora.

2.2 Sistema Brasileiro de TV Digital

O Sistema Brasileiro de TV Digital (ou SBTVD) é um padrão técnico para teledifusão da TV digital, inspirado no padrão japonês (*Integrated Services Digital Broadcasting Terrestrial* - ISDB-T), criado e utilizado pelo Brasil e adotado em diversos países da América do Sul. Foi instituído pelo decreto presidencial 4.901, em 26 de novembro de 2003, e entrou em operação comercial em 2 de dezembro de 2007.

> O Sistema Brasileiro de Televisão Digital (SBTVD) foi criado com o objetivo de não fazer apenas a troca de equipamentos, mas de garantir a inclusão digital através dos novos recursos potenciais de interatividade da TV digital, inclusive, no futuro, o acesso à Internet. (CROCOMO, 2004, p. 56)

Surge então como uma forma de democratização no acesso à informação, possível através da interatividade e também como arma para combater o analfabetismo digital ainda bastante presente na sociedade brasileira.

O SBTVD teve início como um grupo de trabalho estruturado para rever estudos iniciais feitos pela Associação Brasileira de Emissoras de Rádio e Televisão e Sociedade Brasileira de Engenharia de TV e Telecomunicações (ABERT/SET) e apoiado pela Universidade Presbiteriana Mackenzie sobre os padrões de TV digital existentes no mundo.

Foram analisados os padrões ATSC (*Advanced Television System Committee* - Americano), DVB-T (*Digital Video Broadcasting Television* - Europeu) e ISDB-T (Japonês). De acordo com os testes realizados pelo grupo ABERT/SET (ABERT/SET 2000) o padrão ATSC demonstrou qualidade insuficiente para recepções com antena interna, fator de grande relevância para o Brasil onde 47% das recepções são feita desta forma. Entre o DVT-T e o ISDB-T o último apresenta desempenho superior na recepção em ambiente fechado e na flexibilidade de acesso aos serviços digitais e programas de TV através de receptores fixos, móveis e portáteis com qualidade impressionante.

Diante do bom desempenho mostrado pelo padrão ISDB-T ele foi escolhido para ser utilizado como base no desenvolvimento dos padrões do SBTVD, acrescido de algumas novas tecnologias.

Existem cerca de 16 documentos técnicos para o padrão SBTVD, com mais de 3.000 mil páginas publicadas pela ABNT (Associação Brasileira de Normas Técnicas) e pelo fórum do SBTVD detalhando todo o padrão (WIKIPEDIA, 2011).

A Figura 1 mostra um resumo dos padrões definidos pelo SBTV e os mais relevantes são detalhados a seguir.

Codificação de canal de transmissão	Esquema de Modulação	64QAM-OFDM, 16QAM-OFDM, QPSK-OFDM, DQPSK-OFDM (Transmissão Hierárquica)
	Código para Correção de Erro	Inner coding Convolução 7/8,3/4,2/3,1/2 Outer coding:RS(204,188)
	Intervalo de Guarda	1/16,1/8,1/4
	Intercalação de símbolos	Por Tempo, por Frequência, bit, byte
	Tipo de Modulação	BST-OFDM (Estrutura Segmentada OFDM - 13 segmentos)
Acesso Condicional		Multi-2
Middleware		Middleware Ginga: Ginga-NCL (ambiente declarativo) e Ginga-J (ambiente procedural)
Informação do Serviço		ARIB STD B-10
Multiplexação		MPEG-2 Systems
Codificação de Áudio	Fixo/Móvel	Estéreo: MPEG-4 AAC@L2 ou MPEG-4 HE-AAC v1@L2 Multicanal 5.1: MPEG-4 AAC@L4 ou MPEG-4 HE-AAC v1@L4
	Portátil	Estéreo apenas: MPEG-4 HE-AAC v2@L2
Codificação de Vídeo	Fixo/Móvel	MPEG-4 AVC (H.264) HP@L4
	Portátil	MPEG-4 AVC (H.264) BP@L1.3

Figura 1. Resumo dos padrões técnicos do SBTVD (Fonte: WIKIPEDIA, 2011)

2.2.1 Codificação de Áudio

A codificação é importante, pois é necessária para representar as mídias de uma forma digital e para reduzir a quantidade de bits gerados pelas mídias.

> Um sinal digital carrega, em geral, muita informação redundante. Se eliminarmos essa redundância, conseguiremos reduzir em muito a quantidade de bits gerados. Em um sistema de TV digital, técnicas de compressão perceptualmente sem perdas são empregadas para o sinal de áudio, levando em conta o modelo psicoacústico humano. O resultado é um áudio de alta qualidade e com baixa taxa de bits gerada (BARBOSA e SOARES, 2008, p. 110).

O SBTVD adotou o padrão MPEG-4 para codificação de áudio, tendo em vista que este é considerado a melhor alternativa de alta qualidade em uma taxa de bits típica de 128 Kbps. Suas características são apresentadas no Quadro 1:

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

Quadro 1. Codificação de áudio no SBTVD (Fonte: BARBOSA e SOARES, 2008, p. 111)

2.2.2 Codificação de Vídeo

Semelhante ao áudio, as mídias de vídeo também representam grande redundância espacial e redundância temporal.

> No caso de um vídeo, se fosse possível congelá-lo momentaneamente, teríamos uma imagem parada (um quadro de vídeo). Se esse quadro fosse dividido em pequenos retângulos de tamanhos iguais, seria possível verificar que muitos dos retângulos são exatamente iguais a outros existentes na mesma imagem. Essa redundância pode ser eliminada através de técnicas que representam apenas a informação de um retângulo, e o número de vezes que estes precisam ser repetidos. Esse tipo de redundância, em um mesmo quadro de vídeo, é denominado redundância espacial (MONTEZ e BECKER, 2004, p. 17).

É explorando esse tipo de redundância que os algoritmos de codificação conseguem diminuir significativamente o tamanho das mídias. No caso das mídias de vídeo a maior redundância de informações se encontra na redundância espacial.

> A redundância temporal é um outro tipo de redundância que é muito explorado em dados de mídia contínua. Diferentemente da redundância espacial, que tem origem em informação duplicada em um mesmo quadro, a redundância temporal existe em informações

> em quadros contíguos. Como exemplo, no caso do PAL-M, 30 quadros de vídeos são apresentados por segundo, dando a sensação de um vídeo contínuo. Dividindo os quadros em pequenos retângulos, também seria significante o número de retângulos iguais em quadros consecutivos (em um telejornal, por exemplo, o cenário atrás do apresentador usualmente permanece inalterado) (MONTEZ e BECKER, 2004, p. 17).

Ainda são utilizadas outras técnicas de compressão. O mascaramento, por exemplo, é uma técnica utilizada para mascarar propriedades de vídeo ou áudio que não são perceptíveis a visão e audição humana.

O SBTVD adotou a técnica de codificação H.264 também conhecido como MPEG-4 para vídeo. O H.264/AVC emprega uma boa qualidade de imagem a uma taxa baixa e com implementação barata e eficiente. Suas características são apresentadas na Quadro 2:

	Receptores Fixos e Móveis	**Receptores Portáteis**
Padrão	ITU-T H.264 (MPEG-4 AVC)	ITU-T H.264 (MPEG-4 AVC)
Nível@Perfil	HP@L4.0	BP@L1.3
Número de linhas do nível	480 (4:3 e 16:9), 720 (16:9), 1080 (16:9)	SQVGA (160x120 ou 160x90), QVGA (320x240 ou 320x180) e CIF (352x288); todos em 4:3 e 16:9
Taxa de quadros	30 e 60 Hz	15 e 30 Hz

Quadro 2. Codificação de vídeo no SBTVD (Fonte: BARBOSA e SOARES, 2008, 112)

2.2.3 Sistema de Transporte

Os elementos gerados pela estação transmissora (áudio, vídeo e dados) para chegar as TVs Digitais devem ser adicionados em um único fluxo (multiplexação) e viajar através do enlace da fonte transmissora até o receptor. O conversor digital recebe esse fluxo e o separara novamente nos fluxos elementares de áudio, vídeo e dados (demultiplexação). O esquema de multiplexação e demultiplexação pode ser representado pela Figura 2:

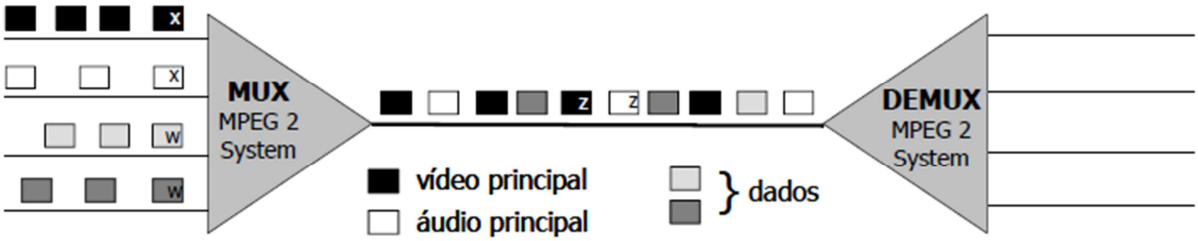

Figura 2. Transporte de dados (Fonte: BARBOSA e SOARES, 2008, p. 113)

O SBTVD adotou o mesmo padrão para transporte de dados dos sistemas americano, europeu e japonês, o MPEG-2 System.

Para fazer a sincronização dos fluxos de dados o MPEG-2 System utiliza o paradigma de eixo de tempo, caracterizado pela adição de carimbos de tempo (*timestamps*) a conjuntos de amostras codificadas de vídeo e áudio, baseadas em um relógio compartilhado.

> Cada fluxo elementar MPEG-2 System (áudio principal, vídeo principal, fluxo do carrossel de dados etc.) tem um identificador único. As especificações MPEG-2 System definem ainda o termo *programa,* chamado de *serviço* no contexto da TV digital, como um grupo composto de um ou mais fluxos elementares com uma mesma base temporal. O fluxo de transporte multiplexado pode conter vários serviços (programas) simultaneamente, cada um podendo ter uma base de tempo diferente (BARBOSA e SOARES, 2008, p. 115).

Resumindo, multiplexar significa organizar os dados de fluxos elementares, em um único fluxo identificando a qual serviço (programa) ele se refere para então transmitir o fluxo através do enlace, para que então seja feito o processo inverso na recepção.

2.2.4 Modulação

Para que o sinal de TV seja transmitido, ele tem que viajar da origem até o destino através de diversos tipos de enlace (cabo, ondas de rádio, satélite, etc.) os quais estão vulneráveis a interferências e ruídos. A modulação é necessária para que esses problemas não possam interferir na comunicação.

De acordo com Haykin (HAYKIN, 1999 apud MONTEZ e BECKER, 2004, p. 20) a modulação é o processo, através do qual alguma característica de uma onda portadora é alterada de acordo com o sinal da informação a ser transmitida.

Ou seja, o transmissor modifica as características da onda eletromagnética portadora, de forma que suas propriedades fiquem mais adequadas ao meio de transmissão. A onda portadora original é recuperada na recepção através de um processo reverso chamado demodulação.

O padrão de modulação utilizado no SBTVD é o BST-OFDM (*Band Segmented Orthogonal Frequency Division Multiplexing*). A OFDM é uma técnica de modulação baseada na idéia de multiplexação por divisão de frequência (FDM) onde

múltiplos sinais são enviados em diferentes freqüências. Com a introdução de segmentação de banda e intercalação do tempo deu origem a BST-OFDM aperfeiçoando ainda mais a técnica de modulação.

2.2.5 Canal de Retorno

O canal de retorno é de extrema relevância para a interatividade na TV digital. Não há nada que impeça que um sistema de TV trabalhe sem canal de retorno, porém a interatividade nesse caso será considerada como interatividade local, fornecida apenas pela aplicação.

O canal de retorno, segundo Montez e Becker (MONTEZ e BECKER, 2004), é o meio através do qual é possível a troca de informações no sentido inverso da difusão, ou seja, do telespectador para a emissora.

O canal de retorno utilizado no SBTVD é a internet. Ela dará a liberdade do usuário poder interagir com a estação transmissora de várias formas, permitindo o download ou upload de dados.

2.3 Middleware

Para Soares (BARBOSA e SOARES, 2008), o *middleware* é uma camada de software localizada entre as aplicações (programa de uso final) e o sistema operacional.

Uma das funções do *middleware* é fornecer suporte às aplicações. Tal suporte é fornecido por meio de interfaces de programação (API – Application Programming Interface), cuja funcionalidade oferecida depende das necessidades da aplicação.

O Ginga é o *middleware* criado e utilizado pelo SBTVD. Tem esse nome em referência ao movimento base da capoeira *a ginga*, e em reconhecimento à cultura, arte e contínua luta por liberdade e igualdade do povo brasileiro (GINGA, 2010). É resultado da soma de dois esforços, o ambiente não-declarativo (ou procedimental) baseado em Java, Ginga-J desenvolvido pela UFPB, e o ambiente declarativo Ginga-NCL, desenvolvido pela PUC-Rio, que utiliza a linguagem Nested Context Language (NCL), e sua linguagem de script Lua.

O Ginga é um software livre com especificações abertas e livre de *royalties*, permitindo qualquer desenvolvedor criar produtos interativos (GINGA, 2010).

2.3.1 Ambientes de Programação

Nos ambientes de programação para TV digital usam-se dois paradigmas de programação: declarativas e não-declarativos (BARBOSA e SOARES, 2008).

2.3.1.1 Middleware Declarativo

As linguagens declarativas têm como característica principal um alto nível de abstração e normalmente estão ligadas a um domínio ou objetivo específico. Neste modelo o programador fornece um conjunto de tarefas que deverão ser realizadas, não se preocupando em como o compilador ou interpretador irá realizá-la. Por se tratar de linguagens com alto nível de abstração, o paradigma declarativo torna-se limitado para determinadas aplicações (BARBOSA e SOARES, 2008).

Para Soares (BARBOSA e SOARES, 2008) Ginga-NCL é o subsistema lógico do *middleware* Ginga responsável pelo processamento de aplicações com base na linguagem NCL.

A linguagem NCL (*Nested Context Language*) é uma linguagem declarativa que mantém os objetos semanticamente unidos em uma apresentação de multimídia.

Segundo Barbosa e Soares (BARBOSA e SOARES, 2008) um aplicativo NCL apenas define como os objetos de mídia são estruturados e relacionados no tempo e espaço. Ou seja, funciona como uma linguagem de “cola”.

Para suprir a limitação do paradigma declarativo, a linguagem NCL dá suporte à outra linguagem de script, chamada Lua, que combina a sintaxe procedural com a declarativa.

A linguagem Lua, escrita em C, é simples, leve, robusta, embarcada e de código aberto. Por sua natureza extensível, trabalha embarcada em uma linguagem principal, trazendo rotinas procedurais para o programa principal escrito em NCL.

De acordo com Zancanaro (ZANCANARO et. al., 2009) aplicações construídas em NCL em conjunto com a Lua, aumentam muito o potencial dos

programas para TV digital, levando a um maior grau de imersão e interatividade a serem disponibilizadas para o usuário.

2.3.1.2 Middleware Não-Declarativo

Nas linguagens não-declarativas o programador tem mais poder sobre o código, informando cada passo a ser executado. Para o desenvolvimento de aplicações nesse tipo de paradigma é exigido certa experiência do programador. Entre tantas linguagens existentes, o Java é a linguagem mais usada para desenvolver aplicações não-declarativas para TV digital (BARBOSA e SOARES, 2008).

O Ginga-J é o subsistema lógico do *middleware* Ginga responsável pelo processamento de aplicações imperativas utilizando a linguagem Java. Ele dá suporte as funcionalidades necessárias para o desenvolvimento de aplicações de TV digital através de interfaces de programação (API).

