UNIVERSIDADE FEDERAL DE SANTA CATARINA

Jogos Digitais para TV Digital Brasileira

Poliane Teixeira Brito

Florianópolis – SC

2009/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Jogos Digitais para TV Digital Brasileira

Poliane Teixeira Brito

Trabalho de conclusão de curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis – SC

2009/2

Poliane Teixeira Brito

Jogos Digitais para TV Digital Brasileira

Trabalho de conclusão de curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador

_____

Prof. Frank Augusto Siqueira, Dr.

Universidade Federal de Santa Catarina
frank@inf.ufsc.br

Banca examinadora

_____

Prof. Antônio Augusto Medeiros Fröhlich, Dr.

Universidade Federal de Santa Catarina
guto@inf.ufsc.br

_____

Bruno Ghisi

brunogh@gmail.com

AGRADECIMENTOS

À Deus.

Aos meus pais pelo amor incondicional e apoio em todos os momentos.

Ao meu namorado, Daniel, pela paciência, força, apoio e conselhos.

Ao meu primo Leonardo, pela força nas horas de ansiedade.

Ao meu orientador pela motivação e apoio.

E a todos os que colaboraram com o desenvolvimento deste trabalho.

RESUMO

Sabendo-se que o impacto da transição entre TV analógica e TV Digital é muito mais do que a simples melhora na imagem e no som, desenvolveu-se este estudo tendo como base a interatividade - esta que é o grande salto da TV Digital - para jogos digitais proporcionado pelo *middleware* brasileiro Ginga.

Palavras chaves: TV Digital, Middleware Ginga, NCL, Lua, Jogos.

ABSTRACT

Knowing that the impact of the transition from analog TV and Digital TV is more than just improved picture and sound, developed this study based on interactivity - this is the big jump of Digital TV - in digital games offered by Ginga Brazilian middleware.

Key words: Digital TV, Middleware Ginga, NCL, Lua, Games.

SUMÁRIO

LISTA DE FIGURAS 10
1. Introdução 11
1.1. Contexto 11
1.2. Objetivos 11
1.2.1. Objetivo Geral 11
1.2.2. Objetivo Específico 12
1.3. Motivação 12
1.4. Estrutura do Trabalho 12
2. Televisão 14
2.1. Histórico 14
2.2. TV em Preto-e-Branco 15
2.3. Televisão em Cores 16
2.4. Televisão em Alta Definição 17
2.5. TV Digital 17
2.5.1. Modelos, sistemas e padrões da TV Digital 18
2.5.1.1. ATSC: Padrão Americano 19
2.5.1.2. DVB: Padrão Europeu 20
2.5.1.3. ISDB: Padrão Japonês 20
2.5.2. Sistema Brasileiro de TV Digital 21
3. Interatividade 22
3.1. *Middleware* Ginga 22
3.1.1. Ginga-NCL 23
3.1.2. Ginga-J 24
3.2. Linguagem Lua 24
3.2.1. Variáveis e Tipos 25
3.2.2. Operadores e Controladores de Fluxo 26
3.2.3. Funções 27
3.2.4. Tabelas e Objetos 28
3.2.5. Biblioteca Padrão 28
3.3. Nested Context Language – NCL 29
3.3.1. Extensões de NCLua 29

3.3.1.1. Programação Orientada a Eventos 30
3.3.1.2. Desenhando na Região NCL 32
3.3.1.3. Interação documento NCL – NCLua 32
4. Projeto de Jogos Digitais 34
4.1. Objetivo e Tema 36
4.2. Pesquisa e Preparação 36
4.3. Projeto 36
4.3.1. Arquitetura I/O 36
4.3.2. Arquitetura do Jogo 37
4.3.3. Arquitetura Programacional 37
4.4. Avaliação do Projeto 38
4.5. Pré-Programação 38
4.6. Programação 38
4.7. Testes 38
5. Projeto Proposto 39
5.1. Requisitos e Análise do Projeto 39
5.1.1. Introdução 40
5.1.1.1. Objetivo do Desenvolvimento 40
5.1.1.2. O Jogo 40
5.1.1.3. Definições e Abreviaturas 41
5.1.2. Visão Geral do Sistema 41
5.1.2.1. Arquitetura do Sistema 41
5.1.2.2. Arquitetura da Aplicação 41
5.1.2.3. Premissas de Desenvolvimento 42
5.1.3. Requisitos da Aplicação 42
5.1.3.1. Requisitos Funcionais 42
5.1.3.2. Requisitos de Interface 43
5.1.4. Restrições de Projeto 43
5.1.4.1. Suporte de Desenvolvimento 43
5.1.4.2. Plataforma de Execução 44
5.1.4.3. Padrões de Modularidade 44
5.1.4.4. Robustez, Integridade e Segurança 44
5.1.4.5. Performance 44

5.1.4.6. Manutenção 45
5.2. Resultados 45
6. Conclusão e Perspectivas Futuras 49
REFERÊNCIAS BIBLIOGRÁFICAS 50
APÊNDICE 52
ANEXOS 57

LISTA DE FIGURAS

Fig. 1: Curva de luminosidade relativa ao olho humano 16
Fig. 2: Sensibilidade dos cones do olho humano 16
Fig. 3: Relacionamento entre modelo, sistema e padrão de TV digital [ZUFFO,2003] 19
Fig. 4: Arquitetura do middleware Ginga [Site Oficial Middleware Ginga] 23
Fig. 5: Arquitetura do Ginga-NCL [SOARES; RODRIGUES; MORENO, 2007] 23
Fig. 6: Arquitetura das API's do Ginga-J [Site oficial Middleware Ginga] 24
Fig. 7: Paradigma de programação orientada a eventos [SANT'ANNA, 2009] 30
Fig. 8: Diagrama de Projetos de Jogos Digitais [SCHELL, 2008] 35
Fig. 9: Tela do primeiro Sokoban 40
Fig. 10: Camadas de Aplicação 42
Fig. 11: Tela Inicial do Jogo 45
Fig. 12: Tela de Ajuda do Jogo 46
Fig. 13: Tela Principal do Jogo 47

1. Introdução

1.1. Contexto

É comum nos dias de hoje ouvir-se falar sobre a “Era da Informação” ou “Era do Conhecimento”, porque a informação tornou-se requisito fundamental a todos os níveis de decisão, constituindo um elemento central no desenvolvimento da capacidade competitiva de uma empresa.

Não apenas as empresas necessitam de informações para a sua evolução, bem como as pessoas precisam para se manterem informadas dos acontecimentos, para seu crescimento pessoal e bagagem cultural.

Assim, surgiram os meios de comunicação, da necessidade de comunicação do ser humano. Existem os que são considerados individuais, como o telefone, cartas, telegramas; e os considerados em massa: rádio, televisão. Já a *Internet*, é um híbrido entre os dois.

A televisão é o meio de comunicação audiovisual mais difundido nos dias de hoje, pela sua fácil acessibilidade.

1.2. Objetivos

1.2.1. Objetivo Geral

Este trabalho tem como objetivo fazer um estudo sobre as capacidades do *middleware* do Sistema Brasileiro de Televisão Digital – *o Ginga,* em dar suporte à aplicativos multimídia interativos.

1.2.2. Objetivo Específico

O foco dos estudos será sobre o *middleware* Ginga, no seu subsistema Ginga-NCL e a linguagem de script Lua. Ao final, será elaborada uma aplicação multimídia – um jogo – baseado no *middleware* Ginga, para demonstrar parte do estudo aqui apresentado.

1.3. Motivação

A televisão digital, atualmente, se encontra em fase de implantação no Brasil. Pouco a pouco as emissoras de televisão das principais cidades brasileiras passam a oferecer a transmissão digital. A primeira cidade a oferecer sinal digital foi a cidade de São Paulo no dia 3 de dezembro de 2007. Nos anos seguintes, gradualmente, outras capitais brasileiras receberam a transmissão no padrão digital e a meta é que, até junho de 2016, esteja concluída a migração às outras cidades brasileiras.

O SBTVD proporcionará uma experiência totalmente nova aos telespectadores e às emissoras. Uma campanha publicitária, por exemplo, poderá interagir diretamente com o seu público alvo, podendo ser utilizada como uma pesquisa de mercado muito mais abrangente e com dados mais precisos.

1.4. Estrutura do Trabalho

Esta seção tem como objetivo apresentar a estrutura do trabalho trazendo uma breve descrição do que será apresentado, em cada uma das seções, e sua finalidade.

No Capítulo 1 é feita a contextualização do assunto, mostrando os objetivos do trabalho e a motivação, apontando, também, a importância dentro do ambiente organizacional.

O Capítulo 2 apresenta a história e a evolução da televisão no âmbito internacional e nacional.

O Capítulo 3 é feito o estudo do objetivo deste trabalho. Aponta-se características do *middleware* Ginga e das linguagens utilizadas para o desenvolvimento da aplicação proposta.

O Capítulo 4 relata o processo na elaboração de projetos de jogos digitais, apontando e caracterizando os principais passos do mesmo, desde o objetivo e tema até o desenvolvimento do código.

O Capítulo 5 mostra o projeto e resultados da aplicação proposta. Faz-se apontamentos do código implementado, exemplificando a teoria exposta neste trabalho.

O Capítulo 6 apresenta as conclusões sobre o trabalho elaborado e sugestões para trabalhos futuros.

O Capítulo 7 apresenta as referências bibliográficas.

2. Televisão

As transmissões por ondas eletromagnéticas (ondas de rádio) chamadas de radio transmissão possibilitaram, primeiramente, as transmissões de voz, posteriormente de imagem e, atualmente, transmissão de dados. A Televisão é um sistema eletrônico de recepção de imagens e som de forma instantânea. Funciona a partir da análise e conversão da luz e do som em ondas eletromagnéticas e de sua reconversão em um aparelho: o televisor. Esse capta as ondas eletromagnéticas e através de seus componentes internos as converte novamente em imagem e som [WIKIPEDIA, 2009].

2.1. Histórico

O primeiro sistema semi-mecânico de televisão analógica foi demonstrado em Fevereiro de 1924 em Londres, e, posteriormente, imagens em movimento em 30 de outubro de 1925. Um sistema eletrônico completo foi demonstrado por John Logie Baird, Philo Farnsworth e Philo Taylor Farnsworth em 1927. O primeiro serviço analógico foi a WGY em Schenectady, Nova Iorque, inaugurado em 11 de maio de 1928 [WIKIPEDIA, 2009].

Os primeiros aparelhos de televisão eram rádios com um dispositivo que consistia num tubo de néon com um disco giratório mecânico (disco de Nipkow) que produzia uma imagem vermelha do tamanho de um selo postal. O primeiro serviço de alta definição apareceu na Alemanha em março de 1935, mas estava disponível apenas em 22 salas públicas. Uma das primeiras grandes transmissões de televisão foi a dos Jogos Olimpícos de Berlim de 1936. O uso da televisão aumentou enormemente depois da Segunda Guerra Mundial devido aos avanços tecnológicos surgidos com as necessidades da guerra e à renda adicional disponível (televisores na década de 1930 custavam o equivalente a 7000 dólares atuais e havia pouca programação disponível) [WIKIPEDIA, 2009].

A televisão em cores surgiu em 1954, na rede norte-americana NBC. Um ano antes o governo dos Estados Unidos da América aprovou o sistema de transmissão em cores proposto pela rede CBS, mas quando a RCA apresentou um novo sistema que não exigia alterações nos aparelhos antigos em preto e branco, a CBS abandonou sua proposta em favor da nova [WIKIPEDIA, 2009].

No Brasil, em 1972 ocorre oficialmente a primeira transmissão em cores, a partir de Caxias do Sul, RS, por ocasião da Festa da Uva, em 19 de fevereiro. Em 31 de março, inaugura-se oficialmente a televisão em cores no Brasil.

2.2. TV em Preto-e-Branco

Criada pelos americanos a TV em preto-e-branco se baseia no antigo cinema. Este padrão tem algumas características, tais como: relação do aspecto imagem (largura/altura) é de 4:3, usa transmissão de 30 quadros por segundo e utiliza 525 linhas por quadro.

A TV em preto-e-branco só foi possível devido à curva de luminosidade relativa ao olho humano. A resposta visual humana está restrita a uma pequena faixa do espectro das radiações eletromagnéticas. Esta faixa do espectro está situada entre 380 e 770 nm, dependendo do observador. Assim sendo, na câmera da TV em preto-e-branco, um feixe elétrico enxerga a imagem conforme essa curva e cria um sinal conhecido pelo nome de “luminância”. No receptor, o sinal luminância superposto a um feixe eletrônico faz com que sejam reproduzidas, em uma tela luminescente, as sensações de “escuro” e “claro” que foram captadas pela câmera.

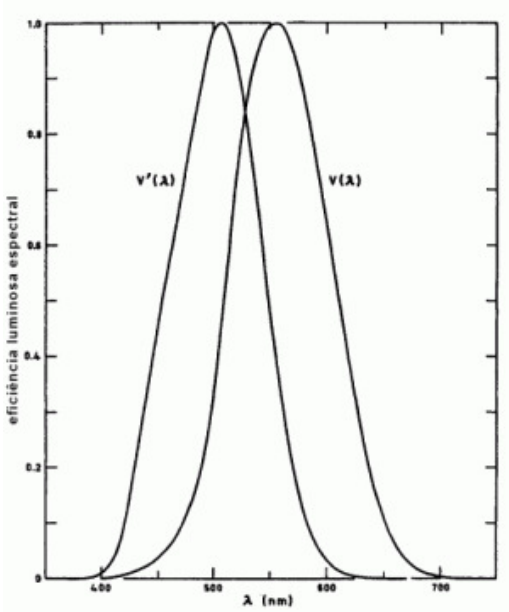

Fig. 1: Curva de luminosidade relativa ao olho humano.

2.3. Televisão em Cores

A TV em cores só foi possível devido ao fato de que o olho humano possui sensores, conhecido como cones, predominantes para as cores: vermelha, verde e azul. As outras cores se formam na combinação de porcentagem de cada cor citada.

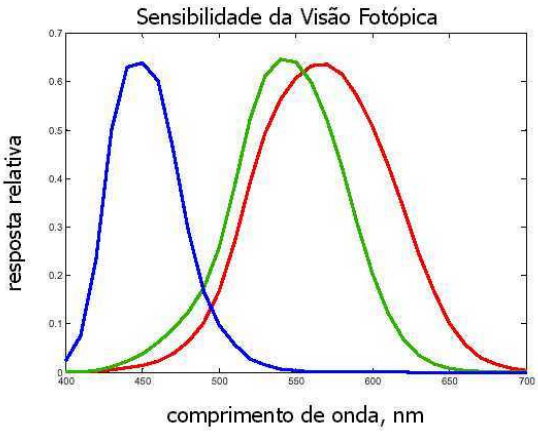

Fig. 2: Sensibilidade dos cones do olho humano.

Baseado neste princípio, a estação utiliza uma câmera tricromática com alguns filtros que analisam a imagem conforme as curvas. Criam-se assim três sinais: R (*Red* - vermelho), G (*Green* – verde) e B (*Blue* – azul). As transmissões das imagens e dos sons da televisão são feitas por ondas

eletromagnéticas, cuja freqüência é medida em Hertz. A largura da banda de transmissão é de 4 *Mhz* (megahertz), ou seja, 4 milhões de oscilações por segundo. No receptor usa-se uma tela luminescente com três feixes que, ao receberem o sinal R, G e B, excitam proporcionalmente as cores corretas e reproduzem assim a imagem original.

2.4. Televisão em Alta Definição

A evolução na direção da televisão de alta definição começou com uma pesquisa desenvolvida no Japão, que criou um sistema analógico que, praticamente, duplicava o número de linhas que passava de 525 para 1125, e permitia ainda a transmissão de som de alta qualidade.

Na década de 1980, pesquisadores japoneses apresentaram o MUSE, que foi considerado o primeiro sistema de televisão de alta definição. Somente em 1997 o Japão partiu para o desenvolvimento de um padrão totalmente digital, denominado ISDB, que entrou em operação em 2003, substituindo o padrão MUSE. Na Europa, em 1986 foi desenvolvido o padrão MAC que também combinava técnicas analógicas e digitais. As características de alta definição foram providas pelo padrão HD-MAC, operando em canais de 27 *MHz*, com transmissão via satélite.

2.5. TV Digital

A TV digital utiliza compressão digital e modulação para enviar vídeo, áudio e sinais de dados aos aparelhos compatíveis com a tecnologia, acarretando em melhores transmissões e recepções de maior quantidade de conteúdo num mesmo canal podendo atingir o alvo de alta definição. Os padrões comerciais são capazes de transportar até 19*Mbps* [WIKIPEDIA, 2009].

Quanto a imagem, utiliza-se de uma mais próxima à imagem panorâmica (formato 16:9), de alta definição, chegando a 1.080 linhas com o padrão HDTV (*High Definition Television*). Além disso, a sintonia do sinal terá a ausência de fantasmas, chuviscos e ainda sem ruídos e interferências no áudio e vídeo. No som, são seis canais estéreos de áudio.