De acordo com Barbosa e Soares (BARBOSA e SOARES, 2008) as APIs são divididas em três módulos:

- APIs verdes: responsáveis por manter o máximo possível de compatibilidade com os sistemas: europeu e americano.
- APIs amarelas: oferecem suporte aos múltiplos usuários, a múltiplos dispositivos e a múltiplas redes. Também oferecem suporte para aplicações que poderão ser recebidas, armazenadas e executadas no futuro. Esta API inclui o JMF (Java Media Framework) que é usada para desenvolver aplicações avançadas como a captura de áudio e vídeo.
- APIs vermelhas: dão suporte às aplicações voltadas para o Brasil, especialmente para a interatividade, promovendo a inclusão social. Permitem também a integração do conteúdo declarativo e procedural na mesma aplicação.

A linguagem Java tem como principal objetivo o desenvolvimento de aplicações com alto nível de interatividade, gráficos de qualidade e poder de processamento.

2.4 Linguagens de Programação

Programas de TV digital interativa podem ser entendidos como aplicações hipermídia/multimídia, pois são formadas por conteúdo de mídia e ligados por links. Nesse cenário, sistemas hipermídia irão constituir uma das ferramentas mais importantes a serem dominadas. Sistemas de autoria hipermídia dão o suporte para a geração de informação, não se restringindo apenas à concepção dos conteúdos em si, mas incluindo também a concepção de como eles devem ser apresentados. Sistemas de exibição hipermídia são os responsáveis pela apresentação especificada. Todos esses sistemas têm por base alguma linguagem de especificação. (CARVALHO et. al., 2009)

No Ginga-NCL a linguagem de programação utilizada para prover a apresentação dos componentes de mídia é a NCL, *Nested Context Language*. Para suprir a limitação do paradigma declarativo, o Ginga-NCL dá suporte a uma linguagem de script, chamada Lua, que combina o paradigma procedural com o declarativo.

2.4.1 Linguagem NCL

A NCL (*Nested Context Language*) é uma linguagem declarativa, uma aplicação XML, baseada no modelo NCM (*Nested Context Model*). A NCL traz uma separação clara entre os conteúdos de mídia e a estrutura de uma aplicação. (BARBOSA e SOARES, 2009)

Um documento hipermídiia, de forma genérica, é composto por nós e elos. Os nós representam abstrações das mídias utilizadas no documento além de trazerem informações adicionais, como por exemplo, informações sobre a sua apresentação. Os elos fazem a sincronização espacial ou temporal entre os nós que compõem o documento.

Segundo Carvalho (CARVALHO et. al., 2009) na construção de um documento hipermídia são necessárias algumas informações básicas, indicando o que deve ser apresentado, como, quando e onde devem ser apresentados.

- **Onde**: Para que um nó de mídia seja apresentado, é necessário definir uma área para exibição. No modelo NCM são definidas regiões para este fim. Indicando a posição e o tamanho de onde os nós poderão ser apresentados.
- **Como**: Para complementar as características de um nó é necessário criar descritores, que podem descrever parâmetros, incluindo a região onde será apresentado, seu volume, sua transparência, a duração de sua apresentação, entre outros.
- **O que**: Um conteúdo de um documento hipermídia é representado através de elementos denominados mídia. Uma mídia representa cada nó de um documento, informando o descritor ao qual está relacionado.
- **Quando**: Após definir os nós que farão parte do documento hipermídia, é necessário definir qual será o primeiro nó a ser exibido e qual será a ordem de apresentação dos demais nós. Essas definições são feitas através do uso de portas e links. As portas são utilizadas para definir um nó inicial e os links são utilizados para relacionamento e sincronização entre outros nós. Um link, entretanto, não define todo o comportamento de um relacionamento por si só, para isso é necessário o uso de conectores.

2.4.1.1 Estrutura de Aplicações NCL

Um arquivo XML é organizado de forma hierárquica onde cada elemento possui um elemento pai e elementos filhos. Um elemento é iniciado pelo símbolo “<” e terminado pelos símbolos “/>”. Entre esses dois símbolos são definidos o nome do elemento e seus atributos, conforme Algoritmo 1.

```
<aluno id=1 nome=”Victor L. Oliveira” instituição=”UPE” />
```

Algoritmo 1. Definição de um elemento

Para definir elementos filhos o elemento não deve ser terminado por “/>”, mas pela repetição do nome do elemento entre os símbolos “</” e “>”.

```
<aluno id=1 nome=”Victor L. Oliveira” instituição=”UPE”
     <disciplina codigo=”123” nome=”Desenvolvimento de Software”
/>
     <disciplina codigo=”234” nome=”Sistemas Cooperativos” />
</aluno>
```

Algoritmo 2. Definição de elementos filhos

Como toda aplicação XML, uma aplicação NCL deve começar exibindo um cabeçalho XML na primeira linha do arquivo.

Todo conteúdo de um documento NCL deve ser escrito dentro do elemento <ncl>. Sua estrutura básica é composta pelo cabeçalho definido pelo elemento <head> e pelo corpo definido pelo elemento <body>.

No cabeçalho ficam contidos as características de apresentação do documento, como as regiões, os descritores, as transições, os conectores e as regras.

No corpo ficam contidos os elementos que definem o conteúdo da aplicação propriamente dita, tal como objetos de mídia, elos, contextos e objetos switch.

O Algoritmo 3 define um exemplo de código com a estrutura básica de um documento NCL:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="main" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
     <head>
     </head>
     <body>
     </body>
</ncl>
```

Algoritmo 3. Estrutura Básica de um documento NCL

2.4.1.2 Regiões

Uma região representa o local onde um objeto de mídia será apresentado. Para que um documento seja apresentado é necessário que no mínimo uma região seja definida. Para isso, dentro do cabeçalho, criamos uma base de regiões definida pelo elemento <regionBase> e uma região específica, definida pelo elemento <region>.

Toda região possui um identificador único representado pelo atributo id, ela podem possuir outros atributos não obrigatórios que são definidos no Quadro 3:

id	Identificador único, utilizado nas referências as regiões
left	Coordenada x do lado esquerdo da região, com relação à coordenada do lado esquerdo da região pai. (Ou área total caso a região não esteja associada a nenhuma outra)
top	Coordenada y do lado superior da região, com relação à coordenada do lado superior da região pai. (Ou área total caso a região não esteja associada a nenhuma outra)
right	Coordenada x do lado direito da região, com relação à coordenada do lado direito da região pai. (Ou área total caso a região não esteja associada a nenhuma outra)
bottom	Coordenada y do lado inferior da região, com relação à coordenada do lado inferior da região pai. (Ou área total caso a região não esteja associada a nenhuma outra)
width	Dimensão horizontal da região
height	Dimensão vertical da região
zIndex	Define a sobreposição das camadas. De acordo com o valor contido neste atributo, uma região será apresentada acima de outras regiões com zIndex menor e abaixo de outras regiões com zIndex maior. Caso os valores de duas regiões sejam iguais, a mídia apresenta por último ficará acima da anterior.
title	Título da região, cujo uso depende da implementação do formatador.

Quadro 3. Atributos de uma região

No Algoritmo 4 é definido um exemplo de código com a criação de regiões:

```
<regionBase>
     <region id=”rgVideo” width="100%" height="100%" zIndex="0">
          <region id="rgImagem" width="20%" height="20%"
zIndex="3"/>
     </region>
</regionBase>
```

Algoritmo 4. Definição de regiões de Vídeo e Imagem

2.4.1.3 Descritores

Os descritores especificam como os objetos de mídia a eles associados serão exibidos. Assim como as regiões, são definidos no cabeçalho dentro de uma base de descritores definida pelo elemento <descriptorBase> e utilizando o elemento <descriptor>. Porém, diferente das regiões, um descritor não pode estar aninhado a outro descritror.

Um descritor pode possuir vários atributos, alguns são definidos no Quadro 4:

id	Identificador único, utilizado nas referências as descritor
region	Identificador da região associada ao descritor. Todo objeto que utilize esse descritor será inicialmente exibida nessa região.
explicitDur	Define a duração do objeto de mídia associado ao descritor. O valor desse atributo pode ser expresso em segundos, no formato “9s”. Também pode ser expresso como “horas:minutos:segundos:fração”. Quando o atributo explicitDur não for especificado, será levada em consideração a duração padrão da mídia.
freeze	Identifica o que acontece ao final da apresentação do objeto de mídia associado ao descritor. Em um vídeo, o valor “true” indica que, ao término natural do vídeo, o último quadro deve ser congelado indefinidamente, até que algum elo termine sua exibição.
focusIndex	No início da execução do documento de navegação, o foco é passado para o elemento associado ao descritor de menor índice. Se um descritor não definir um índice, o elemento associado não receberá foco na navegação
moveLeft	Define o descritor, através do índice, que receberá o foco quando a o botão “seta para esquerda” for pressionado.
moveRight	Define o descritor, através do índice, que receberá o foco quando a o botão “seta para direita” for pressionado.
moveUp	Define o descritor, através do índice, que receberá o foco quando a o botão “seta para cima” for pressionado.
moveDown	Define o descritor, através do índice, que receberá o foco quando a o botão “seta para baixo” for pressionado.

Quadro 4. Atributos de um descritor

No Algoritmo 5 é definido um exemplo de código com a criação de descritores:

```
<descriptorBase>
     <descriptor id=”dVideo” region=”rgVideo”/>
     <descriptor id=”dImagem” region=”rgImagem” />
</descriptorBase>
```

Algoritmo 5. Definição dos descritores de Vídeo e Imagem

2.4.1.4 Mídias, Âncoras e Propriedades

Uma mídia é a representação do objeto que será apresentado pelo documento, é definido pelo elemento <media> dentro do seu corpo.

Um objeto de mídia possui os atributos definidos no Quadro 5:

id	Identificador único, utilizado nas referências as mídias
src	Fonte do objeto de mídia, ou seja, a localização ou caminho do objeto de mídia
type	Atributo opcional que define o tipo de mídia
descriptor	Identificador do descritor que controla a apresentação do objeto de mídia
refer	Referência a um outro nó de mídia previamente definido, como forma de reuso de nó
instance	Utilizado apenas quando o atributo refer é definido. Pode assumir os valores “new”, “instSame” e “gradSame”

Quadro 5. Atributos de um objeto de mídia.

Algumas vezes pode ser necessário referenciar uma parte de um conteúdo de mídia, para isso são utilizadas âncoras.

Uma ancora serve para referenciar parte de um conteúdo de mídia, no exemplo de um vídeo, as âncoras poderiam ser trechos desse vídeo. São utilizadas para sincronizar um objeto de mídia com outros (músicas, vídeos, textos, etc) e é definida pelo elemento <area>.

Uma ancora possui os atributos definidos no Quadro 6:

id	Identificador único da âncora
begin	Início da âncora de uma mídia contínua. Pode ser definido em segundos no formato “2s” ou no formato Hora“:”Minuto“:”Segundo“.”Fração, onde Hora é um inteiro no intervalo [0,23], Minuto é um inteiro no intervalo [0,59], Segundo é um inteiro no intervalo [0,59] e Fração é um inteiro positivo
end	Fim da âncora de uma mídia contínua. Pode ser definida da mesma forma de Begin
text	Texto da âncora no arquivo de origem (atributo válido apenas para mídias de texto)
label	Identificador da âncora no arquivo de origem, seguindo a interpretação dada pela ferramenta de exibição

Quadro 6. Atributos de uma âncora.

Além das ancoras, os objetos de mídia, podem definir propriedades que poderão ser manipuladas pelos elos. Por exemplo, volume de áudio de um objeto de áudio, coordenadas e dimensões de exibição de um objeto de mídia visual, grau de transparência, etc.

Uma propriedade define os atributos do Quadro 7:

name	Nome da propriedade ou grupo de propriedades
value	Valor inicial atribuído à propriedade ou grupo de propriedades

Quadro 7. Atributos de uma propriedade.

No Algoritmo 6 é definido um exemplo de código para criação mídias e âncoras:

```
<media id="video" src="media/video.avi" descriptor="dVideo">
                <area id="area01" begin="3s" end="6s"/>
                <area id="area02" begin="10s" end="13s"/>
                <area id="area03" begin="17s" end="20s"/>
                <area id="area04" begin="24s" end="27s"/>
</media>
```

Algoritmo 6. Definição de uma mídia de vídeo e suas ancoras.

2.4.1.5 Portas, Conectores e Elos

Uma porta serve para definir um nó inicial para a apresentação do documento, no seu corpo deve estar contido pelo menos uma porta.

Uma porta contém os atributos definidos no Quadro 8:

id	Identificador único da porta
component	Nó componente sendo mapeado pela porta
interface	Interface do nó sendo mapeado. Podem ser âncoras ou propriedades, caso o elemento mapeado seja uma mídia.

Quadro 8. Atributos de uma porta.

No Algoritmo 7 é definido um exemplo de código para a criação de uma porta:

```
<port component="video" id="inicio"/>
```

Algoritmo 7. Definição de uma porta.

Para definir o sincronismo e, em particular, a interatividade entre os objetos de uma aplicação NCL existem os conectores e elos.

Os conectores definem relações genéricas que serão utilizadas pelos elementos de um documento NCL. Durante essa definição não são indicados os participantes de um relacionamento específico. Imagine, por exemplo, a relação

"ensina à". Esta relação define que alguém ensina a alguém, porém não define quem ensina e quem aprende, ou seja, os participantes.

Um conector é definido em uma base de conectores <connectorBase> pelo elemento <causalConnector> dentro do cabeçalho do documento. Este conector basicamente definirá uma relação de causa e efeito.

A causa será definida pelo elemento <simpleCondition> com uma condição que deverá ser satisfeita para que o conector seja ativado. Caso haja mais de uma condição a ser satisfeita, será usado o elemento <compoundCondition> que obrigatoriamente deverá definir o atributo operator que recebe os valores “and” ou “or”, indicando se todas ou pelo menos uma condição deve ser satisfeita para que o conector seja ativado.

Tanto uma causa simples como uma composta, será definida através do atributo role. NCL possui um conjunto de nomes reservados para papéis de condição. Os nomes e seus significados são listados no Quadro 9:

onBegin	É ativado quando a apresentação do elemento ligado a esse papel é iniciada
onEnd	É ativado quando a apresentação do elemento ligado a esse papel é terminada
onAbort	É ativado quando a apresentação do elemento ligado a esse papel for abortada
onPause	É ativado quando a apresentação do elemento ligado a esse papel for pausada
onResume	É ativado quando a apresentação do elemento ligado a esse papel for retornada após uma pausa
onSelection	É ativado quando uma tecla (a ser especificada) for pressionada enquanto o elemento ligado a esse papel estiver sendo apresentado ou qualquer tecla ENTER for pressionada enquanto o elemento ligado a esse papel estiver com o foco
onBeginAttribution	É ativado logo antes que um valor (a ser especificado) seja atribuído a uma propriedade do elemento ligado a esse papel
onEndAttribution	É ativado logo após um valor (a ser especificado) ter atribuído a uma propriedade do elemento ligado a esse papel

Quadro 9. Papéis predefinidos de condição.

Quando a opção onSelection é utilizada é necessário definir o atributo key, que fará referência a tecla pressionada no controle para que o conector seja ativado. Pode receber os valores: "0" a "9", "A" a "Z", "*", "#", "MENU", "INFO", "GUIDE",

"CURSOR_DOWN", "CURSOR_LEFT", "CURSOR_RIGHT", "CURSOR_UP", "CHANNEL_DOWN", "CHANNEL_UP", "VOLUME_DOWN", "VOLUME_UP", "ENTER", "RED", "GREEN", "YELLOW", "BLUE", "BACK", "EXIT", "POWER", "REWIND", "STOP", "EJECT", "PLAY", "RECORD" e "PAUSE"

Quando a condição for satisfeita o elemento <simpleAction> será ativado para definir uma ação a ser executada. Caso seja necessário realizar mais de uma ação o elemento a ser definido será o <compoundAction> que receberá outras ações simples como filhas. Quando utilizado, este elemento deve definir o atributo operator que recebe os valores “par” ou “seq”, indicando se as ações devem ser executadas em paralelo ou sequencialmente.

O elemento <simplesAction> define através do atributo role o nome do papel de ação. NCL também possui um conjunto de nomes reservados para papéis de ação. Os nomes e seus significados são listados no Quadro 10:

start	Inicia a apresentação do elemento ligado a esse papel
stop	Termina a apresentação do elemento ligado a esse papel
abort	Aborta a apresentação do elemento ligado a esse papel
pause	Pausa a apresentação do elemento ligado a esse papel
resume	Retoma a apresentação do elemento ligado a esse papel
set	Estabelece um valor a uma propriedade de um elemento associado a esse papel

Quadro 10. Papéis predefinidos de ação.

No Algoritmo 8 é definido um exemplo de código para a criação de conectores:

```
<connectorBase>
     <causalConnector id="onBeginStart">
          <simpleCondition role="onBegin"/>
          <simpleAction role="start"/>
     </causalConnector>
</connectorBase>
```

Algoritmo 8. Definição de conectores.

Após definir as relações, é necessário identificar os participantes da mesma. Seguindo o exemplo de relação "ensina à", um elo que utilizasse essa relação identificaria os elementos "professor" e "aluno". O relacionamento completo, especificado pelo elo, ficaria então "o professor ensina ao aluno".

Um elo é definido pelo elemento <link> que através do atributo xconnector irá fazer referenciar a relação utilizada, no caso, algum conector.

Para a criação de ligações entre os elementos e os papéis, um elo define elementos filhos <bind>. Um <bind> possui os atributos definidos no Quadro 11:

role	Identifica o papel sendo utilizado
component	Identifica, através do id, o elemento participando da relação
interface	Identifica uma âncora ou propriedade do elemento, caso este seja uma mídia, ou uma porta de um elemento, caso este seja uma composição, através do seu id.

Quadro 11. Atributos de um bind.

No Algoritmo 9 é definido um exemplo de código para a criação de elos:

```
<link xconnector="onBeginStart">
        <bind role="onBegin" component="video" interface="area01"/>
        <bind role="start" component="imagem"/>
</link>
```

Algoritmo 9. Definição de elos.

Para a passagem de parâmetros entre conectores e elos, são utilizados os elementos <connectorParam> e <bindParam> respectivamente. Estes elementos definem os atributos name e value, onde o primeiro identifica o parâmetro e o segundo o seu valor.

Mais informações sobre NCL podem ser encontradas na obra de Soares e Barbosa, Programando em NCL 3.0: Desenvolvimento de aplicações para o Middleware Ginga, TV digital e Web (SOARES e BARBOSA, 2009).

No apêndice A pode ser encontrado o código completo de uma aplicação que reproduz um vídeo e a cada 10 segundos exibe uma imagem na tela por 5 segundos.

2.4.2 Linguagem Lua

Lua é uma linguagem de programação poderosa, rápida e leve, projetada para estender aplicações (LUA, 2011).

Criada em 1993 no laboratório Tecgraf da PUC-Rio a linguagem Lua, a princípio, fazia parte de um Projeto da Petrobras, porém, devido a sua eficiência, clareza e facilidade de aprendizado, passou a ser usada em diversos ramos da programação, como no desenvolvimento de jogos (por exemplo, Angry Birds, Tibia, etc).

Lua combina programação procedural com poderosas construções para descrição de dados, baseadas em tabelas associativas e semântica extensível. É tipada dinamicamente, interpretada a partir de bytecodes, e tem gerenciamento automático de memória com coleta de lixo.

2.4.2.1 Convenções Léxicas

Na linguagem Lua, os identificadores podem ser uma cadeia de letras, dígitos e sublinhado, porém não podem começar com um dígito. Os identificadores são utilizados para nomear variáveis, funções e campos de tabelas.

O Quadro 12 mostra algumas palavras reservadas que não podem ser utilizadas como identificadores:

and	break	do	else	elseif	End	false
function	if	in	local	nil	Not	or
return	then	true	until	while	For	repeat

Quadro 12. Palavras reservadas da linguagem Lua.

Lua é *case sensitive*, ou seja, diferencia letras maiúsculas de minúsculas.

Uma string é delimitada por aspas duplas ou aspas simples e possui a mesma sequência de escape da linguagem C, representada no Quadro 13:

‘\a’	Campainha
‘\b’	Backspace
‘\f’	Alimentação de formulário
‘\n’	Quebra de linha
‘\r’	Retorno de carro
‘\t’	Tabulação horizontal
‘\v’	Tabulação vertical
‘\\’	Barra invertida
‘\”’	Aspa dupla
‘\’’	Aspa simples

Quadro 13. Caracteres de escape.

2.4.2.2 Tipos e Variáveis

Lua é uma linguagem de tipagem dinâmica, ou seja, não é necessário declarar o tipo da variável, pois, seu tipo é automaticamente determinado dependendo do valor armazenando.

Existem oito tipos básicos em Lua:

- **nil**: é utilizado para indicar a ausência de um valor. Se uma variável for declarada e nenhum valor for atribuído seu valor será nil.
- **boolean:** representa os valores booleanos true e false, se uma variável for considerada igual a nil ou false será considerada falsa, caso contrário será verdadeira.
- **number:** representa o único tipo numérico em Lua, é ponto flutuante por padrão.
- **string:** representa uma cadeia de caracteres, que são delimitadas por aspas simples ou aspas dupla. Para se escrever uma cadeia que se estende por várias linhas deve-se utilizar [[para abrir cadeia e]] para fechar cadeia.
- **userdata:** é usado para armazenar dados C em variáveis Lua. Valores desse tipo só podem ser criados ou modificados através da API de Lua com C.
- **function:** tipo que representa funções.
- **thread:** representam fluxos de execução independentes e dá suporte ao uso de co-rotinas.
- **table:** representa uma tabela em Lua. Tabelas é o único mecanismo de estruturação de dados em Lua. Com tabelas pode-se representar vetores, tabelas de símbolos, conjuntos, grafos, registros, etc.

Para declarar uma variável em Lua basta escrever seu identificador.

Para atribuir um valor a uma variável, utiliza-se o operador de atribuição =. Numa operação de atribuição é possível efetuar a atribuição de mais de um valor para mais de uma variável em uma única linha de comando, conforme o Algoritmo 10:

```
saudacao, nome = “oi”, “Victor”
print (saudacao, nome)
-- Saida: oi Victor
```

Algoritmo 10. Declaração e atribuição de variáveis.

Em Lua para realizar comentários é utilizado -- seguido do comentário.

2.4.2.3 Operadores

Em Lua são utilizados três tipos de operadores: aritméticos, relacionais e lógicos.

Os operadores aritméticos mais utilizados em são listados no Quadro 14:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
^	Exponenciação
-	Negação

Quadro 14. Principais operadores aritméticos.

Lua possui os operadores relacionais listados no Quadro 15:

<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a
==	Igual a
~=	Diferente

Quadro 15. Principais operadores relacionais.

Os operadores relacionais sempre retornam true ou false.

Os operadores lógicos da linguagem Lua são listados no Quadro 16:

and	Retorna o primeiro argumento se ele for falso, caso contrário retorna o segundo argumento.
or	Retorna o primeiro argumento se ele não for falso, caso contrário retorna o segundo argumento.
not	Retorna true ou false.

Quadro 16. Principais operadores lógicos.

Lua ainda permite a concatenação de strings e para isso disponibiliza o operador .. (dois pontos seguidos). Se na concatenação um dos operandos for um número, este será convertido para string

No Algoritmo 11 é mostrado um exemplo de código em Lua com a utilização de operadores:

```
-- Exemplo operadores aritméticos
a = 3*2
print(a) -- Saída: 6
b = 3^(2)
print(b) -- Saída: 9

-- Exemplo operadores relacionais
if (b>a) then
        print(b) -- Saída: 9
end

-- Exemplo operadores lógicos
print (2 and 3) -- Saída: 3
print (false or 3) -- Saída: 3
```

Algoritmo 11. Utilização de operadores.

2.4.2.4 Estruturas de Controle

Em Lua pode-se utilizar as estruturas de controle if e/ou else para condicionais e while, repeat e for para interação.

Todo bloco de código de uma estrutura deve ter um terminador explícito. Para if, while e for o terminador é o end, para o repeat o terminador é until.

if then else	O if testa uma condição, se for verdadeira a parte then é executada senão else é executada. A parte else é opcional. Em Lua não existe a estrutura case, então para se construir uma estrutura de ifs aninhadas pode-se utilizar elseif.
while	Uma das estruturas de repetição de Lua. Primeiramente a condição é testada se for falsa não entra no loop. Se for verdadeira, executa o bloco em loop avaliando novamente a condição a cada ciclo.
repeat-until	Repete o que está no seu bloco de código até a condição testada pelo until ser verdadeira
for	É uma estrutura de repetição que tem um valor incial, uma condição de repetição e o passo do laço (valor incrementado), quando não é especificada assume-se que seu valor é 1. É semelhante ao for utilizado em outras linguagens de programação como C ou Java.

Quadro 17. Estruturas de controle.

No Algoritmo 12 é mostrado um exemplo de código em Lua com a utilização de estruturas de repetição:

```
if a<6 then  --Exemplo if-then-else
        return a      -- Caso condição verdadeira retorna valor de a
else
        return b      -- Caso condição verdadeira retorna valor de b
end

local i = 1 -- Exemplo while
while a[i] do      -- Testa condição, se verdadeira executa o código e testa novamente.
        print(a[i])
        i = i + 1
end

local j = 0 -- Exemplo repeat
repeat
         print(i)
         i = i+1
until i ~= 5
```

Algoritmo 12. Utilização de estruturas de controle.

Para sair de um bloco de programa pode-se utilizar os comando break e return. O break é muito utilizado em estruturas de repetição, pois, termina a execução da estrutura.

Com o comando return é possível retornar valores de uma função ou simplesmente terminar a execução de um bloco.

2.4.2.5 Funções

Entende-se por função, uma subrotina ou porção de código para resolver um problema específico, parte de um problema maior.

Em Lua uma função pode retornar um valor, ou não. No Algoritmo 13 é mostrado um exemplo de como declarar uma função em Lua:

```
-- Declarando a função soma
function soma(a,b)
	return a+b
end

-- Chamando a função soma e mostrando resultado
resultado = soma(5,6)
print(resultado) -- Saída: 11
```

Algoritmo 13. Declaração e utilização de funções.

Na declaração de uma função podem ser passados vários parâmetros ou nenhum. Caso na utilização da função sejam passados menos parâmetros do que o número utilizado na declaração, os parâmetros que faltarem serão fornecidos com o valor nil. Caso forem passados mais parâmetros do que a função utiliza, os valores extras serão descartados.

Lua também permite a declaração de funções com número variável de parâmetros, deve-se utilizar a notação ..., para que uma função acesse o número de parâmentros passados. No Algoritmo 14 é demonstrado como se utilizar uma função com o número de parâmetros variável:

```
-- Declarando a função
function maior_que_1000(...)
	for i, t in ipairs {...} do -- Percorre o vetor de parâmetros
		if t > 1000 then -- Se for maior que 1000 imprime o valor
			print (t)
		end
	end
end

-- Utilizando a função
maior_que_1000(45, 670, 2800, 10203) -- Imprime 2800 e 10203
```

Algoritmo 14. Declaração e utilização de funções com número variável de parâmetros.

2.4.2.6 Tabelas

Em Lua as tabelas são a única forma de estrutura de dados, implementando diversas estruturas, como vetores, matrizes, grafos, etc.

Tabelas são criadas utilizando o construtor {}. Podemos criá-las vazias ou com valores já pré-definidos, onde uma tabela poderá conter outras tabelas.

```
-- Declarando uma tabela vazia
tabela = {}

-- Declarando uma tabela com valores pré definidos
cursos = {engenharia, sistemas_info, medicina, direito}
```

Algoritmo 15. Declaração de tabelas.

Para inserir valores dentro de uma tabela podemos utilizar a função table.insert, e se desejarmos saber o tamanho da tabela utilizamos o operador # semelhante ao array.length utilizado em Java.

Caso se deseje utilizar uma tabela como vetor, basta declarar os valores informando também o valor do seu índice.

A seguir é mostrado um exemplo de código que mostra o do tamanho de uma tabela e insere um novo elemento.

```
-- Declarando uma tabela e imprimindo o seu tamanho
tabela = {“Maria”, “Joao””, “Jose”, “Adamastor”}
print(#tabela) -- Saída: 4

-- Inserindo novo elemento na tabela e imprimindo o seu tamanho
table.insert(tabela,"Josefa")
print(#tabela) -- Saída: 5
```

Algoritmo 16. Mostrando tamanho de tabelas e inserindo novos elementos

Mais informações sobre a linguagem Lua podem ser encontradas no manual de referência disponível em: http://www.lua.org/manual/5.1/pt/ (IERUSALIMSCHY et. al., 2006).

2.4.3 Integração NCL-Lua

Linguagens declarativas como NCL, apresentam uma maior facilidade de aprendizagem, no entanto, pecam pela falta de controle das informações processadas, o que não acontece em uma linguagem imperativa como Lua.

O poder de uma linguagem declarativa é elevado quando integrada com uma linguagem imperativa. Essa integração deve seguir critérios que não afetem os princípios da linguagem declarativa, mantendo uma separação clara entre os dois ambientes. Para garantir esses critérios, um script Lua deve ser escrito

separadamente do documento NCL que por sua vez irá relacioná-lo como um objeto de mídia qualquer. Sendo assim, o documento NCL é que controlará o ciclo de vida do script Lua.

O modelo de execução de um NCLua é orientado a eventos, ou seja, o fluxo da aplicação é guiado por eventos externos oriundos do documento NCL. O acionamento desses eventos hora é feito pelo formatador NCL, hora é feito pelo NCLua, que irá comunicar ao formatador a ocorrência de eventos internos, criando assim um canal bi-direcional de comunicação.

Além do conjunto padrão de bibliotecas, quatro novos módulos foram desenvolvidos para facilitar a utilização de NCLua:

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

2.4.3.1 Módulo event

Dentre os módulos que foram introduzidos no ambiente NCLua o modulo event é o mais importante, pois é responsável por realizar a ponte de comunicação entre o documento NCL e o script NCLua.

Para que o NCLua receba um evento do formatador, é necessário que o script registre o tratamento para o evento. A partir daí, caso o evento seja recebido, a ação determinada pela função responsável por tratar o evento é realizada.

Para registrar uma função tratadora de eventos deve fazer uma chamada ao envent.register(evt), conforme Algoritmo 17:

```
function handler (evt)
        -- Codigo para tratar o evento
end
event.register(hander)
```

Altorimo 17. Registrando o tratamento de um evento.

Um NCLua além de receber eventos, também pode enviar eventos para o ambiente NCL. Para isso utiliza-se a função event.post(evt).

Os eventos são tabelas simples cujo campo class identifica a classe do evento. As seguintes classes são definidas:

- **ncl:** Trata dos eventos relacionados à ponte entre o NCL e o NCLua.
- **key:** Acesso ao controle remoto por parte do usuário.
- **tcp:** Uso da conexão de acesso a internet.
- **sms:** Envio e recebimento de mensagens de texto.
- **user:** Permite a criação de eventos customizados.

Para a classe ncl, os seguintes campos também estão presentes:

- **type:** Assume presentation, para âncoras de conteúdo (áreas) ou attribution, para âncoras de propriedade.
- **action:** Pode assumir um dos valores das transições da máquina de estado: start, stop, pause, resume, abort
- **area ou property:** Nome da área ou propriedade, dependendo do valor em type.
- **value:** Caso type seja attribution, assume o valor da propriedade no campo property.

No Algoritmo 18 é mostrado um exemplo de código que trata o evento do início do script NCLua e assim que iniciado sinaliza o seu fim:

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

Algoritmo 18. Tratando código do evento de início do script NCLua.

Para a classe de eventos key, utilizamos os seguintes campos:

- **type:** Deve ser do tipo press ou release.
- **key:** Valor da tecla em questão.

Abaixo são definidas as funções do módulo event:

- **event.register (pos, f, class):** registra a função passada como tratador de eventos. Parâmetros:
  - ✓ pos: É opcional e indica a posição em que f é registrada. Caso não seja passado, a função registrada será a última a receber eventos.
  - ✓ f: Representa a função de tratamento de eventos.
  - ✓ class: É opcional e indica que classe de eventos a função deve receber.
- **event.unregister (f):** Cancela o registro da função passada como um tratador de eventos. Parâmetros:
  - ✓ f: Representa a função de tratamento de eventos.
- **event.post (dst, evt):** Gera um novo evento. Parâmetros:
  - ✓ dst: É o destino do evento. Pode assumir os valores 'in' (envio para a própria aplicação) e 'out' (envio para o formatador). Se não for passado, assume o valor 'out' por default.
  - ✓ evt: Evento a ser passado.
- **event.timer (time, f):** Cria um timer em milissegundos e, ao fim, chama a função f. Parâmetros:
  - ✓ time: Tempo em milissegundos.
  - ✓ f: Função a ser executada.

- **event.uptimer (time):** Retorna o número de milissegundos decorridos desde o início da aplicação. Parâmetros:
  - ✓ time: Tempo em milissegundos.

2.4.3.2 Módulo canvas

Um NCLua pode ter a necessidade de desenhar objetos gráficos na tela. Para isso o módulo canvas oferece uma API gráfica trazendo a possibilidade de utilizar imagens, linhas, círculos, textos e etc.

A região que serão utilizada para o desenho dos objetos gráficos, será a mesma região designada no documento NCL lua para representar o script NCLua.

Abaixo são definidas as funções do módulo canvas:

- **canvas:new(width, height):** Instancia um canvas com tamanho definido. Parâmetros:
  - ✓ width: Largura do canvas.
  - ✓ height: Altura do canvas.
- **canvas:new(image_path):** instancia um canvas cujo conteúdo é a imagem passada como parâmetros. Parâmetros:
  - ✓ image_path: Caminho da imagem.
- **canvas:attrSize():** Retorna as dimensões do canvas.
- **canvas:attrColor(color_name, A).** Altera a cor do canvas. Parâmetros:
  - ✓ color_name: pode assumir os valores: ‘white’, ‘aqua’, ‘lime’, ‘yellow’, ‘red’, ‘fuchsia’, ‘purple’, ‘maroon’, ‘blue’, ‘navy’, ‘teal’, ‘green’, ‘olive’, ‘silver’, ‘gray’, e ‘black’.
  - ✓ A: valor da transparência, varia de 0 (totalmente transparente) à 255 (totalmente opaco).
- **canvas: attrColor():** Retorna as cores do canvas.
- **canvas:attrClip(x, y, width, height):** Limita a area do canvas para desenho. O valor inicial é o canvas inteiro. Parâmetros:
  - ✓ x: Coordenada x da área limitada.
  - ✓ y: Coordenada y da área limitada.

- ✓ width: Largura da área limitada.
- ✓ height: Altura da área limitada.

- **canvas:attrClip():** Retorna as dimensões da área limitada.
- **canvas:attrCrop(x, y, width, height):** Atributo de corte do canvas. Parâmetros:
  - ✓ x: Coordenada x da área limitada.
  - ✓ y: Coordenada y da área limitada.
  - ✓ width: Largura da área limitada.
  - ✓ height: Altura da área limitada.
- **canvas:attrCrop():** Retorna as dimensões de corte do canvas.
- **canvas:attrFont(face, size, style):** Altera a fonte do canvas. Parâmetros:
  - ✓ face: Nome da fonte. As fontes ‘Tiresias’ e ‘Verdana’ devem estar obrigatoriamente disponíveis.
  - ✓ size: Tamanho da fonte em pixels.
  - ✓ style: Estilo da fonte. Os estilos possíveis são: ‘bold’, ‘italic’, ‘bold-italic’ ou nil se nenhum estilo for definido.
- **canvas:attrFont():** Retorna a fonte do canvas.
- **canvas:attrFlip(horiz, vert):** Espelhamento do canvas usado em funções de composição. Parâmetros:
  - ✓ horiz: Assume o valor booleano true se o espelhamento for horizontal.
  - ✓ vert: Assume o valor booleano true se o espelhamento for vertical.
- **canvas:attrFlip():** Retorna a configuração de espelhamento do canvas.
- **canvas:attrOpacity(opacity):** Altera a opacidade do canvas. Parâmetros:
  - ✓ opacity: Novo valor de opacidade do canvas.
- **canvas:attrOpacity():** Retorna a opacidade do canvas.
- **canvas:attrRotation(degress):** Configura o atributo de rotação do canvas. Parâmetros:
  - ✓ degrees: Rotação do canvas em graus (Deve ser múltiplo de 90).
- **canvas:attrRotation():** Retorna o atributo de rotação do canvas.

- **canvas:attrScale(w, h):** Escalona o canvas com nova largura e altura. Um dos parâmetros pode tomar o valor true, indicando que a proporção do canvas deve ser mantida. Parâmetros:
  - ✓ w: Largura do escalonamento do canvas.
  - ✓ h: Altura do escalonamento do canvas.
- **canvas:attrScale():** Retorna os valores de escalonamento do canvas.
- **canvas:drawLine(x1, y1, x2, y2):** Desenha uma linha. A linha tem extremidades em (x1, y1) e (x2, y2).
- **canvas:drawRect(mode, x, y, width, height):** desenha um retângulo.
  - ✓ mode: Modo de desenho. Pode receber frame para desenhar a moldura ou fill para preenchê-lo.
  - ✓ x: Coordenada do retângulo.
  - ✓ y: Coordenada do retângulo.
  - ✓ width: Largura do retângulo.
  - ✓ height: Altura do retângulo.
- **canvas:drawRoundRect(mode, x, y, width, height, arcWidth, arcHeight):** Desenha um retângulo com as bordas arredondadas. Alguns dos parâmetros são semelhantes aos do canvas:drawRect, os adicionais são:
  - ✓ arcWidth: Largura do arco do canto arredondado.
  - ✓ arcHeight: Altura do arco do canto arredondado.
- **canvas:drawPolygon(mode):** Desenha um polígono. O parâmetro mode pode receber:
  - ✓ open: para desenhar o polígono sem ligar o último ponto ao primeiro.
  - ✓ close: para desenhar o polígono ligando o último ponto ao primeiro.
  - ✓ fill: para desenhar o polígono ligando o último ponto ao primeiro e colorir a região interior.
- **canvas:drawEllipse(mode, xc, yc, width, height, ang_start, ang_end):** Desenha uma elipse. Parâmetros:
  - ✓ mode: Modo do desenho. Pode receber arc para desenhar apenas a circunferência ou fill para preenchimento interno.
  - ✓ xc: Coordenada x do centro da elipse.
  - ✓ yc: Coordenada y do centro da elipse.
  - ✓ width: Largura da elipse.

- ✓ height: Altura da elipse.
- ✓ ang_start: Ângulo de início.
- ✓ ang_end: Ângulo de fim.

- **canvas:drawText(x, y, text):** Desenha um texto. Parâmetros:
  - ✓ x: Coordenada do texto.
  - ✓ y: Coordenada do texto.
  - ✓ text: Texto a ser desenhado.
- **canvas:clear(x, y, w, h):** Limpa o canvas com a cor configurada em attrColor. Parâmetros:
  - ✓ x: Coordenada da área de clear.
  - ✓ y: Coordenada da área de clear.
  - ✓ w: Largura da área de clear.
  - ✓ h: Altura da área de clear.
- **canvas:flush():** Atualiza o canvas após operações de desenho e de composição. É suficiente chamá-lo apenas uma vez após uma sequência de operações.
- **canvas:compose(x, y, src, [src_x, src_y, src_width, src_height]):** Faz sobre o canvas (canvas destino), em sua posição (x,y), a composição pixel a pixel com src (canvas de origem). Os parâmetros entre colchetes são opcionais, quando ausentes, o canvas inteiro é composto. Parâmetros:
  - ✓ x: Posição da composição.
  - ✓ y: Posição da composição.
  - ✓ src: Canvas a ser composto.
  - ✓ src_x: Posição da composição no canvas src.
  - ✓ src_y: Posição da composição no canvas src.
  - ✓ src_width: Largura da composição no canvas src.
  - ✓ src_height: Algura da composição no canvas src.
- **canvas:pixel(x, y, R, G, B, A):** Altera a cor de um pixel do canvas. Parâmetros:
  - ✓ x: Coordenada x do pixel.
  - ✓ y: Coordenada y do pixel.
  - ✓ R: Componente vermelha da cor.
  - ✓ G: Componente verde da cor.

- ✓ B: Componente azul da cor.
- ✓ A: Componente alpha da cor.

- **canvas:pixel(x, y):** Retorna a cor de um pixel do canvas. Parâmetros:
  - ✓ x: Coordenada x do pixel.
  - ✓ y: Coordenada y do pixel.
- **canvas:measureText(text):** Retorna as coordenadas para o texto passado. Parâmetros:
  - ✓ text: Texto a ser medido.

2.4.3.3 Módulo settings

O módulo settings é utilizado para carregar arquivos de configuração. Propriedades de um nó settings só podem ser modificadas por meio de elos NCL. Não é permitido atribuir valores aos campos representando variáveis nos nós settings.

2.4.3.4 Módulo persistent

Aplicações NCLua permitem que dados sejam armazenados em uma área restrita e recuperados entre execuções.

O exibidor Lua define uma área reservada, inacessível a objetos NCL não procedurais. Não existe nenhuma variável pré-definida ou reservada e objetos procedurais podem atribuir valores a essas variáveis diretamente.

Mais informações sobre NCLua podem ser encontradas no manual de referência disponível em: http://www.lua.inf.puc-rio.br/~francisco/nclua/referencia/index.html (LUA, 2007).

3. UMA METODOLOGIA PRÁTICA PARA O DESENVOLVIMENTO DE APLICAÇÕES INTERATIVAS UTILIZANDO GINGA-NCL

No cenário de desenvolvimento de aplicações interativas para TV digital, existe uma grande variedade de opções no que diz respeito à utilização de ferramentas de desenvolvimento, linguagens de programação e simuladores.

Pelo fato do Ginga ainda estar em processo de criação e difusão, muita das ferramentas e simuladores utilizados no processo de desenvolvimento de aplicações ainda estão sendo aperfeiçoadas, o que faz com que as informações sobre o assunto se encontrem de maneira dispersa e muitas vezes desatualizadas.

Neste capítulo, serão apresentadas as etapas necessárias para o desenvolvimento de aplicações interativas para TV digital utilizando Ginga-NCL, na tentativa de fornecer um suporte adequado e atualizado para os que desejam se iniciar na programação para TVD.

A metodologia proposta pode ser representada pelo fluxograma da Figura 3, que será detalhada no decorrer do capítulo:

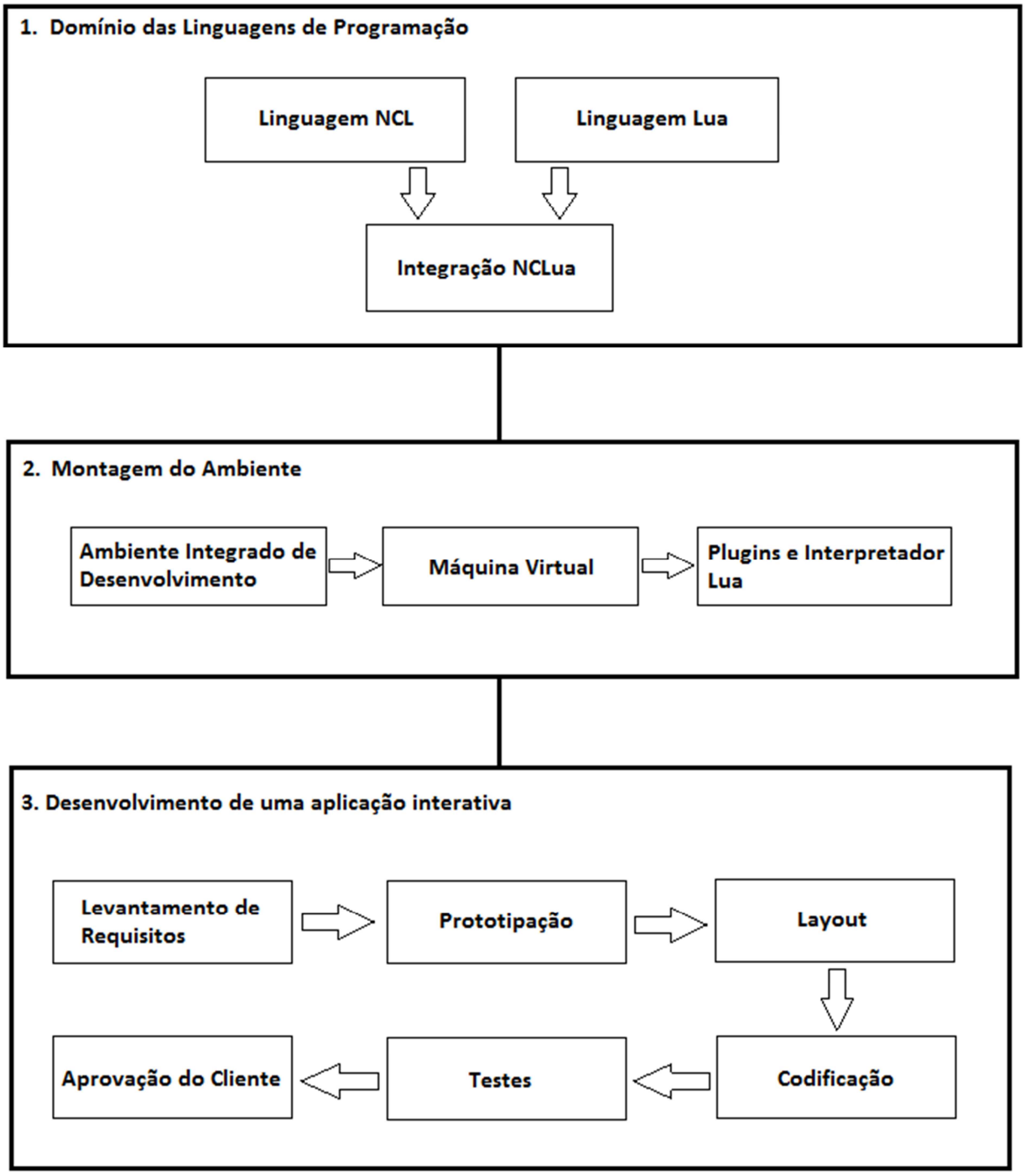

Figura 3. Fluxograma de uma metodologia prática para o desenvolvimento de aplicações interativas para TVD utilizando Ginga-NCL.

3.1 Domínio das Linguagens de Programação

A primeira etapa para o desenvolvimento de uma aplicação é a de conhecer e ter domínio sobre a linguagem de programação que será utilizada.

A compreensão da linguagem de programação irá proporcionar ao desenvolvedor a habilidade necessária para a criação de aplicações. Não é necessário um domínio total das linguagens, porém, fica evidente que quanto mais conhecimento se tiver da linguagem mais eficiente será o processo de codificação.

Para o desenvolvimento desta etapa é altamente recomendada a leitura da seção 2.4 deste trabalho, onde ficaram compreendidos os principais conceitos da linguagem NCL, Lua e da integração NCLua.

3.2 Montagem do Ambiente

Nesta etapa, serão apresentadas as ferramentas necessárias para a criação de aplicações interativas para TVD.

É importante deixar claro que as ferramentas apresentadas aqui são apenas algumas, das muitas opções, que existem para a criação e simulação de aplicações interativas. Porém, houve a preocupação em se utilizar as ferramentas mais atualizadas e preferidas pela comunidade Ginga.

3.2.1 Ambiente Integrado de Desenvolvimento

Um Ambiente Integrado de Desenvolvimento ou IDE, do inglês *Integrated Development Enviroment*, conforme apontado por Bertollo (BERTOLLO et. al., 2002), tem o objetivo de promover um ambiente capaz de suportar todo o processo de desenvolvimento, com diversas ferramentas integradas trabalhando em conjunto.

No contexto de TV digital existem duas IDEs que são amplamente conhecidas para o desenvolvimento de aplicações, a IDE Eclipse e o Composer.

O Composer é uma ferramenta de autoria de documentos NCL criado pela PUC-Rio. Devido a sua interface amigável não é necessário grande domínio em

programação para criação de aplicações, entretanto este ambiente não dá suporte à utilização de *scripts* luas.

O Eclipse é uma IDE de desenvolvimento criado para a linguagem Java e segue o modelo open source de desenvolvimento de software. O projeto foi inicialmente criado pela IBM que desenvolveu sua primeira versão e doou-o como software livre para a comunidade. Uma característica do eclipse é a forte orientação ao desenvolvimento baseado em plug-ins.

Através dos plug-ins, o eclipse pode ser usado não só para desenvolver em Java, mas também entre diversas linguagens, entre elas NCL e Lua.

No Quadro 18 são mostradas informações sobre a IDE Eclipse, inclusive o link para download:

Nome:	Eclipse IDE
Descrição:	Ambiente integrado de desenvolvimento open source que faz uso de plug-ins para dar suporte a diversas linguagens de programação.
Pré-requisito:	O JDK (Java Development Kit) precisa estar instalado no sistema operacional.
Licença:	Gratuita
Link download:	http://www.eclipse.org/downloads/

Quadro 18. Informações Eclipse IDE.

3.2.2 Máquina Virtual

Para conseguirmos simular uma TV com interatividade e testar nossos exemplos, o laboratório Telemídia da PUC-Rio, criou uma máquina virtual com o *middleware* Ginga-NCL instalado.

Na ciência da computação, máquina virtual é o nome dado a uma máquina, implementada através de software, que executa programas como um computador real. (WIKIPEDIA, 2011).

O Ginga-NCL Virtual STB é um sistema Linux que implementa os mais avançados recursos de apresentação de aplicações declarativas, melhor desempenho e maior proximidade de uma implementação embarcada em *set-top box* reais.

Na comunidade Ginga, no Portal do Software publico é possível notar a dificuldade enfrentada por usuários não-avançados para colocar a máquina virtual ginga em funcionamento.

No Quadro 19 são mostradas informações sobre a máquina virtual Ginga, inclusive o link para download:

Nome:	Ginga-NCL Virtual STB
Descrição:	Maquina virtual Linux para VMWare, incluindo Ginga-NCL C++ v. 0.12.3
Pré-requisito:	Ginga-NCL Virtual STB é uma virtual appliance para produtos VMWare. Assim, o VMWare Player é necessário.
Licença:	Gratuita
Link download:	http://www.gingancl.org.br/pt-br/ferramentas

Quadro 19. Informações Ginga-NCL Virtual STB.

Para executarmos o Ginga-NCL Virtual STB é necessário ter instalado um player para máquina virtual. A comunidade Ginga recomenda a utilização do VMware Player por ser gratuito, de fácil instalação e entendimento.

No Quadro 20 são mostradas informações sobre o VMware Player, inclusive o link para download:

Nome:	VMware Player
Descrição:	É um software usado para a criação e emulação de máquinas virtuais em seu computador, algo ideal para a utilização de diferentes sistemas operacionais em um mesmo PC, de maneira simples. É um dos mais clássicos do gênero.
Licença:	Gratuita
Link download:	http://downloads.vmware.com/d/info/desktop_end_user_computing/vmware_player/4_0

Quadro 20. Informações VMware Player.

Após a instalação do VMware Player, para executar o Ginga-NCL basta clicar em Open a Virtual Machine e procurar pela imagem da máquina virtual Ginga-NCL Virtual STB.

Na Figura 3 é possível ver a imagem após o carregamento da máquina virtual:

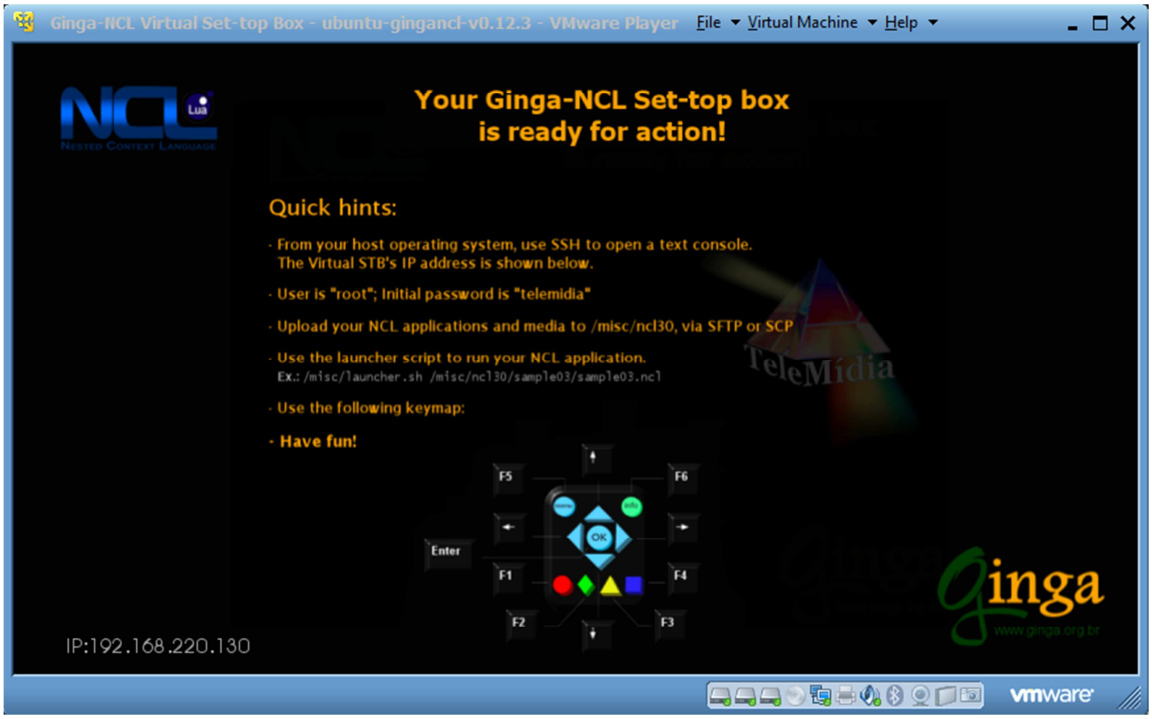

Figura 4. Ginga-NCL Set-top box carregado.

Apesar de a máquina virtual estar rodando no nosso computador, ela se comporta como uma máquina remota. Então para podermos transferir nossa aplicação para o *set-top box* e testá-lo precisaremos de uma conexão FTP e SSH, respectivamente.

Para transferir nossa aplicação ao set-top box devemos utilizar um cliente FTP, que permita a transferência e manipulação de arquivos remotos. O IP, login e senha necessários para a conexão são exibidos na tela do set-top box assim que carregado.

Depois de transferir a aplicação, precisaremos de um cliente SSH para abrir o console do set-top box e digitar a linha de comando necessário para execução do nosso documento.

Na tela da máquina virtual é exibido um exemplo de como rodar uma aplicação. O IP, login e senha são os mesmo utilizados na conexão FTP.

Figura 5. Exemplo de como rodar uma aplicação NCL.

3.2.3 Plug-ins e Interpretador Lua

Para criação de documentos NCL e Lua no eclipse, é necessária a instalação de plug-ins para que a IDE passe a dar suporte a essas linguagens.

O Plug-in NCL Eclipse tem o objetivo de agilizar o desenvolvimento de aplicações para TV digital Interativa em NCL. Permite que todas as facilidades deste conhecido ambiente sejam reutilizadas, como a sugestão de código e outras, facilitando sua integração com outras ferramentas de desenvolvimento.

Baseia-se em princípios claros de produtividade, facilidade e integração no desenvolvimento de conteúdo interativo para TV digital.

O NCL-Eclipse pode ser instalado através do sistema de instalação automática do Eclipse.

Para efetuar a instalação do NCL Eclipse inicie o Eclipse e acesse Help -> Install New Software. O nome das opções pode variar de acordo com a versão do eclipse, porém, o contexto da instalação será o mesmo.

Após clicar em Add será exibida uma caixa de diálogo para a informação do nome e localização do site onde o Eclipse irá buscar a atualização. Entre com as informações:

- Name: NCL Eclipse
- Location: http://www.laws.deinf.ufma.br/ncleclipse/update

A Figura 5 demonstra a adição do repositório NCL-Eclipse:

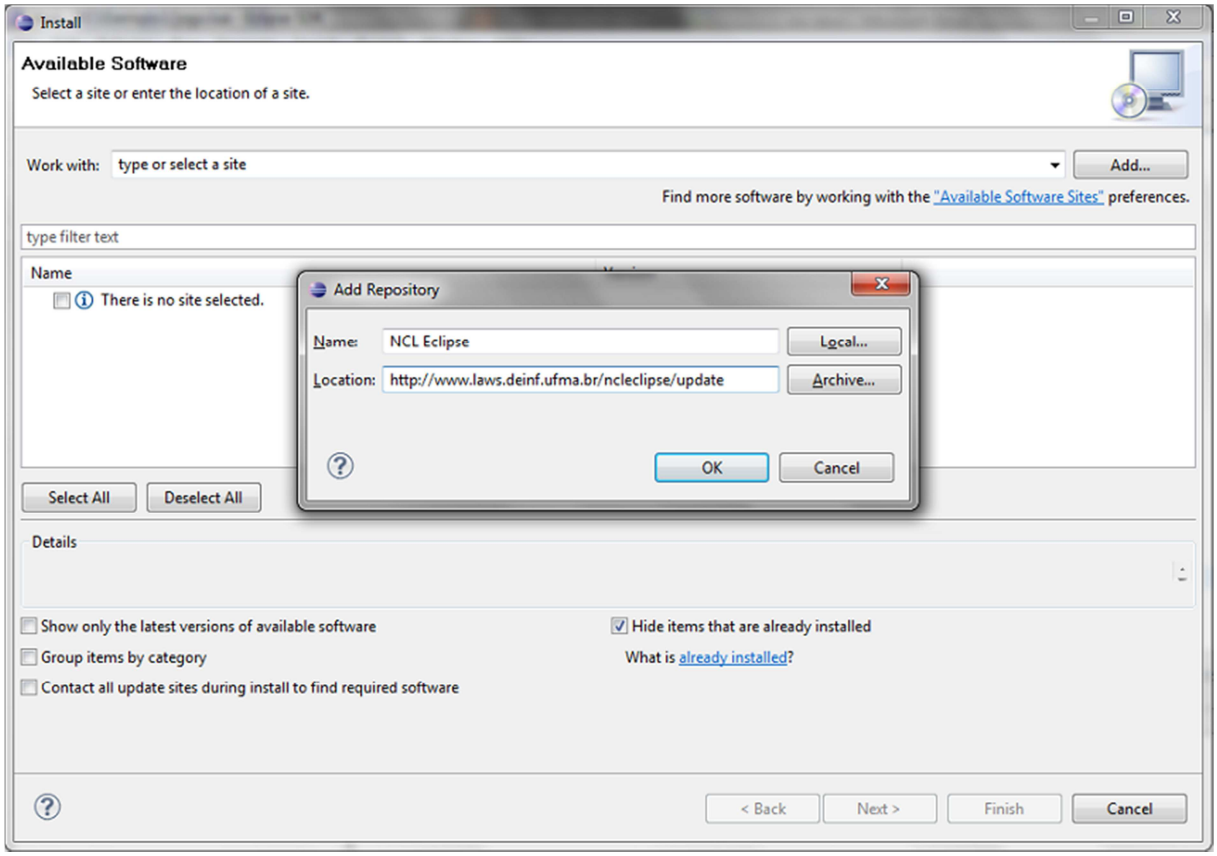