A interatividade também está presente nesse sistema. Além do sinal digital de áudio e vídeo, há um segundo sinal digital que disponibilizará a interatividade; essa que está baseada em um *middleware* presente nos *set-top-box* – conversores para o sinal de digital; que fará a interação das informações com telespectador – emissora e vice-versa. Será possível acessar serviços, como bancos (*T-Banking*), educação (*T-Learning*), compras (*T-Commerce*), governo (*T-Gov*), saúde (*T-Health*), além de jogos, e-mails e chats.

2.5.1. Modelos, sistemas e padrões da TV Digital

Segundo ALEXANDER (1977), um padrão expressa uma relação entre um contexto, um problema e uma solução. Cada padrão descreve um problema que ocorre repetidas vezes no nosso ambiente e então descreve o básico da solução deste problema de tal modo que se possa usar esta solução milhões de vezes sem precisar trilhar o mesmo caminho até a solução novamente.

ALVAREZ (1990) diz que, sistema pode ser definido como um conjunto de elementos interdependentes que interagem com objetivos comuns formando um todo, e onde cada um dos elementos componentes comporta-se, por sua vez, como um sistema cujo resultado é maior do que o resultado que as unidades poderiam ter se funcionassem independentemente. Qualquer conjunto de partes unidas entre si pode ser considerado um sistema, desde que as relações entre as partes e o comportamento do todo sejam o foco de atenção. Na TV Digital envolve o padrão de transmissão, *middleware*, os serviços a serem oferecidos e o tipo de interatividade [MARTINS *apud* BECKER e MONTEZ, 2004].

Modelo é a maneira em que se organiza o meio, a organização das relações entre os diversos atores, e incluem as demandas econômicas, sociais, culturais, políticas, governamentais e tecnológicas [ALMAS, 2005]. É o modelo que define as condições para o estabelecimento do sistema e as questões referentes ao padrão tecnológico.

Sendo assim, o padrão da TV Digital define-se como o conjunto de definições e especificações técnicas necessárias para a correta implementação e implantação do sistema a partir do modelo proposto [ZUFFO, 2003].

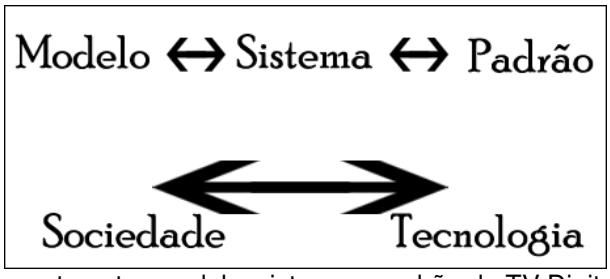

Fig. 3: Relacionamento entre modelo, sistema e padrão de TV Digital [ZUFFO, 2003]

Atualmente, existem três padrões mundiais principais de TV Digital, que surgiram com o intuito de padronizar as especificações da mesma. São eles: ATSC (Padrão Americano), DVB (Padrão Europeu) e ISDB (Padrão Japonês).

2.5.1.1. ATSC: Padrão Americano

ATSC (*Advanced Television Systems Committee*): adotado nos Estados Unidos, Canadá, México e Coréia do Sul, produz imagens no formato 16:9 (*wide screen*) e com até 1920×1080 pixels – seis vezes mais que o padrão analógico que sucedeu o *NTSC*. Permite transmitir até seis canais virtuais em definição padrão e oferece qualidade de som similar ao dos *Home Theaters*, por meio do sistema *Dolby Digital*, que utiliza seis canais de áudio. O consórcio existe desde 1982, mas o padrão só entrou em funcionamento comercial nos Estados Unidos em 1998. É considerado o mais robusto, ideal para

transmissão em alta-definição, mas é o menos desenvolvido no quesito mobilidade [MOREIRA, Daniela. 2006].

2.5.1.2. DVB: Padrão Europeu

DVB (*Digital Video Broadcasting*): adotado comercialmente em 1998, pelo Reino Unido, o padrão também foi abraçado por Índia, Austrália e Nova Zelândia. O consórcio responsável pela sua definição reúne mais de 270 empresas. Possui padrões para transmissão terrestre (*DVB-T*), por cabo (*DVB-C*) e satélite (*DVB-S*). É conhecido por ser mais versátil, facilitando a transmissão de múltiplos canais virtuais na mesma freqüência. Opera na freqüência de 8 MHz, fator que o deixa em desvantagem em relação ao japonês e ao americano, que operam em 6 MHz, mesmo espectro usado no Brasil para a TV aberta [MOREIRA, Daniela. 2006].

2.5.1.3. ISDB: Padrão Japonês

ISDB (*Integrated Service Digital Broadcasting*): vem sendo desenvolvido desde a década de 70. Inicialmente, o ISDB substituiu o antigo MUSE (*Multiple Sub-Nyquist Sampling Encoding*), um sistema analógico de televisão de alta definição, com modo de transmissão era via satélite. Já em 2003, os primeiros receptores para televisão digital terrestre começaram a ser comercializados, expandindo assim a TV digital no território japonês.

O padrão ISDB é formado por um conjunto de documentos que definem as medidas adotadas em relação ao meio de transmissão, transporte, codificação e *middleware,* camada de comunicação entre o software e hardware.

Do ponto de vista de tecnologia e desempenho, o padrão japonês pode ser considerado o mais avançado, pois teve a mobilidade e flexibilidade como principal pré-requisito durante o seu desenvolvimento, sendo assim adequado

para recepção portátil de dados e imagens. Além deste fato, este padrão tem uma intensa convergência, suporta modulação digital de alta qualidade e ainda engloba os conceitos de televisão de alta definição [ISDB,1998].

2.5.2. Sistema Brasileiro de TV Digital

O Brasil tem como padrão o ISDB-TB, tendo como base o padrão ISDB japonês, acrescentando tecnologias desenvolvidas por universidades brasileiras.

O SBTVD tem como foco a inclusão digital, oferecendo serviços que hoje só estão disponíveis na *Internet* na televisão e ensinar os telespectadores a utilização dos mesmos.

O *Laboratório de Sistemas Integráveis da Escola Politécnica da USP* desenvolve o receptor responsável pela interatividade a melhor qualidade do som e imagem, baseado numa placa *Intel* com decodificação MPEG-2.

Quanto à modulação, ela é baseada na tecnologia BST-OFDM, com banda segmentada. O sistema permite a transmissão de múltiplos *streams*, como vídeo em alta e baixa definição.

O *middleware* adotado pelo SBTVD é o Ginga; é a base para o desenvolvimento de aplicações interativas para a TV Digital Brasileira. É o aplicativo que interliga outros dois aplicativos que normalmente encontram-se em camadas diferentes, neste caso, a camada de baixo nível: o hardware (*set top box*); e a camada de alto nível: o software.

O canal de retorno é o responsável pela interação telespectador – emissora. Utiliza-se, para o desenvolvimento do canal de retorno, tecnologias intra-banda, onde o próprio canal de TV é usado para transmitir as informações do telespectador para a emissora. Assemelha-se à tecnologia de telefonia celular.

3. Interatividade

Assim como a Internet, a TV digital interativa representa a possibilidade de acesso a um mundo virtual de informações e serviços.

A TV digital tem a capacidade de executar aplicações com interfaces ricas oferecendo ao usuário possibilidades tão extensas quanto as que possuímos hoje em dia nos computadores pessoais e distribuídas pela internet.

O impacto da transição entre TV analógica e TV digital, é muito mais do que a simples melhora na imagem e do som.

O ponto fraco da TV digital, na qual os computadores com acesso à internet têm vantagem, é justamente o canal de retorno. Na especificação atual, os usuários não podem interagir diretamente com os canais de televisão, com outros usuários ou com aplicações remotas. Acarreta em uma limitação bastante pesada, visto que nos dias de hoje a internet esta onipresente no cotidiano de todas as pessoas, mesmo que de forma indireta.

3.1. *Middleware* Ginga

Por definição, m*iddleware* é uma camada de *software* situada entre uma camada de baixo nível - em termos computacionais (usualmente Sistemas Operacionais) - e uma camada de mais alto nível como um framework de desenvolvimento de aplicações. Além disso, um *middleware* é o *software* responsável pela execução de um mesmo programa em várias plataformas.

O *middleware* Ginga é subdividido em dois subsistemas principais interligados, que permitem o desenvolvimento de aplicações seguindo dois paradigmas de programação diferentes. Dependendo das funcionalidades requeridas no projeto de cada aplicação, um paradigma será mais adequado que o outro. Esses dois subsistemas são chamados de Ginga-J (para

aplicações imperativas/procedurais Java) e Ginga-NCL (para aplicações declarativas NCL).