Figura 6. Adicionando o repositório NCL Eclipse.

Após adicionar o repositório, selecione a ultima versão do NCL Eclipse e conclua a instalação.

Para criar um novo documento NCL clique em File -> New -> Other e escolha NCL -> NCL Document.

Após estas etapas será possível criar documentos NCL, porém para criação de *scripts* Lua é necessário instalar outro *plug-in* chamado Lua Eclipse.

Com o Lua Eclipse é possível editar *scripts* Lua com destaque de sintaxe, de conclusão de código, verificação de erros de compilação, agrupamento de código e comentários, a execução do script com um interpretador pré-configurado, além das ferramentas que a plataforma Eclipse fornece.

A instalação segue o padrão do Eclipse é semelhante com a instalação do NCL Eclipe. Basta adicionar um novo repositório com as seguintes informações:

- Name: Lua Eclipse
- Location: http://luaeclipse.luaforge.net/preview/update-site/win32.win32.x86/

Após a instalação do Lua Eclipse, devemos instalar o interpretador Lua que pode ser obtido no site http://code.google.com/p/luaforwindows/downloads/list e instalado de forma simples através de um executável.

Para que os *scripts* Lua possam rodar é necessário configurar no Eclipse o interpretador Lua padrão para nossos aplicativos. Para isso, selecione Window -> Preferences e em seguida selecione a categoria Lua -> Installed Interpreters e adicione o interpretador Lua instalado anteriormente como na Figura 6:

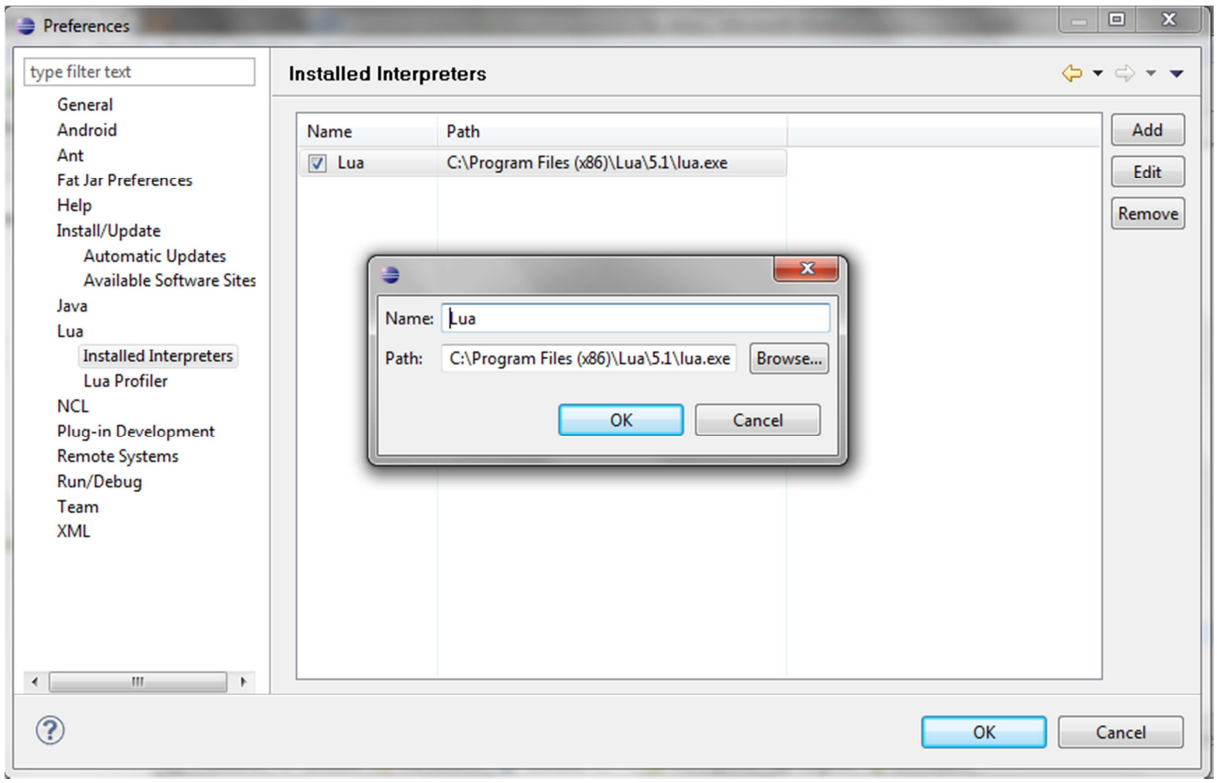

Figura 7. Adicionando o interpretador Lua.

Para incluir um novo arquivo em seu projeto selecione o projeto desejado, clique com o botão direito e selecione New -> New Lua File.

3.3 Desenvolvimento de uma aplicação interativa

Esta etapa é a responsável pela criação de uma aplicação interativa onde o modelo de desenvolvimento adotado foi o TQTVD (TOTVS – QUALITY para TV

Digital), empresa responsável pelo desenvolvimento de uma implantação profissional do *middleware* Ginga, demonstrado por Santos (SANTOS, 2009) e apresentado na Figura 7:

Figura 8. Modelo de processo para desenvolvimento baseado TQTVD (Fonte: SANTOS, 2009).

- Levantamento de Requisitos: Se faz necessário para a identificação de problemas relativos ao escopo, apresentando as funcionalidades e prioridades que o software terá, a fim de eliminar ambiguidades que possam vir a causar problemas no decorrer do projeto.
- Conceito e Prototipação: É uma abordagem baseada numa visão evolutiva do desenvolvimento da aplicação. Envolve a produção de versões de um sistema futuro com o qual se podem realizar verificações e experimentações.
- Layout: São versões não operacionais do sistema, necessárias para avaliação da interface, a fim de manter a usabilidade da aplicação.
- Codificação: Criação do documento hipermídia e dos scripts da aplicação.
- Testes: Se faz necessário simular a aplicação, a fim de procurar possíveis erros que passaram despercebidos no processo de codificação. A qualidade do software está diretamente ligada aos testes realizados na aplicação.
- Aprovação do Cliente: Aqui é onde a aplicação está concluída somente esperando para que o cliente valide-a e exponha sua opinião.

4. VALIDAÇÃO DA METODOLOGIA PROPOSTA

Para validação da metodologia proposta, neste capítulo será desenvolvida uma aplicação interativa utilizando os métodos do capítulo 3, a fim de por em prova sua eficiência.

4.1 Desenvolvimento da aplicação interativa EADQuiz

A TV digital surge no Brasil como uma poderosa ferramenta de inclusão digital. O governo aposta que com o advento da interatividade na TV, brasileiros que não tem acesso a recursos tecnológicos como um computador ou internet.

Diante deste contexto o aplicativo que será desenvolvido propõe uma forma para avaliar o conhecimento dos telespectadores de programas que exibam material didático por meio de um questionário sobre o tema envolvido, batizado com o nome EADQuiz.

Para o desenvolvimento da aplicação foi seguida toda a metodologia proposta no capítulo 3.

Para cumprimento da primeira etapa de domínio da linguagem de programação, a seção 2.4 deste trabalho foi extremamente relevante, pois foi possível adquirir uma carga teórica e prática a respeito das linguagens utilizadas: NCL, Lua e a integração NCLua.

Na segunda etapa de montagem do ambiente foi seguido o passo a passo mostrado na seção 3.2, com a instalação da IDE Eclipse, os plugins NCL Eclipse, Lua Eclipse, o interpretador Lua e a máquina virtual Ginga. Para o desenvolvimento específico do EADQuiz ainda foram utilizadas ferramentas para o tratamento das imagens da aplicação.

Para a terceira etapa de desenvolvimento de uma aplicação interativa, foi seguido o modelo TQTVD apresentado na seção 3.3 no contexto da aplicação proposta, EADQuiz, que é detalhado nas seções a seguir.

Apenas a etapa de aprovação não foi realizada devido a não utilização do aplicativo em um contexto real.

4.1.1 Levantamento de Requisitos

Esta etapa se faz necessário para a identificação de problemas relativos ao escopo, apresentando as funcionalidades e prioridades que o software terá.

4.1.1.1 Definição do perfil do usuário

O telespectador de programas didáticos será a pessoa beneficiada com a aplicação criada, pois, poderá ao final do programa testar o seu conhecimento respondendo algumas questões a fim de saber se o conteúdo foi realmente assimilado.

Para responder o questionário não serão necessários grandes conhecimentos em informática, pois as questões serão respondidas através dos quatro botões de interatividade, facilmente identificados no controle pela sua cor e formato diferenciados.

4.1.1.2 Ferramentas utilizadas

Foi necessário fazer uso de alguns aplicativos para auxiliar no desenvolvimento da aplicação. Segue a lista abaixo dos programas e o papel desempenhado no projeto:

- Balsamiq Mockups Web Demo: Ferramenta de prototipação de telas. Pode ser acessado pelo endereço: http://builds.balsamiq.com/b/mockups-web-demo/?q=demos/mockups/Mockups.html
- Eclipse IDE (Helios): Ambiente integrado de desenvolvimento.
- Ginga-NCL Virtual STB v0.12.3: Máquina virtual Ginga-NCL.
- VMware Player v4.0: Ferramenta de virtualização.
- WinSCP: Cliente FTP.
- Putty: Cliente SSH.
- CorelDraw X5 e Fireworks CS5: Design e tratamento de imagens.

Dentre os plug-ins para eclipse empregados estão:

- NCL Eclipse v1.5.2.
- Lua Eclipse v1.3.1.

O aplicativo foi desenvolvido e testado utilizando o sistema operacional Windows Seven 64bits.

4.1.2 Prototipação

A fim de determinar um produto final para que as iterações ocorressem com o mínimo de riscos para entrega do projeto, foi estabelecido um modelo de protótipo evolutivo.

A prototipação é constituída das partes essenciais do projeto onde podem ser adicionadas novas funcionalidades e melhorias no software, sendo ela baseadas no levantamento de requisito, seguindo a prioridade do documento. Assim, para viabilização de negócio com a aplicação desenvolvida, foram contemplados os seguintes objetivos:

- Entendimento de requisitos.
- Modelagem da interface das telas.
- Criação de um protótipo inicial do documento NCL.
- Criação de um protótipo inicial do script Lua.
- Integração dos protótipos;

4.1.3 Layout

Segundo Maracci (MARACCI et. al., 2009) layout pode ser definido como protótipos não-operacionais do sistema, ficando compreendido para o escopo desse projeto.

4.1.3.1 Modelo de Telas

O modelo das telas foi criado utilizando a ferramenta on-line de prototipação de sistemas Balsamiq Mockups.

O modo de trabalho no Balsamiq é bastante simples, a inteface do sistema é composta por:

- Área de trabalho: onde será desenhado o protótipo;

- Lista de componentes: componentes utilizados para a elaboração do protótipo;
- Barra de categorias de componentes: categorias utilizadas para selecionar os componentes com maior agilidade;
- Barra de menus: menus do sistema, como salvar, importar, entre outros;
- Barra de ícones rápidos: utilizados para funções tradicionais, como copiar, colar, agrupar e desagrupar elementos, etc.

As figuras a seguir representam o modelo de telas criado no Balsamiq Mockups e seus comentários:

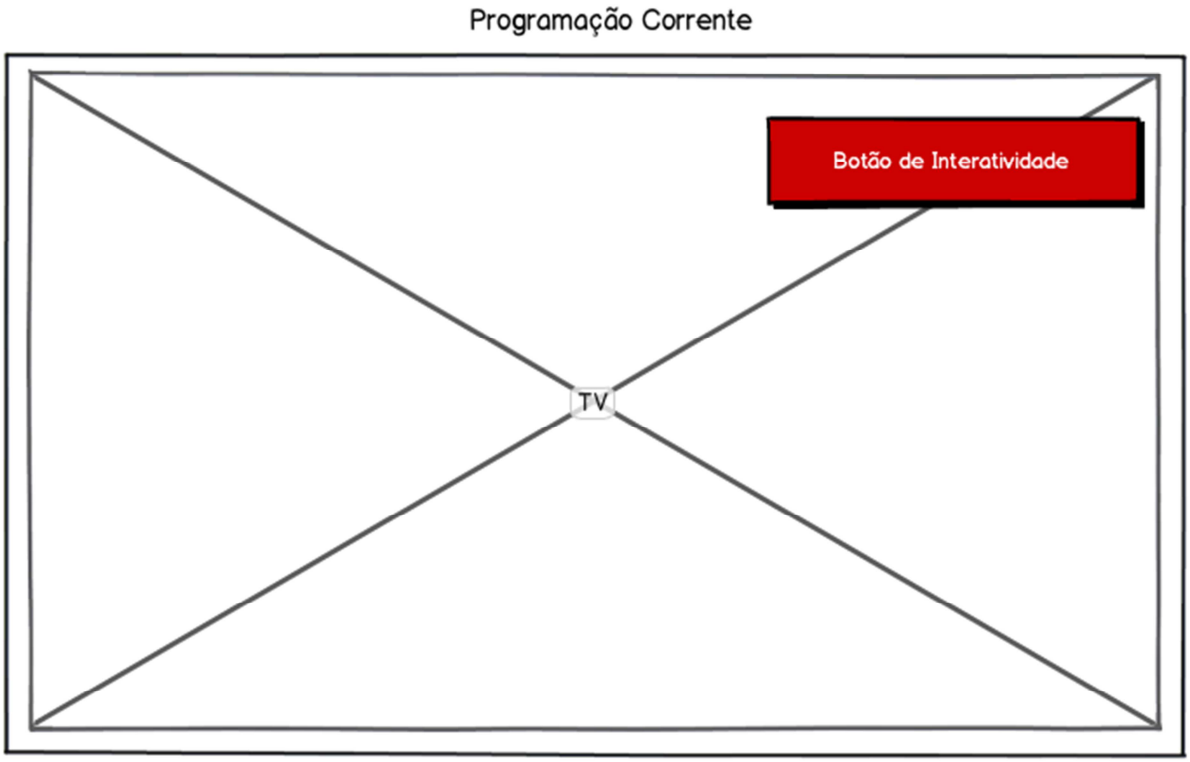

Figura 9. Modelo de Tela inicial criado no Balsamiq Mockups.

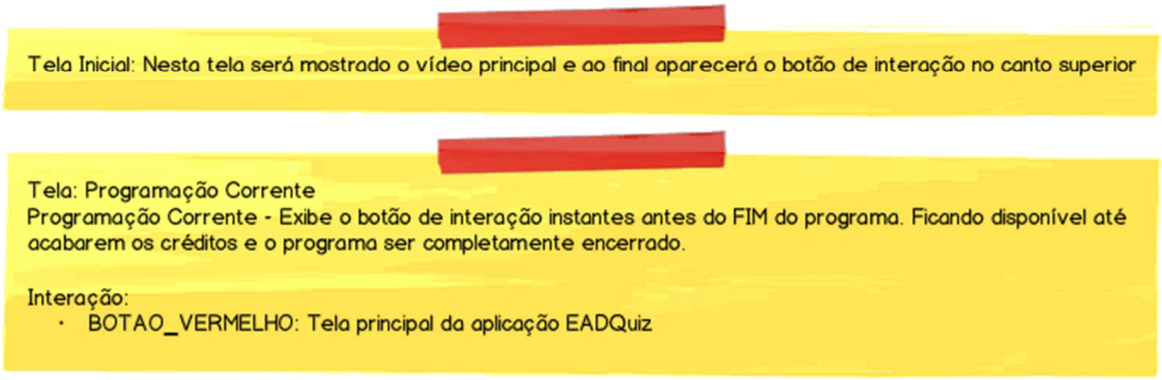

Figura 10. Comentários do Modelo de Tela criado no Balsamiq Mockups.

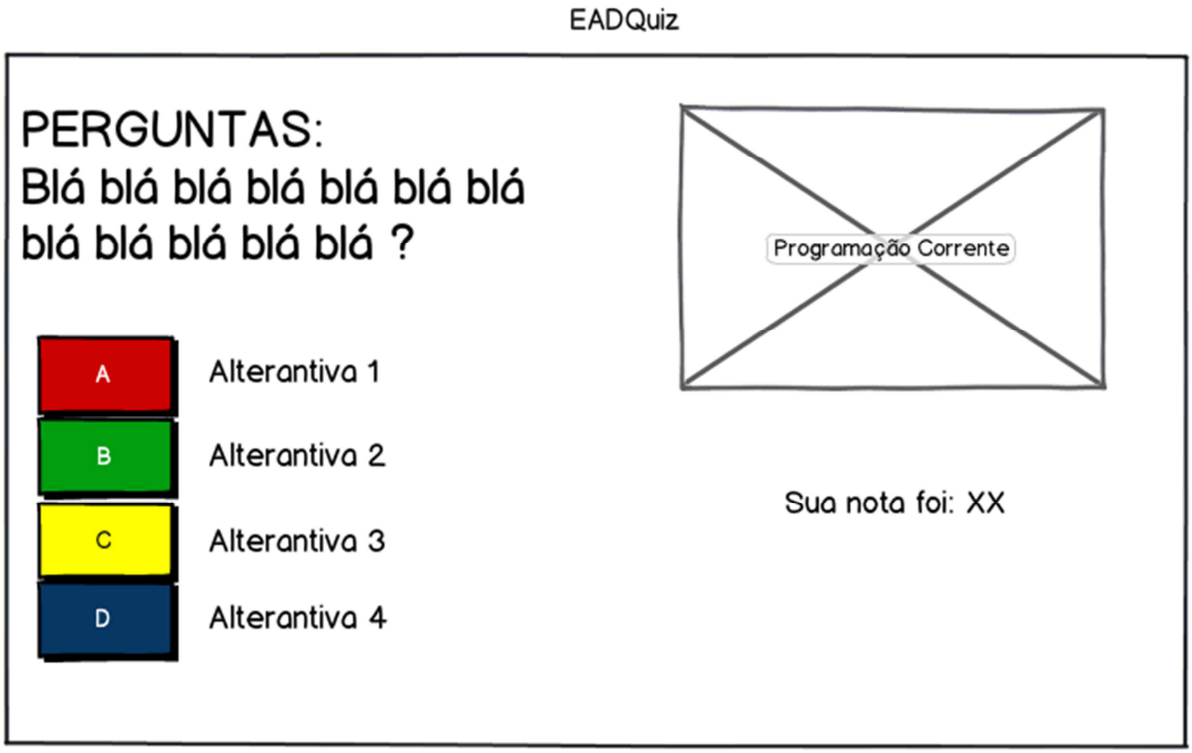

Figura 11. Modelo de Tela do script criado no Balsamiq Mockups.

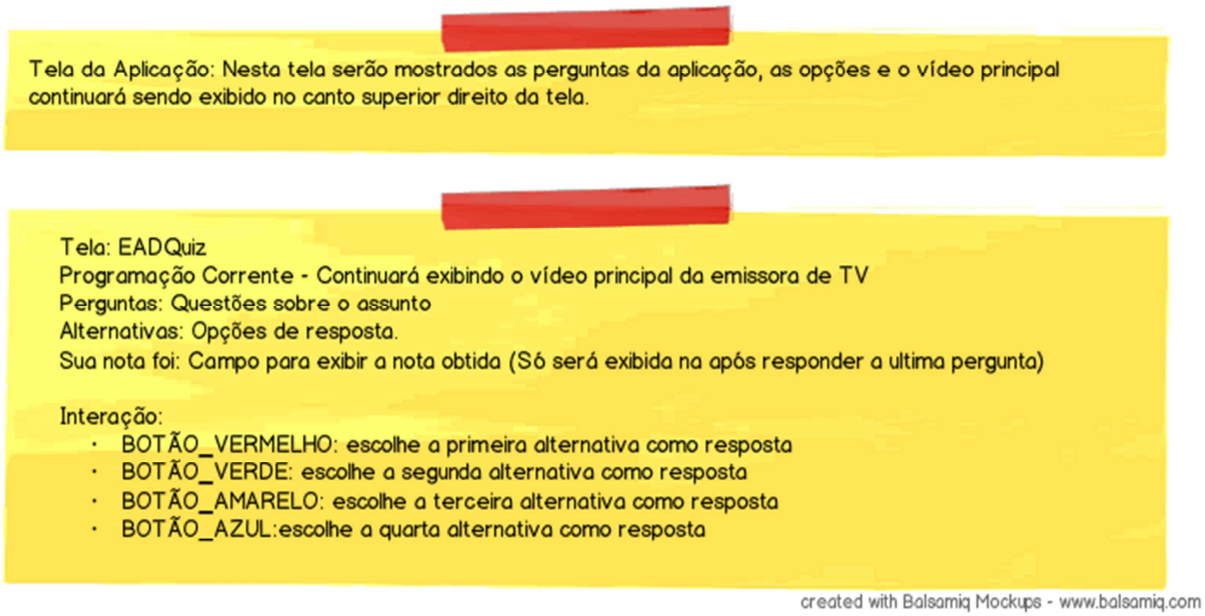

Figura 12. Comentários do Modelo de Tela do script criado no Balsamiq Mockups.

4.1.4 Codificação

O principal objetivo do EADQuiz é aplicar um questionário ao final de um programa didático, para poder testar os conhecimentos do telespectador e avaliá-lo através de uma nota.

Assim o código NCL ficou responsável pela inicialização do vídeo e composição do botão de interação para o início do script lua.

No trecho de código representado na Figura 12 é possível ver o elo responsável por iniciar a aplicação Lua quando o botão de interação vermelho é pressionado. É possível observar também, que o vídeo não é interrompido para a execução do script, ou seja, podemos realizar a prova sem interromper a exibição da programação da emissora de TV.

```
<link xconnector="onSelectionSetStartStop">
    <bind role="onSelection" component="botao" >
        <bindParam name="keyCode" value="RED"/>
    </bind>
    <bind role="set" component="video" interface="height">
        <bindParam name="value" value="40%"/>
    </bind>
        <bind role="set" component="video" interface="width">
        <bindParam name="value" value="40%"/>
    </bind>
    <bind role="set" component="video" interface="top">
        <bindParam name="value" value="30"/>
    </bind>
    <bind role="set" component="video" interface="right">
        <bindParam name="value" value="30"/>
    </bind>
    <bind role="stop" component="botao"/>
    <bind role="start" component="lua"/>
</link>
```

Figura 13. Trecho de código NCL para inicialização do script Lua.

O script lua foi responsável por desenhar a tela da aplicação com as perguntas e alternativas, além do processamento e exibição da nota ao telespectador.

O código completo do documento NCL e script Lua são encontrados no apêndice B.

4.1.5 Simulação e Testes

Para realizar a simulação e teste do aplicativo, utilizamos a máquina virtual Ginga-NCL Virtual STB v0.12.3.

Após emular a máquina virtual no VMware Player é necessário criar uma conexão FTP para transferir os arquivos da aplicação, e uma conexão SSH para estabelecer uma conexão com o sistema operacional.

As principais informações para a utilização do set-top box se encontram na tela inicial da máquina virtual, assim como as teclas que representam os comandos do controle na TV real.

Figura 14. Tela de informações para utilização do set-top Box.

Desse modo o protótipo foi simulado e testado como em um set-top-box real a fim de validar as questões de interface com o usuário e validação dos requisitos.

Nas Figura 14 e 15 é possível observar a tela inicial e final da aplicação, respectivamente.

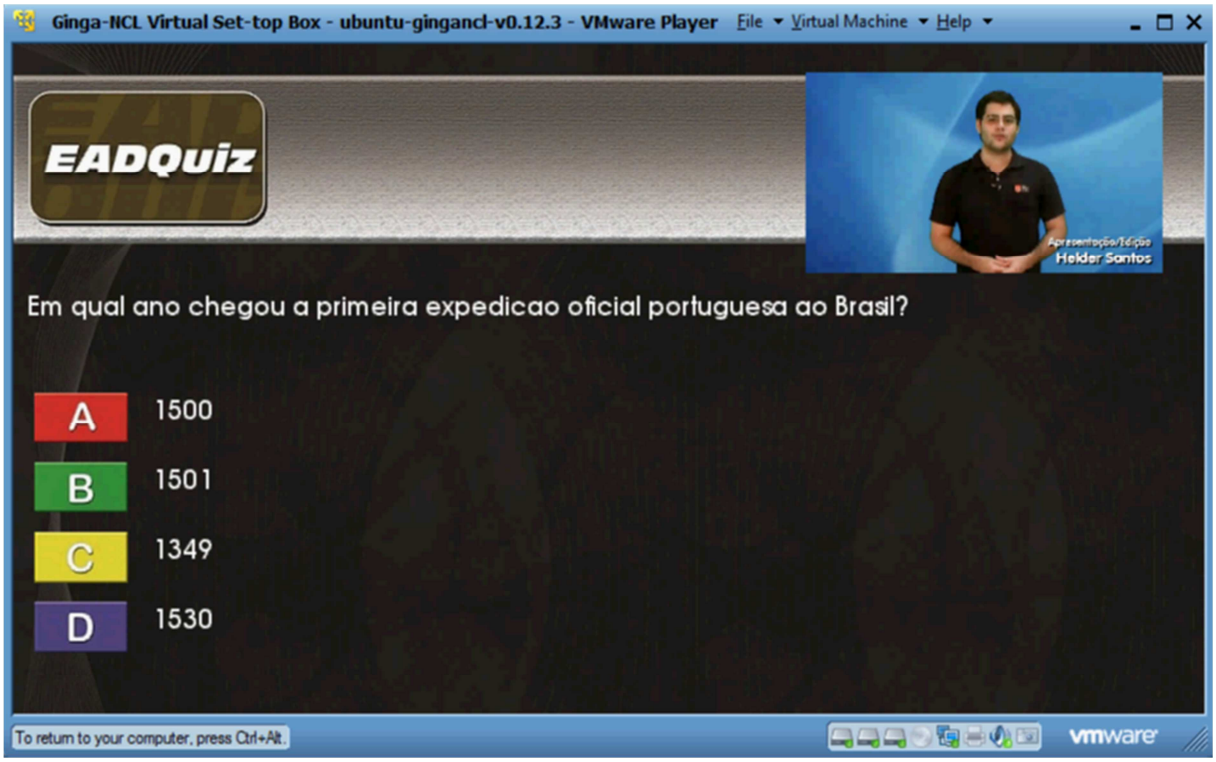

Figura 15. Tela inicial da aplicação.

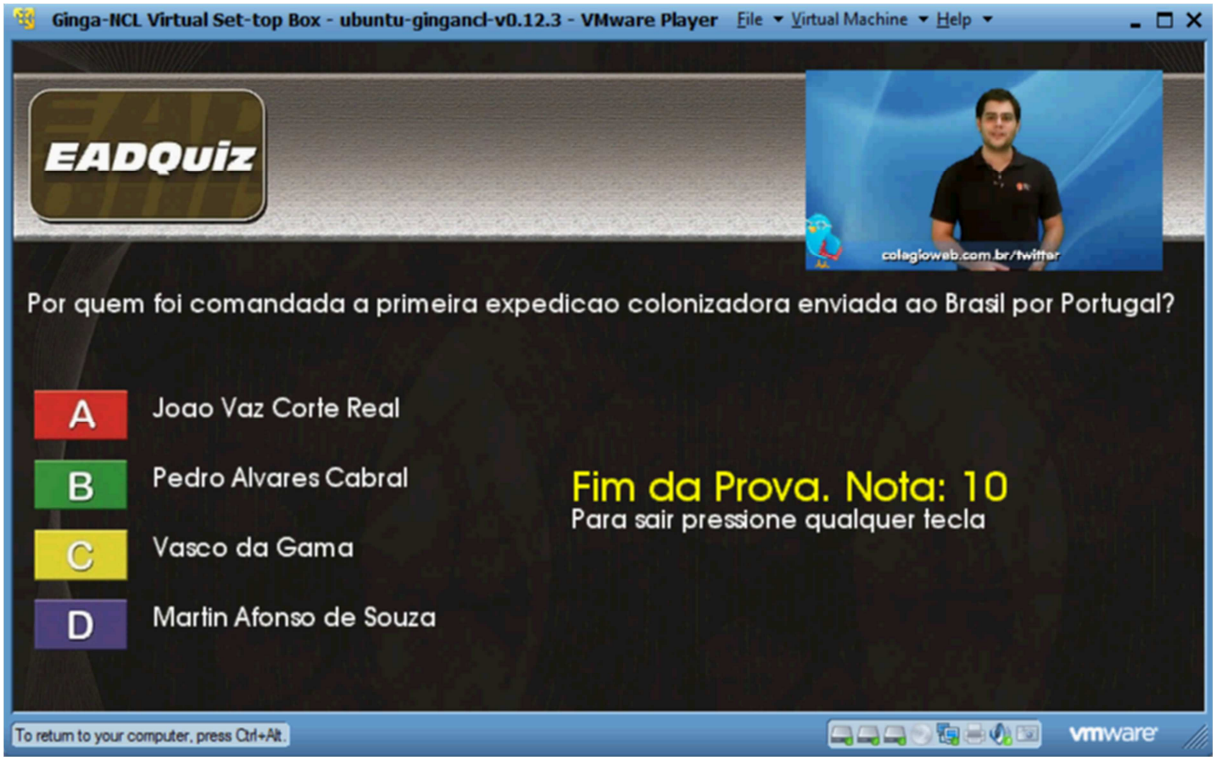

Figura 16. Tela final da aplicação.

4.2 Avaliação da Metodologia Proposta

Diante dos resultados obtidos com a aplicação EADQuiz, ficou evidente que a metodologia se mostrou eficaz, se aplicada corretamente.

Os resultados se mostraram bastante satisfatórios, porém, é importante observar que apesar dos testes realizados a aplicação não foi utilizada em um contexto real, ficando a lacuna em relação à aprovação por parte do cliente.

Dessa forma é possível que a metodologia contenha etapas em que seja necessário um maior aprofundamento na criação de outras aplicações, pois, mesmo mostrando um desempenho satisfatório, ela ainda não foi testada em todos os contextos reais possíveis.

Portanto, a metodologia proposta se mostra relevante, visto que a aplicação EADQuiz superou as expectativas e obteve êxito em todos os testes realizados.

5. CONCLUSÃO

Os conceitos a respeito das tecnologias envolvidas na TVD aberta brasileira ainda carecem de fontes de informações e ferramentas de suporte a estudantes e profissionais da área da informática. Por esta razão, este trabalho concentra suas contribuições na criação de uma alternativa no sentido de facilitar e popularizar o desenvolvimento de aplicações interativas para TVD. Seguindo este objetivo, o trabalho aqui descrito, teve como proposta a criação de uma metodologia prática para o desenvolvimento de aplicações interativas.

O trabalho proposto apresentou as informações técnicas a cerca do SBTVD e do *middleware* declarativo Ginga-NCL. Os resultados obtidos puderam ser colocados a prova através da criação de um aplicativo interativo que serve de objeto de contribuição ao sistema de ensino brasileiro: Educação à distância pela TVD.

Na Figura 3 obteve-se uma visão estruturada e sintetizada das etapas desenvolvidas nesse trabalho, que se seguidas corretamente podem facilitar e popularizar o acesso de novos desenvolvedores que desejam ingressar na área de TV digital.

5.1 Trabalhos Futuros

Para trabalhos futuros pode-se tornar o script Lua configurável, a fim de tornar a aplicação reutilizável. Dessa forma ela poderia ser utilizada por vários programas didáticos alterando-se somente um arquivo de configuração contendo as perguntas e alternativas.

Criação de uma metodologia prática para o outro subsistema do SBTVD, Ginga-J, que não foi abordado nesta pesquisa.

O aprofundamento dos estudos quanto aos impactos que a Educação à distância pela TVD na sociedade brasileira, e como isso pode ser utilizado por instituições de ensino juntamente as emissoras de TV.

O desenvolvimento de *frameworks* e ferramentas de desenvolvimento, a fim de abstrair as complexibilidades técnicas para que usuários sem tanto conhecimento em programação possam desenvolver suas aplicações sem perda de qualidade.

6. REFERÊNCIAS

ABERT/SET Group Brazil. **Brazilian Digital Television Tests.** Abril de 2000. Disponível em: http://www.set.com.br/artigos/nab.pps - Acessado em 14 de junho de 2011.

ABNT. **Associação Brasileira de Normas Técnicas.** Disponível em: http://www.abnt.org.br/default.asp - Acessado em 14 de junho de 2011.

ABNT. **ABNT NBR 15606-2:2007.** Rio de Janeiro, RJ, 2007. Disponível em: http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15606-2_2007Vc_2008.pdf - Acessado em 7 de setembro de 2011.

BARBOSA, Simone D. J.; SOARES, Luiz F. G. **TV Interativa no Brasil se faz com Ginga: Fundamentos, Padrões, Autoria Declarativa, e Usabilidade.** p. 105-174. Rio de Janeiro, RJ, 2008.

BECKER, Valdecir; FORNARI, Augusto; HERWEG, Günter H. Filho; MONTEZ, Carlos. **Recomendações de Usabilidade para TV Digital Interativa.** Florianópolis, SC, 2006.

BECKER, Valdecir; MONTEZ, Carlos. **TV DIGITAL INTERATIVA – Conceitos, desafios e perspectivas para o Brasil**. Florianópolis: Ed. da UFSC, 2005. 2ª edição. Disponível em http://www.itvproducoesinterativas.com.br/pdfs/TV-Digita-Interativa_2a_EDICAO.pdf - Acessado em 21 de agosto de 2011.

BELLO, José Luiz de Paiva. **Metodologia Científica.** Rio de Janeiro, RJ, 2004. Disponível em http://www.pedagogiaemfoco.pro.br/met10.htm - Acessado em 21 de agosto de 2011.

BERTOLLO, Gleidson; RUY, Fabiano B.; MIAN, Paula G.; PEZZIN, Juliana; SCHWAMBACH, Mellyssa. NATALI, Ana C. C.; FALBO, Ricardo A. **ODE – Um Ambiente de Desenvolvimento de Software Baseado em Ontologias.** Vitória, ES, 2002.

CARVALHO, Diesmeleno S. **Desenvolvimento para TV Digital Interativa.** Campo Grande, MS, 2011. Disponível em: http://www.diemesleno.com.br/2011/04/curso-desenvolvimento-para-tv-digital-interativa/ Acessado em 15 de setembro de 2011.

CARVALHO, Rafael. **Como estruturar o seu ambiente de desenvolvimento para o Ginga-NCL.** Disponível em: http://rafaelcarvalho.tv/2011/como-estruturar-seu-ambiente-de-desenvolvimento-para-o-ginga-ncl-34 - Acessado em 9 de agosto de 2011.

CARVALHO, Rafael.; SANTOS, Joel A. F.; DAMASCENO, Jean R.; SILVA, Julia V.; SAADE, Débora C. M. **Introdução às Linguagens NCL e Lua: Desenvolvendo Aplicações Interativas para TV Digital**. Rio de Janeiro, RJ, 2009

CROCOMO, Fernando Antonio. **TV Digital e Produção Interativa: A Comunidade Recebe e Manda Notícias**. Florianópolis, SC, 2004.

DIAS, A. Gestão **de Marketing**. São Paulo, Saraiva, 2003.

DIAS, Édipo Tenório Holanda. **A Implantação do Sistema Brasileiro de TV Digital: Análise do processo de adaptação das agências de publicidade de Caruaru à nova Tecnologia.** Caruaru, PE, 2010

FERNANDES, Jorge; LEMOS, Guido; ELIAS, Gledson. **Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas.** Salvador, BA, Agosto de 2004.

G1, Globo.com. **Fogão e aparelho de TV são os bens presentes em mais casas brasileiras.** 18 de setembro, 2009. Disponível em: http://g1.globo.com/Noticias/Brasil/0,,MUL1308909-5598,00-FOGAO+E+APARELHO+DE+TV+SAO+OS+BENS+PRESENTES+EM+MAIS+CASAS+BRASILEIRAS.html - Acessado em 12 de julho de 2011.

GINGA. 2010. Disponível em: http://www.ginga.org.br - Acessado em 19 de agosto de 2011.

IBGE. **Pnad 2008: Mercado de trabalho avança, rendimento mantém-se em alta, e mais domicílios têm computador com acesso à Internet**. 18 de Setembro, 2009. Disponível em: http://www.ibge.gov.br/home/presidencia/noticias/noticia_visualiza.php?id_noticia=1455&id_pagina - Acessado em 21 de Agosto de 2011.

IERUSALIMSCHY R.; FIGUEIREDO, L. H.; CELES W.; **Lua 5.1 Reference Manual.** Agosto, 2006. Disponível em: http://www.lua.org/manual/5.1/pt/ - Acessado em 21 de Agosto de 2011.

KOOGAN/HOUAISS. Enciclopédia e dicionário ilustrado. 4.ed. Rio de Janeiro. Seifer, 1999.

LEMOS, A. L.M. **Anjos Interativos e Retribalização do Mundo. Sobre Interatividade e Interafaces Digitais**. 1997.

LUA. **Manual de Referência NCLua.** 2007 Disponível em: http://www.lua.inf.puc-rio.br/~francisco/nclua/referencia/index.html - Acessado em 10 de outubro 2011.

LUA. **The Programming Language.** 2011. Disponível em: http://www.lua.org/ - Acessado em 15 de setembro de 2011.

MALHOTRA, Naresh K. **Pesquisa de marketing: uma orientação aplicada.** 3. .ed. Porto Alegre. Bookman, 2001

MARACCI, Francisco V.; COLANZI, Thelma E.; PADOVAN, Andrea J.; ROSA, Marcelo V. C.; BRANCO, Emanuelle F. S. **WAEI/MSE: uma instância do processo WAE para micro e pequenas empresas de software**. Maringá, PR, 2009.

MONTEZ, Carlos; BECKER, Valdecir. **TV Digital Interativa: Conceitos e Tecnologias. In: WebMidia e LA-Web 2004** – Joint Conference. Ribeirão Preto, SP, Outubro de 2004.

MORAIS, Mavy D. P. **Convergência entre Web e TV Digital: Uma proposta de integração com o SIGA.** Caruaru, PE, 2009.

NCL Eclipse. **Introdução.** 2011. Disponível em: http://www.laws.deinf.ufma.br/~ncleclipse/#.TsFxWD0xXCt - Acessado em 10 de outubro de 2011.

PROTA, Thiago M. **MoonDo: Um Framework para Desenvolvimento de Aplicações Declarativas no SBTVD.** Recife, PE, 2010.

SANTOS, Hélder. **O Papel do Lua no mercado de TV Digital.** Outubro, 2009

SILVA, Antônio Romão A. F. **Manual para Elaboração do Trabalho de Conclusão de Curso (Monografia de Graduação sem Estresse) Metodologia e Técnica de Pesquisa Instrumental.** 2006

SOARES, L. F. G.; BARBOSA, S. D. J. **Programando em NCL 3.0: Desenvolvimento de aplicações para o Middleware Ginga, TV digital Web.** Rio de Janeiro: Elsevier, 2009.

VALIM, Maurício. **A História da Televisão: da sua invenção ao início das transmissões em cores.** Agosto, 1998. Disponível em: http://www.tudosobretv.com.br/historty/histor.htm - Acessado em 10 de julho de 2011.

WIKIPEDIA. **Máquina Virtual.** 2011. Disponível em: http://pt.wikipedia.org/wiki/M%C3%A1quina_virtual - Acessado em 10 de outubro de 2011.

WIKIPEDIA. **SBTVD**. 2011. Disponível em: http://pt.wikipedia.org/wiki/SBTVD - Acessado em 14 de junho de 2011.

ZANCANARO, Airton. SANTOS, Paloma Maria. TODESCO, José Leomar. Ginga-J ou **Ginga-NCL: características das linguagens de desenvolvimento de recursos interativos para a TV Digital.** Bauru, SP, 2009.

Apêndice A – Documento NCL

O código a seguir reproduz um vídeo e a cada 10 segundos exibe uma imagem na tela por 5 segundos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="exemplo01" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
      <head>
            <regionBase>
                  <region id="rg_video" left="0" top="0" width="100%"
height="100%" zIndex="0">
                        <region id="rg_imagem" width="20%" height="20%"
zIndex="0"/>
                  </region>
            </regionBase>
            <descriptorBase>
                  <descriptor id="dp_video" region="rg_video"/>
                  <descriptor id="dp_imagem" region="rg_imagem"
focusIndex="0"/>
            </descriptorBase>
            <connectorBase>
                  <causalConnector id="onBeginStart">
                        <simpleCondition role="onBegin"/>
                        <simpleAction role="start"/>
                  </causalConnector>
                  <causalConnector id="onEndStop">
                        <simpleCondition role="onEnd"/>
                        <simpleAction role="stop" max="unbounded"/>
                  </causalConnector>
            </connectorBase>

      </head>
      <body>
            <port component="video" id="inicio"/>
            <media id="video" src="media/video.mpg" type="video/mpeg"
descriptor="dp_video">
                  <area id="area01" begin="10s" end="15s"/>
                  <area id="area02" begin="25s" end="30s"/>
                  <area id="area03" begin="40s" end="45s"/>
                  <area id="area04" begin="55s" end="60s"/>
            </media>

            <media id="imagem" descriptor="dp_imagem"
src="media/imagem.jpg" type="image/jpeg"/>

            <link xconnector="onBeginStart">
                  <bind role="onBegin" component="video"
interface="area01"/>
                  <bind role="start" component="imagem"/>
            </link>
            <link xconnector="onBeginStart">
                  <bind role="onBegin" component="video"
interface="area02"/>
                  <bind role="start" component="imagem"/>
            </link>
            <link xconnector="onBeginStart">
                  <bind role="onBegin" component="video"
interface="area03"/>
```

```
                  <bind role="start" component="imagem"/>
            </link>
            <link xconnector="onBeginStart">
                  <bind role="onBegin" component="video"
interface="area04"/>
                  <bind role="start" component="imagem"/>
            </link>
            <link xconnector="onEndStop">
                  <bind role="onEnd" component="video" interface="area01"/>
                  <bind role="stop" component="imagem"/>
            </link>
            <link xconnector="onEndStop">
                  <bind role="onEnd" component="video" interface="area02"/>
                  <bind role="stop" component="imagem"/>
            </link>
            <link xconnector="onEndStop">
                  <bind role="onEnd" component="video" interface="area03"/>
                  <bind role="stop" component="imagem"/>
            </link>
            <link xconnector="onEndStop">
                  <bind role="onEnd" component="video" interface="area04"/>
                  <bind role="stop" component="imagem"/>
            </link>
      </body>
</ncl>
```