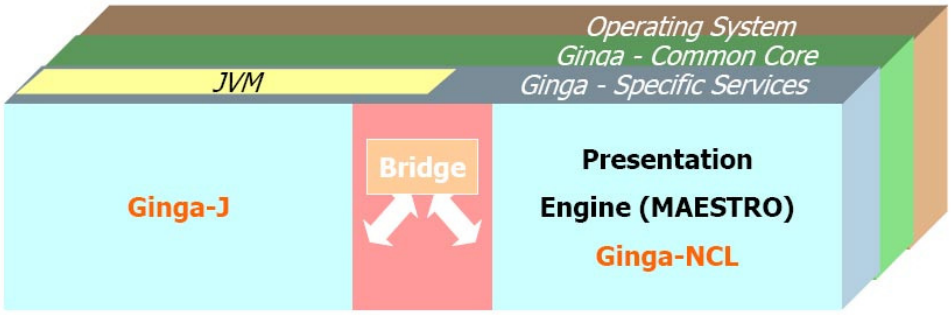

Fig. 4: Arquitetura do middleware Ginga. [Site Oficial Middleware Ginga]

3.1.1. Ginga-NCL

O ambiente declarativo Ginga-NCL, utiliza a linguagem NCL (*Nested Context Language*) como ferramenta para elaboração de aplicações declarativas. A linguagem define um formato *XML* (*eXtensible Markup Language*) que faz a sincronia entre mídias e oferece, também, a integração com objetos que possuem conteúdo procedural; sejam *Xlets* (escrito na linguagem *JAVA* similares aos *applets*) ou NCLua (escritos na linguagem Lua).

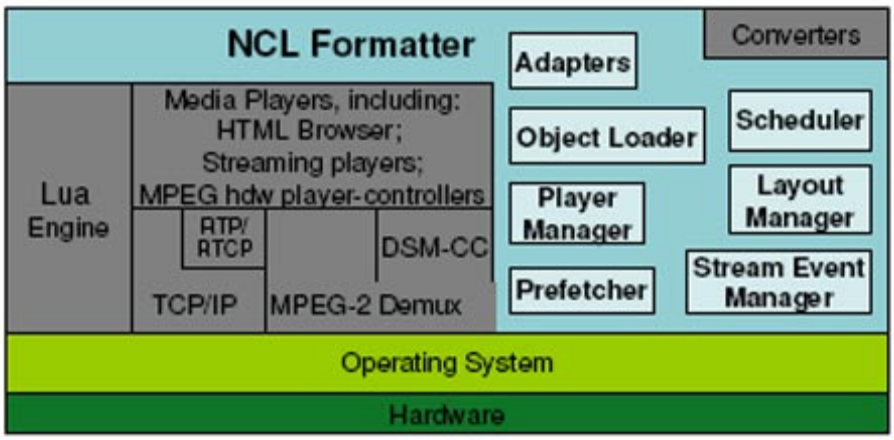

Fig. 5: Arquitetura do Ginga-NCL [SOARES; RODRIGUES; MORENO, 2007]

3.1.2. Ginga-J

O ambiente procedural Ginga-J oferece suporte a aplicações desenvolvidas usando a linguagem Java. Ginga-J é subdividido em três conjuntos chamados de *API*'s verdes, amarelas e vermelhas. As *API*'s verdes são responsáveis pela compatibilidade com outros padrões – os sistemas americano e europeu. Nas amarelas estão inclusas a *API JMF 2.1* (*Java Media Framework*) necessária para aplicações avançadas como captura de som e vídeo. Elas são compatíveis com outros padrões através de um adaptador. Já a vermelha, é exclusiva do SBTVD e possui um caráter inovador; oferece uma facilidade na integração de outros dispositivos de entrada e saída ao *set-top box* e a ponte NCL (NCL *Bridge*) que permite a integração de conteúdo procedural e declarativo na mesma aplicação.

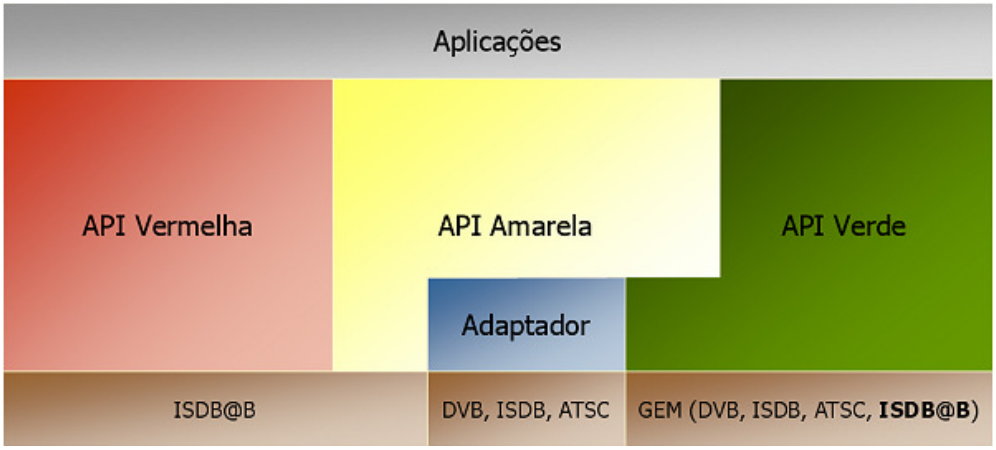

Fig. 6: Arquitetura das *API*'s do Ginga-J [Site Oficial Middleware Ginga]

3.2. Linguagem Lua

Lua é uma linguagem de script projetada e implementada em 1993, pelo Grupo de Computação Gráfica da PUC-Rio, na necessidade crescente das suas aplicações serem configuráveis externamente pelos usuários. Com isso, conseguiu-se fazer com que alterações simples – como alterações de cores - pudessem serem feitas sem a recompilação da aplicação. Poderosa e leve, a linguagem Lua funciona acoplada a programas hospedeiros implementados em linguagens compiladas.

Atualmente, Lua é a linguagem de script mais utilizada no desenvolvimento de jogos, onde podemos encontrar grandes empresas como LucasArts, Blizzard, Eletronic Arts e Microsoft, respectivamente com os jogos Grim Fandango, World of Warcraft, Crysis e Impossible Creatures.

A seguir apresentaremos uma breve introdução às principais características da linguagem.

3.2.1. Variáveis e Tipos

Não existem tipos associados às variáveis. Assim, uma variável pode assumir momento um tipo, momento outro. O tipo não está ligado à variável, e sim ao valor.

```
-- armazenando string em y
y = "chove"
-- armazenando número em x
x = 3
--armazenando número em y
y = 4
-- armazenando nil em x
x = nil
```

Variáveis globais não são declaradas. Ao dizermos `y = "chove"`, a variável por *default* é global; caso queira uma variável local, basta apenas adicionar `local` na frente.

Os valores podem ser:

- *nil*: indica ausência de valor, também é interpretado como falso numa expressão booleana;

```
-- atribuindo nil à variável x
x=nil
```

- *boolean*: valor booleano, admitindo valores `false` e `true`;

```
-- atribuindo false à variável x
x=false
```

- *number*: valor numérico, não havendo diferença entre números inteiros e reais;

```
-- atribuindo 2 à variável x
x=2
-- atribuindo 2.5 à variável x
x=2.5
```

- *string*: valor cadeia de caracteres. Utiliza-se aspas duplas (“...”), aspas simples (‘...’) ou duplo colchetes (`[[...]]`);

```
-- atribuindo “cavalo” à variável x
x=’cavalo’
```

- *table*: vetor associativo;

```
-- a variável x como table
x={‘cavalo’, ‘ovelha’, ‘boi’}
```

- *function*: uma função.

```
-- atribuindo uma função
fatorial = function (n) ... end
```

3.2.2. Operadores e Controladores de Fluxo

A linguagem utiliza os seguintes operadores aritméticos: `+` (adição), `-` (subtração), `*` (multiplicação), `/` (divisão), `^` (exponenciação) e – unário (negação).

Já os operadores relacionais são: < (menor que), > (maior que), <= (menor ou igual que), >= (maior ou igual que), == (igualdade), ~= (diferença). Os operadores lógicos são os usuais: `and` (e), `or` (ou) e `not` (negação).

Quanto aos controladores de fluxo, os principais estão presentes na linguagem.

```
if expr then   if expr then      if expr1 then
   ...              ...              ...
end               else             elseif expr2 then
                                ...              ...
                        end              else
                                         ...
                                         end
```

Execuções iterativas com testes no início ou no fim:

```
-- inicio              --fim
while expr do          repeat
      ...                    ...
end                    until expr
```

E ainda o `for` numérico:

```
for var=expr_inicial, expr_final, expr_incremento do
      ...
end
```

Os laços `while`, `repeat` e `for` podem ser interrompidos usando o comando `break`.

3.2.3. Funções

Como qualquer outro valor, uma função em Lua, pode ser criada, armazenada em uma variável ou campo de tabela e passada como parâmetro ou valor de retorno de outra função.

```
      -- recebendo parâmetro        -- retornando valor
      function fat (parametro)      function fat ()
            ...                           i = 0
      end                                 ...
return i
                                    end
```

As funções em Lua, podem não possuírem nomes, tornando-se anônimas. Assim atribuímos as mesmas a uma variável global.

```
-- função anônima atribuída à uma variável global
fatorial = function (y) ... end
-- chamando função
x=2
print(fatorial(x))
```

3.2.4. Tabelas e Objetos

O tipo `table` representa um vetor associativo. As tabelas são as únicas formas de estruturação de dados em Lua, utilizando uma combinação de *array* e *hash*.

```
      table = {}              -- cria nova tabela
table[1]=4              -- armazena 4 no índice 1
table[2]="cavalo"       -- armazena "cavalo" no índice 2
table["cavalo"]=5       -- armazena 5 no índice "cavalo"
table[table[2]]=0       -- armazena 0 no índice "cavalo"
```

Quando o índice é uma string simples, o Lua permite que a atribuição `table[“cavalo”]=5` possa ser escrita simplesmente por `table.cavalo=5`.

3.2.5. Biblioteca Padrão

Além das funções que pertencem à biblioteca padrão, a distribuição oficial do Lua vem com bibliotecas que implementam várias funções importantes na composição de um programa.

As bibliotecas adicionais são:

- *string*: oferece funções para a manipulação de *strings*;
- *table*: funções para manipulação de tabelas, como inserção, remoção e ordenação de elementos;

- *math*: funções semelhantes à biblioteca matemática de C, como `math.sqtr`, `math.sin`, `math.log`;
- *os:* funções relacionadas ao sistema operacional;
- *debug:* oferece funções para depuração de códigos em Lua.

3.3. Nested Context Language – NCL

O NCL provê facilidades para a especificação de aspectos de interatividade, sincronismo espaço temporal entre objetos de mídia adaptabilidade e suporte a múltiplos dispositivos. Diferente do HTML (*HyperText Markup Language*) e do XML (*eXtensible Markup Language*), a linguagem NCL não mistura a definição do conteúdo de um documento com sua estruturação, oferecendo um controle não invasivo, tanto do *layout* do documento, quanto da sua apresentação temporal. [Portal do Software Público Brasileiro]

Segundo Souza (2008), o uso da linguagem NCL, não é suficiente para o desenvolvimento de aplicações interativas mais exigentes, pois esses aplicativos exigem um controle de fluxo em tempo de execução. Para tal, estendeu-se a linguagem Lua com novas funcionalidades.

3.3.1. Extensões de NCLua

Para se integrar à NCL, estendeu-se a linguagem Lua, para atender exigências do *middleware* Ginga, tal como responder às teclas do controle remoto ou desenhar dentro da região NCL.

Os seguintes módulos estão disponíveis para scripts NCLua:

- Módulo *event*: permite que os objetos NCLua se comuniquem com o documento NCL e outras entidades externas, como o controle remoto e o canal de interatividade;
- Módulo *canvas*: oferece funcionalidades para desenhar objetos gráficos na região do NCLUA;
- Módulo *settings*: oferece acesso às variáveis definidas no objeto `settings` do documento NCL (objeto do tipo "`application/x-ncl-settings`");
- Módulo *persistent*: exporta uma tabela com variáveis persistentes entre execuções de objetos imperativos.

3.3.1.1. Programação Orientada a Eventos

Objetos NCLua tratam os mecanismos de integração com o documento NCL, através do paradigma de programação orientada a eventos.

O módulo `event` do NCLua trata as interações externas à aplicação, como a dada pelo controle remoto e/ou canal de interatividade.

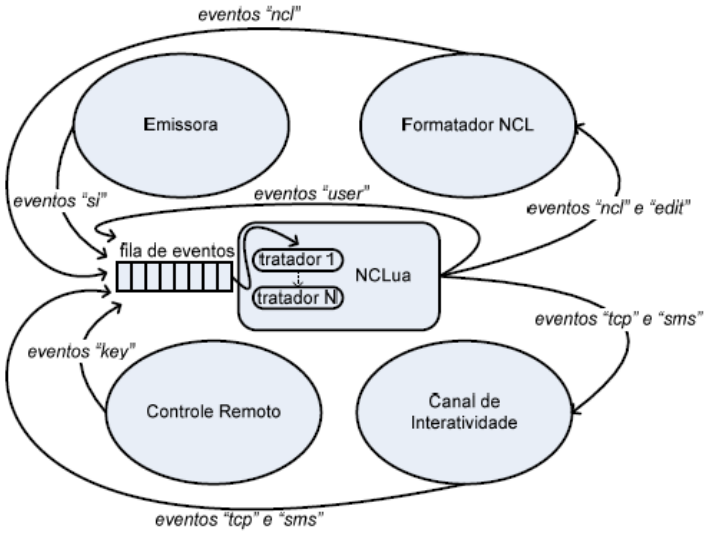

Fig. 7: Paradigma de programação orientada a eventos [SANT'ANNA, 2009]

Para que o NCLua receba os eventos externos gerados, basta registrar pelo menos uma função de tratamento de eventos, a qual possui uma estrutura comum aos scripts:

```
...
function tratador (evento)
      ...
end
event.register(tratador)
```

Os eventos são representado por uma `table` Lua:

```
evento = {
      class='key',
      type='press',
      key='RED'
}
```

Onde *class,* podem adquirir os valores: *ncl* (usada na comunicação entra NCLua-NCL); *key* (pressionamento das teclas do controle remoto); *tcp* (acesso ao canal de interatividade utilizando o protocolo *tcp*); *sms* (envio e recebimento de SMS); *edit* (comandos de edição ao vivo); *si* (acesso a um conjunto de informações multiplexadas) e *user* (possibilidade da aplicação estender sua funcionalidade e criar seus próprios eventos). [SANT'ANNA, 2009]

Há a função `event.post` onde é utilizada para que o NCLua comunique o seu estado ao documento NCL ou possa enviar dados ao canal de interatividade.

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

Onde `type` pode assumir os valores: `presentation` (estão associados à apresentação de âncoras de conteúdo, sendo identificadas pelo campo `label` do evento) e `attribution` (associados às propriedades do objeto NCLua que são identificados pelo campo `name`). [SANT'ANNA, 2009]

3.3.1.2. Desenhando na Região NCL

Um NCLua tem a possibilidade de fazer operações gráficas durante a apresentação de uma aplicação. Quando um NCLua é iniciado, automaticamente é instanciado um objeto gráfico que é atribuido à variável global canvas. Este objeto aponta para a região associada ao nó de mídia NCLua no documento NCL e é através dele que todas as operações gráficas são feitas. [SANT'ANNA, 2009]

```
<!-- Região NCL -->
<region id="luaRegion" width="300" height="100" top="200" left="20"/>
```

Existem diversas operações gráficas suportadas, tais como desenho de linhas, textos e imagens.

Segue um exemplo simples para ilustrar o uso do módulo canvas:

```
width, height = canvas:attrSize()   -- pega as dimensões da região
canvas:drawLine(0,0, width,height)  -- desenha uma linha
img = canvas:new('image.png')       -- carrega a 'image.png'
canvas:compose(100, 100, img)       -- desenha a imagem na posição
canvas:flush()                      -- atualiza a região NCL
```

3.3.1.3. Interação documento NCL – NCLua

O NCLua interage com o documento NCL através de elos. Os conectores ou *XConnectors* definem como os elos são ativados e o que eles disparam. Por exemplo: através de conectores, é possível fazer com que a exibição de uma mídia comece simultaneamente a outra, bem como termine simultaneamente à outra e diversas outras ações.

Normalmente, os conectores estão definidos em um arquivo externo ao código. Se preferir, você também pode criar os seus próprios conectores na base de conectores, entretanto é recomendado o uso do arquivo externo já que ele contém dezenas de conectores prontos para o uso, diminuindo-se, assim, o trabalho.

```
<!-- Exemplo de xconnector -- >
<link xconnector="onBeginStart">
      <bind role="onBegin" component="videoId"/>
      <bind role="start" component="luaId"/>
</link>
```

4. Projeto de Jogos Digitais

Primeiramente, definiremos jogo como sendo uma atividade lúdica composta por uma série de ações e decisões, limitado por regras e pelo universo do jogo, que resultam em uma condição final. As regras e o universo do jogo são apresentados por meios eletrônicos e controlados por uma programa digital. Existem para proporcionar uma estrutura e um contexto para as ações de um jogador. As regras, também, são para criar situações interessantes com o objetivo de desafiar e se contrapor ao jogador. As ações do jogador, suas decisões, escolhas e oportunidades, na verdade, sua jornada, tudo isso compõe a "alma do *game*". A riqueza do contexto, o desafio, a emoção e a diversão da jornada de um jogador, e não simplesmente a obtenção da condição final, é que determinam o sucesso do jogo. [SCHUYTEMA, 2008].