Apêndice B – Código completo da aplicação EADQuiz

Documento NCL:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Generated by NCL Eclipse -->
<ncl id="main" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
      <head>
            <regionBase>
                  <region id="rgVideo" width="100%" height="100%"
zIndex="1">
                        <region id="rgImagem" left="60%" top="5%"
bottom="87%"/>
                        <region id="rgBotao" left="60%" top="5%"
bottom="87%"/>
                  </region>
                  <region id="rgLua" width="100%" height="100%"
zIndex="0"/>
            </regionBase>

            <descriptorBase>
                  <descriptor id="dVideo" region="rgVideo" />
                  <descriptor id="dImagem" region="rgImagem"/>
                  <descriptor id="dBotao" focusIndex="0" region="rgBotao"/>
                  <descriptor id="dLua" region="rgLua" focusIndex="0" />
            </descriptorBase>

            <connectorBase>
                  <causalConnector id="onBeginStart">
                        <simpleCondition role="onBegin" />
                        <simpleAction role="start" />
                  </causalConnector>

                  <causalConnector id="onEndStop">
                        <simpleCondition role="onEnd" />
                        <simpleAction role="stop" />
                  </causalConnector>

                  <causalConnector id="onSelectionStop">
                        <connectorParam name="keyCode"/>
                        <connectorParam name="value"/>
                        <simpleCondition role="onSelection"
key="$keyCode"/>
                        <simpleAction role="stop"/>
                  </causalConnector>

                  <causalConnector id="onSelectionSetStartStop">
                        <connectorParam name="keyCode"/>
                        <connectorParam name="value"/>
                        <simpleCondition role="onSelection"
key="$keyCode"/>
                        <compoundAction operator="seq">
                              <simpleAction role="set" max="unbounded"
value="$value"/>
                              <simpleAction role="stop"/>
                              <simpleAction role="start"/>

                        </compoundAction>
                  </causalConnector>
```

```
            </connectorBase>

      </head>

      <body>

            <port id="pVideo" component="video"/>

            <media id="video" src="media/video.avi" descriptor="dVideo">
                  <property name="height"/>
                  <property name="width"/>
                  <property name="right"/>
                  <property name="top"/>
                  <area id="mostrarImagem" begin="36s" end="55s"/>
                  <area id="mostrarBotao" begin="90s" end="110s"/>
            </media>
            <media id="imagem" src="media/imagem.png" descriptor="dImagem">
            </media>
            <media id="botao" src="media/botao1.png" descriptor="dBotao">
            </media>
            <media id="lua" src="media/main.lua" descriptor="dLua">
                  <area id="corBotao" />
            </media>
            <media id="nodeSettings" type="application/x-ncl-settings">
            <property name="service.currentKeyMaster" value="lua"/>
            </media>

            <link xconnector="onBeginStart">
                  <bind role="onBegin" component="video"
interface="mostrarImagem" />
                  <bind role="start" component="imagem" />
            </link>

            <link xconnector="onEndStop">
                  <bind role="onEnd" component="video"
interface="mostrarImagem" />
                  <bind role="stop" component="imagem" />
            </link>

            <link xconnector="onBeginStart">
                  <bind role="onBegin" component="video"
interface="mostrarBotao"/>
                  <bind role="start" component="botao" />
            </link>

            <link xconnector="onEndStop">
                  <bind role="onEnd" component="video"
interface="mostrarBotao"/>
                  <bind role="stop" component="botao" />
            </link>


            <link xconnector="onSelectionSetStartStop">
                  <bind role="onSelection" component="botao" >
                        <bindParam name="keyCode" value="RED"/>
                  </bind>
                  <bind role="set" component="video" interface="height">
                        <bindParam name="value" value="30%"/>
                  </bind>
```

```
                        <bind role="set" component="video"
interface="width">
                              <bindParam name="value" value="30%"/>
                        </bind>
                        <bind role="set" component="video" interface="top">
                              <bindParam name="value" value="20"/>
                        </bind>
                        <bind role="set" component="video" interface="right">
                              <bindParam name="value" value="30"/>
                        </bind>
                        <bind role="stop" component="botao"/>
                        <bind role="start" component="lua"/>
                  </link>

      </body>
</ncl>
```

Script Lua:

```
function redraw(pergunta, alternativa1, alternativa2, alternativa3,
alternativa4)
            canvas:clear()
--Desenhando FUNDO

            local regLarg, regAlt = canvas:attrSize()
            canvas:attrFont('arial', 19)
            local img = canvas:new('media/telaLua.png')
            local fundo = { img=img, x=0, y=0, dx=regLarg, dy=regAlt }

            canvas:compose(fundo.x, fundo.y, fundo.img)

--DESENHANDO TEXTO

            local txt = pergunta
            local txtLarg, txtAlt = canvas:measureText(txt)
            local txtX = 10
            local txtY = 175
            canvas:attrColor('white')
            canvas:drawText(txtX, txtY, txt)

--DESENHANDO BOTÕES E ALTERNATIVAS

            local botaoA = canvas:new('media/botaoA.png')
            local imgLargA, imgAltA = canvas:attrSize()
            canvas:compose ((regLarg-imgLargA)+15, 250, botaoA)
            local txtA = alternativa1
            local txtLargA, txtAltA = canvas:measureText(txtA)
            canvas:drawText(100, 250, txtA)

            local botaoB = canvas:new('media/botaoB.png')
            local imgLargB, imgAltB = canvas:attrSize()
            canvas:compose ((regLarg-imgLargB)+15, 300, botaoB)
            local txtB = alternativa2
            local txtLargB, txtAltB = canvas:measureText(txtB)
            canvas:drawText(100, 300, txtB)

            local botaoC = canvas:new('media/botaoC.png')
            local imgLargC, imgAltC = canvas:attrSize()
            canvas:compose ((regLarg-imgLargC)+15, 350, botaoC)
            local txtC = alternativa3
```

```
            local txtLargC, txtAltC = canvas:measureText(txtC)
            canvas:drawText(100,350, txtC)

            local botaoD = canvas:new('media/botaoD.png')
            local imgLargD, imgAltD = canvas:attrSize()
            canvas:compose ((regLarg-imgLargD)+15, 400, botaoD)
            local txtD = alternativa4
            local txtLargD, txtAltD = canvas:measureText(txtD)
            canvas:drawText(100,400, txtD)

            canvas:flush()

end

-- DESENHA NOTA
function desenhaNota(nota)
canvas:attrFont('arial', 30)
canvas:attrColor('yellow')
local txtNota = ('Fim da Prova. Nota: ' .. nota)
local txtLargNota, txtAltNota = canvas:measureText(txtNota)
canvas:drawText(400,300, txtNota)
canvas:flush()
end

function tratador (evt)


if (numPergunta == 1) then
pergunta = 'Em qual ano chegou a primeira expedicao oficial portuguesa ao
Brasil?'
alternativa1 = '1500'
alternativa2 = '1501'
alternativa3 = '1349'
alternativa4 = '1530'
alternativaCerta = 'B'
end

if (numPergunta == 2) then

pergunta = 'Qual rei frances ficou conhecido por apoiar o contrabando de
pau Brasil?'
alternativa1 = 'Francisco I'
alternativa2 = 'Dom Pedro I'
alternativa3 = 'Henrique VIII'
alternativa4 = 'Elizabeth II'
alternativaCerta = 'A'
end

if (numPergunta == 3) then

pergunta = 'Por quem foi comandada a primeira expedicao colonizadora
enviada ao Brasil por Portugal?'
alternativa1 = 'Joao Vaz Corte Real'
alternativa2 = 'Pedro Alvares Cabral'
alternativa3 = 'Vasco da Gama'
alternativa4 = 'Martin Afonso de Souza'
alternativaCerta = 'D'
end

if (numPergunta == 5) then
      event.post {
```

```
                              class = 'ncl',
                              type = 'presentation',
                              action = 'stop'
                        }
end
redraw(pergunta, alternativa1, alternativa2, alternativa3, alternativa4)

      if evt.class == 'key' then
            if evt.type == 'press' then

                  if (evt.key == 'RED') and (alternativaCerta == 'A') then

                        nota = nota + 10
                        numPergunta = numPergunta + 1
                        return
                  elseif (evt.key == 'GREEN') and (alternativaCerta =='B')
then

                        nota = nota + 10
                        numPergunta = numPergunta + 1
                        return
                  elseif (evt.key == 'YELLOW') and (alternativaCerta =='C')
then

                        nota = nota + 10
                        numPergunta = numPergunta + 1
                        return
                  elseif (evt.key == 'BLUE') and (alternativaCerta == 'D')
then

                        nota = nota + 10
                        numPergunta = numPergunta + 1
                        return
                  else
                        nota = nota + 0
                        numPergunta = numPergunta + 1
                        return
                  end
            end
      end

if (numPergunta == 4) then
      nota = nota/3
      resultado =  string.format('%.2f', nota)
      desenhaNota(resultado)
      canvas:attrFont('arial', 19)
      canvas:attrColor('white')
      txtSair = ('Para sair pressione qualquer tecla')
      canvas:drawText(400,330, txtSair)
      canvas:flush()
      numPergunta = numPergunta + 1
end

end

-- CODIGO LUA
nota = 0
numPergunta = 1
event.register(tratador)
```