Projeto de jogos é essencialmente um processo artístico, mas também é um processo técnico. O projetista (*game designer*) possui grandes objetivos artísticos, mesmo quando dissolvidos em montanhas de códigos. Durante o processo de desenvolvimento do jogo, habita-se dois mundos muito diferentes: o mundo artístico e o mundo técnico [CRAWFORD, 1984].

O projeto de um jogo é a planta baixa do jogo. O projetista é a pessoa designada para criar a planta baixa e, a partir dela, com a combinação adequada de talento e esforço, surgirá um jogo. [SCHUYTEMA 2008]

Não se pode resumir o projeto de um jogo em apenas um procedimento formal, pois se trata de uma atividade muito complexa. Além disso, a personalidade do criador deverá ditar as regras a serem utilizadas, sem levar em conta que procedimentos podem bloquear a criatividade do mesmo.

Um jogo deve ter um objetivo claramente definido, que deve ser expresso em termos do efeito que ele terá sobre o jogador. Isso não afirma que o jogo será emocionante, agradável ou bom, a meta deve estabelecer as

fantasias nas quais o jogo irá se apoiar e quais os tipos de emoção ele pretende gerar.

Assim que a meta for resolvida, o tema deve ser escolhido. O tema é o meio de expressar a meta, o ambiente em que o jogo será jogado [CRAWFORD, 1984].

A figura 8, segundo SCHELL (2008), mostra e exemplifica o processo de projeto de jogos digitais.

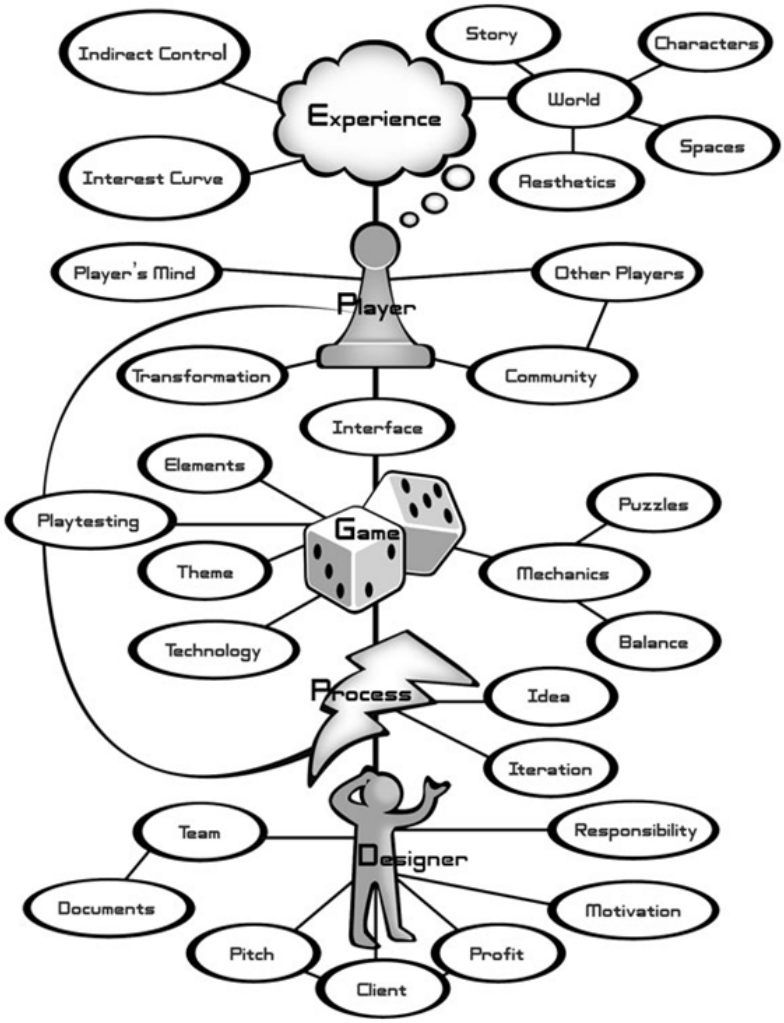

Fig. 8: Diagrama de Projetos de Jogos Digitais [SCHELL, 2008]

4.1. Objetivo e Tema

Um dos principais itens a serem relevados para dar uma intenção e objetivos claros na concepção do jogo.

Um jogo deve ter um objetivo claramente definido, que deve ser expresso em termos do efeito que ele terá sobre o jogador. Isso não afirma que o jogo será emocionante, agradável ou bom, a meta deve estabelecer as fantasias que o jogo irá apoiar e os quais os tipos de emoção que irá gerar.

4.2. Pesquisa e Preparação

Com objetivos e tema concretos, deve-se, então, começar a pesquisa sobre os assuntos que o jogo conterá. A mecânica do jogo terá que ser bem compreendida pelos envolvidos no desenvolvimento e as primeiras linhas de códigos surgem nesse período.

4.3. Projeto

Com os ideais do jogo claros, as arquiteturas que sustentarão o mesmo, devem começar a serem analisadas e/ou projetadas e desenvolvidas para que o projeto saia do papel e comece a tornar-se algo mais concreto.

4.3.1. Arquitetura I/O

Arquitetura I/O (Input/Output – Entrada/Saída) é o meio que os dados serão disponibilizados e enviados na interação jogo-jogador.

A entrada é o contato do jogador com o jogo, geralmente feito através de dispositivos externos, como o teclado, mouse e o joystick. A partir do estilo pré-estabelecido do projeto, os dispositivos de entrada devem ser bem escolhidos

e qual a interação deles no jogo, para que não haja a frustração posterior por parte dos jogadores por uma má escolha da estrutura. No caso de uma televisão, o principal dispositivo de I/O com o qual um usuário interage é o controle remoto.

Como um controle remoto não possui uma ergonomia adequada para um jogo típico de vídeo-games ou jogos que requerem mais ação por parte do jogador, o jogo proposto será focado em uma interatividade mais simples, baseado na usabilidade de jogos para celular, o qual apresenta uma interface de entrada/saída semelhante a um controle remoto.

Dispositivos de saída disponibilizarão os gráficos e os sons emitidos. Gráficos estão lá por um motivo: para comunicar. Usar gráficos para comunicar ao usuário energicamente e com sentimento e por nenhum outro motivo [CRAWFORD, 1984].

4.3.2. Arquitetura do Jogo

A idéia inicial para o desenvolvimento do jogo era a de usar uma arquitetura cliente/servidor. Esta idéia foi abandonada pela especificação do canal de retorno não ser compatível com os requisitos da aplicação.

4.3.3. Arquitetura Programacional

Este é quem traduz a arquitetura I/O para a arquitetura do Jogo. Geralmente quem assume esse papel, é um sistema operacional ou ainda um *middleware.* Aqui os fluxos de memórias devem ser levados em consideração para um bom um desempenho do jogo.

4.4. Avaliação do Projeto

A três arquiteturas já comentadas, devem ser analisadas num conjunto com os objetivos, levando em consideração a estabilidade do todo. A concepção global deverá estar dentro das expectativas iniciais e do orçamento.

A decisão se o projeto seguirá ou será interrompido é tomado nesta fase.

4.5. Pré-Programação

A pré-programação consiste na documentação do projeto. Toda a arquitetura analisada até agora deve estar incluída. A linguagem deve ser adequada ao nível de jogador, não ao nível técnico.

4.6. Programação

O projeto será executado. Deixará de ser apenas documentação e começará a ser desenvolvido em linhas de código.

4.7. Testes

Os testes serão feitos quase que exclusivamente nos emuladores disponíveis. Quando a aplicação estiver concluída ou em fase de RC *(release candidate)*, poderá ser testada e executada em equipamentos reais disponibilizados para esta empreitada.

5. Projeto Proposto

Será implementado um jogo sob o *middleware* Ginga-NCL com as linguagens NCL e Lua para as partes declarativas e lógicas respectivamente.

Como ferramentas serão utilizadas a IDE Eclipse com o *plugin* da Linguagem NCL e o *plugin* para a Linguagem Lua (utilizando a *engine* Lua na sua versão 5.1.4; e o Ginga-NCL *set-top box* emulado através do programa *VMWare*, utilizado para eventuais testes.

O jogo será um complemento de uma campanha publicitária de um produto de algum anunciante da TV Digital.

5.1. Requisitos e Análise do Projeto

Projeto: Sokopoly
Aplicação: Jogo Sokoban (*Warehouse Keeper*) para a TV Digital
Requisitos do Software: Desenvolvimento na Linguagem de Programação LUA e a Linguagem de Hipermídia NCL para o Middleware Ginga-NCL. O software será executado a partir de uma publicidade na TV Digital com intuito promocional. Possuirá *interface* gráfica e a realização da partida será apenas para um jogador.

Identificador do Documento: analiseRequisitos02
Versão: 1.6
Data: 09/10/2009

5.1.1. Introdução

5.1.1.1. Objetivo do Desenvolvimento

Este projeto foi desenvolvido com o intuito de aplicar os conhecimentos adquiridos no desenvolver do TCC (Trabalho de Conclusão de Curso). O Sokopoly será um jogo para a TV Digital onde o telespectador poderá executá-lo através de uma publicidade na mesma.

O *software* será utilizado pelas empresas com uma visão promocional do produto que estará anunciando na propaganda. Uma maneira diferente de *marketing* que pode atingir um público de todas as idades ou uma determinada faixa etária.

5.1.1.2. O Jogo

Sokoban é um clássico dos jogos de quebra-cabeças (*puzzle*) de origem japonesa desenvolvido por Hiroyuki Imabayashi em 1980. Foi publicado pela primeira vez em 1982 pela empresa *Thinking Rabbit* situada em Takarazuka no Japão.

Tendo como objetivo empurrar as *caixas (boxes)* distribuídas no tabuleiro, até o local determinado para as mesmas, sem que o personagem e/ou as caixas fiquem presos.

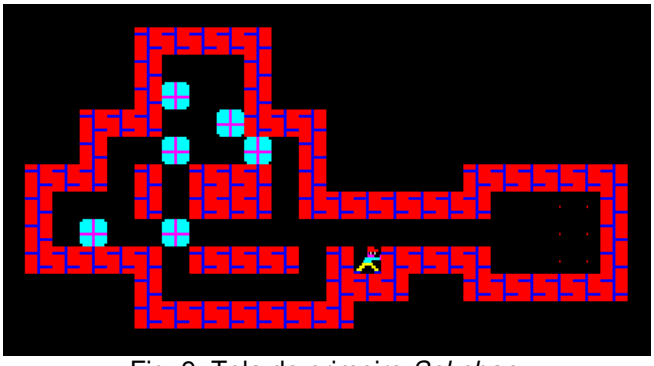

Fig. 9: Tela do primeiro *Sokoban.*

5.1.1.3. Definições e Abreviaturas

Para possibilitar um completo entendimento das descrições aqui feitas, definimos agora alguns dos elementos e termos utilizados no decorrer deste projeto.

Tabuleiro: é a tela onde as caixas e o *player* serão disponibilizados para a resolução do *puzzle*;

Puzzle: é o nível, quebra-cabeça o qual se está solucionando;

Box: ou caixa - é o objeto em que o *player* deverá empurrar até o ponto determinado;

Player: objeto o qual usuário terá controle.

5.1.2. Visão Geral do Sistema

5.1.2.1. Arquitetura do Sistema

A arquitetura consiste em *software* único, dotado de uma *interface* gráfica, a qual possui todas as funções com os quais o usuário deverá interagir para que o jogo transcorra normalmente.

5.1.2.2. Arquitetura da Aplicação

A aplicação estará rodando sob o *Middleware* Ginga no subsistema Ginga-NCL, o qual se comunicará com a camada do Sistema Operacional. Será desenvolvida seguindo o paradigma de orientação a eventos.

Segue a representação gráfica da aplicação, em relação às camadas existentes:

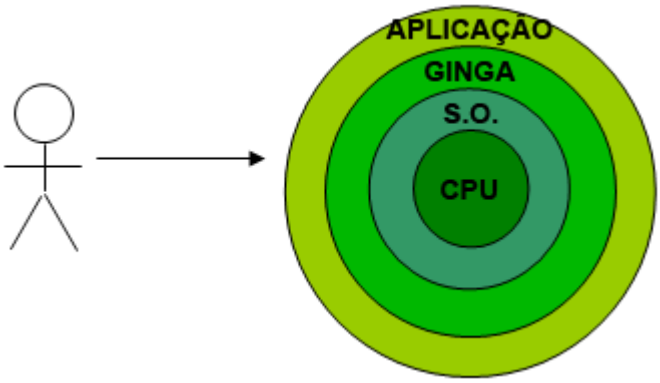

Fig. 10: Camadas de Aplicação

5.1.2.3. Premissas de Desenvolvimento

Ao usuário, será apresentada uma interface que o permitirá jogar. Uma vez escolhida a opção de “Início”, o tabuleiro será gerado e o usuário poderá começar a jogar.

5.1.3. Requisitos da Aplicação

5.1.3.1. Requisitos Funcionais

Requisito 5.1.3.1.001: Apenas uma caixa deve ser empurrada por vez;

Requisito 5.1.3.1.002: As caixas não podem ser passadas por cima e/ou puladas;

Requisito 5.1.3.1.003: As caixas não podem ser puxadas;

Requisito 5.1.3.1.004: A aplicação deve oferecer instruções de controle sobre o jogo. Estas informações devem ser acessadas através da opção ajuda no menu principal;

Requisito 5.1.3.1.005: O *player* é movimentado através das teclas direcionais do controle remoto;

Requisito 5.1.3.1.006: O jogo acaba quando todas as caixas chegarem ao destino final ou quando não há mais movimentos possíveis (caixas e/ou *player* presos).

5.1.3.2. Requisitos de Interface

Requisito 5.1.3.2.001: Após exibição do vídeo publicitário, o usuário deve ser questionado se deseja executar a aplicação (jogo) ou não – parte de desenvolvimento não cabível a este projeto;

Requisito 5.1.3.2.002: Selecionada a opção “Ajuda”, a aplicação mostra uma nova tela com instruções de comando para o jogo;

Requisito 5.1.3.2.003: Em todas as telas deverá ter uma opção indicando a possibilidade de encerrar a aplicação.

5.1.4. Restrições de Projeto

5.1.4.1. Suporte de Desenvolvimento

O código seguirá o Paradigma da Orientação a Eventos onde será utilizada a Linguagem de Programação Lua e a Linguagem Hipermídia NCL, através do ambiente IDE Eclipse.

5.1.4.2. Plataforma de Execução

A interpretação do *software* se dará pelo *Middleware* Ginga, sendo totalmente dependente do mesmo e independente da plataforma de execução do Ginga-NCL.

5.1.4.3. Padrões de Modularidade

O programa será concebido em apenas um módulo, pois não apresenta um nível avançado de complexidade que exija este tipo de fracionamento.

5.1.4.4. Robustez, Integridade e Segurança

O *software* não oferece ameaças ao sistema operacional em que estará instalado, visto que não faz uso de recursos externos ao próprio código. Não havendo acesso a disco, rede, *Internet* ou diretamente a componentes de *hardware*, o projeto pode abster-se de prever especificações de segurança ou robustez.

5.1.4.5. Performance

Os requisitos gráficos ou lógicos do projeto não são significantes frente aos equipamentos capazes de executar o *Middleware* Ginga-NCL. Sendo assim, restringe-se os requisitos de performance aos próprios do *Middleware* Ginga-NCL.

5.1.4.6. Manutenção

Visto a simplicidade do aplicativo, os diagramas *UML* propostos neste projeto encarregam-se da competência de fornecer ao desenvolvedor responsável as informações necessárias para modificações ou ajustes no código do *software*.

5.2. Resultados

Desenvolveu-se, então, uma aplicação com *marketing* voltado para o produto *Monopoly* (jogo de tabuleiro da empresa Hasbro).

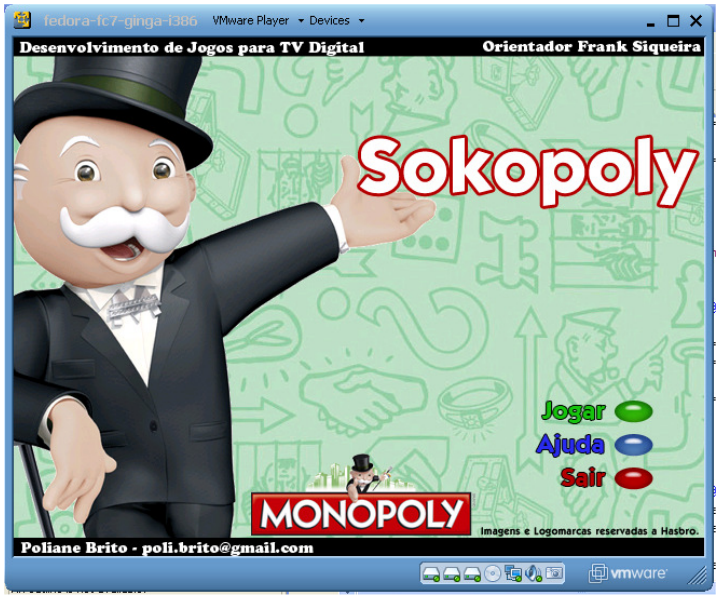

Fig. 11: Tela Inicial do Jogo

Podemos observar na Figura 10, a tela inicial do jogo implementado rodando sob o *VMWare Player*. As opções Jogar, Ajuda e Sair, são tratadas através dos eventos NCL e executados a partir das teclas do controle remoto.

A seguir parte do código responsável pela transição de telas, onde é trata dos fluxos gerados pelas teclas coloridas do controle remoto:

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

Um xconnector que utiliza o código mostrado acima, é o que mostra a tela de ajuda, cujo código segue:

```
      <link xconnector="onKeySelectionStopNStartN">
            <linkParam name="keyCode" value="BLUE" />
            <bind component="inicio" role="onSelection" />
            <bind role="stop" component="inicio"/>
            <bind role="start" component="ajuda"/>
      </link>
```

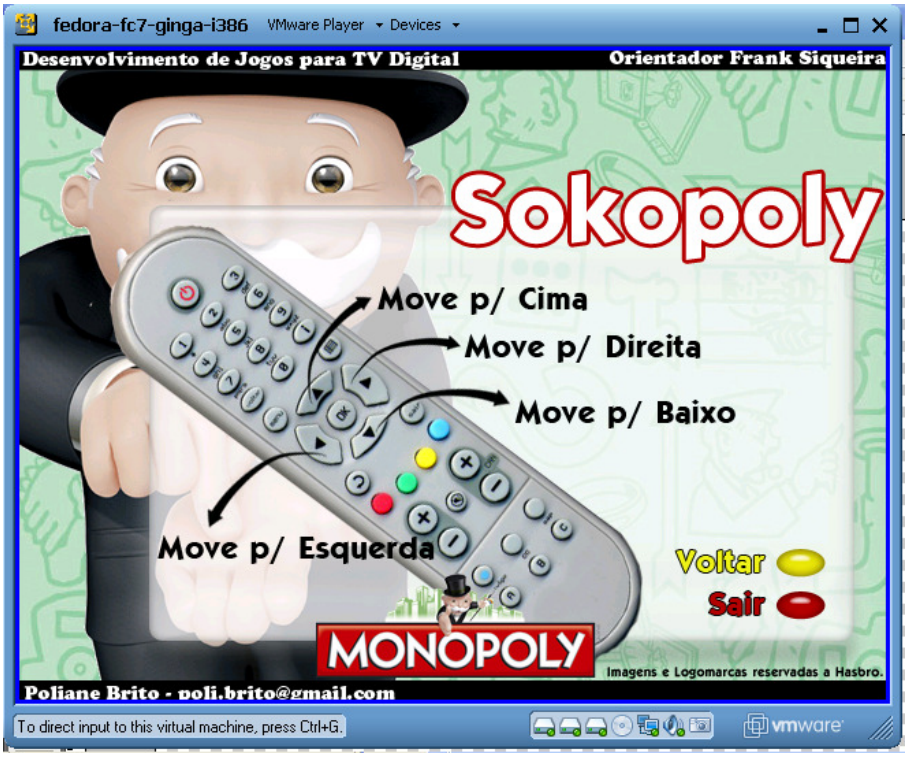

Fig. 12: Tela de Ajuda do Jogo

Todos os códigos e telas mostrados até agora, são gerados apenas pelo documento NCL.

Os próximos blocos apresentados serão scripts da linguagem Lua, cujo script é acionado pelo documento NCL.

Fig. 13: Tela Principal do Jogo

Na figura 12, observamos a tela principal do jogo, com o tabuleiro gerado através do módulo `canvas` do NCLua.

Já os eventos são registrados através do módulo `event`:

```
      function handler (evt)
            if evt.class == 'key' and evt.type == 'press' then
                  if evt.key == 'CURSOR_UP' then
                  ...
                  elseif evt.key == 'CURSOR_DOWN' then
                  ...
                  elseif evt.key == 'CURSOR_LEFT' then
                  ...
                  elseif evt.key == 'CURSOR_RIGHT' then
                  ...
                  end
            end
            desenhaLevel()
      end
event.register(handler)
```

No código, observamos o registro da função `handler`, a qual escuta e trata os eventos gerados pelo controle remoto, fazendo a movimentação do jogador e, conseqüentemente das caixas sob o tabuleiro.

Quanto a função `desenhaLevel()`, ela é a responsável pela criação do tabuleiro e atualização do mesmo, utilizando os recursos do módulo `canvas`.

Utilizou-se nesta tela dos números 1 e 3 (Tela Inicial e Sair respectivamente) para eventos de transição, pois na máquina virtual do *set-top box* disponibilizada (versão 0.10.1) possui um *bug,* que não permite que as teclas F1, F2, F3 e F4 (respectivamente botões vermelho, verde, amarelo e azul) sejam mapeadas quando executadas a partir de um script NCLua.

Também, tentou-se a elaboração de uma função a qual implementava mais de um *level* ao jogo; utilizando-se da biblioteca padrão *io* da linguagem Lua seria feita a leitura de um arquivo *.txt* onde estariam os *leveis* armazenados, e que apenas mais tarde, encontrou-se uma fonte onde dizia que tal biblioteca não era distribuída junto com o pacote do Ginga.

6. Conclusão e Perspectivas Futuras

Ao longo do desenvolvimento deste projeto, encontrou-se muitas dificuldades tanto na elaboração do documento quanto a implementação da aplicação. Ainda não há um vasto conjunto de informações sobre o assunto escolhido e, quando encontrados, os dados são repetitivos, porque são muitas as mudanças e as informações se tornam desatualizadas rapidamente.

O aprofundamento no assunto é de grande valia, visto que grandes empresas estão investindo para que sejam pioneiras em aplicativos especialistas ou mesmo genéricos, sendo notável o crescimento e o potencial da área e, conseqüentemente, exigindo mais profissionais aptos para tal.

Analisando as ferramentas disponibilizadas para o desenvolvimento das aplicações, as mesmas ainda estão em estado de pesquisa e desenvolvimento e, assim, encontrou-se muitos *bugs* ou incompatibilidades de versões. Como trabalho futuro, a elaboração de ferramentas que apóiem os desenvolvedores de aplicações para a TV Digital, é de grande ajuda, visto que diminuiria o tempo de desenvolvimento das aplicações.

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, Christopher; Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press, 1977.

ALMAS, Almir. *Televisão digital interativa: hora de conteúdo.* Revista Comunicação & Comunidade, Rio de Janeiro, 2005.

ALVAREZ, M. E. B; *Organização, Sistemas e Métodos*. São Paulo: McGraw-Hill, 1990.

BECKER, Valdecir; *TV Digital Interativa*. 2007

CRAWFORD, Chris; *The Art of Computer Game Design.* Berkeley: McGraw-Hill, 1984.

IERUSALIMSCHY, R.; FIGUEIREDO, L.H.; CELES, W.; *"Lua 5.0 Reference Manual". Technical Report*. PUC-Rio, 2003.

IERUSALIMSCHY, R.; *Programming in Lua – Lua.org*. 2003.

ISDB; *"ISDB-T - Terrestrial Integrated Services Digital Broadcasting: Specification of Channel Coding, Framing Structure and Modulation",* September 1998.

MARTINS, Ricardo Benetton. *apud* BECKER, Valdecir; MONTEZ, Carlos. *TV Digital Interativa: conceitos, desafios e perspectivas para o Brasil.* Florianópolis: Editora da UFSC, 2004.

Middleware Ginga – TV Interativa se faz com Ginga! *http://www.**ginga**.org.br*. Acessado em 10/09/2009.

Portal do *Software* Público Brasileiro - *http://www.**softwarepublico**.gov.br*. Acessado em 02/08/2009.

SANT'ANNA, F.; NETO,C.S.S.; BARBOSA, S.D.J.; SOARES, L.F.G.; *"Aplicações Declaraivas NCL com Objetos NCLua Imperativos Embutidos"*. PUC-Rio, 2009.

SCHELL, Jesse. *The Art of Game Design: A Book of Lenses.* Burlington: Elsevier, 2008.

SCHUYTEMA, Paul. *Design de games: Uma abordagem prática.* São Paulo: Cengage Learning, 2008.

SOARES, Luis Fernando Gomes. *Middleware Ginga.* Fórum SBTVD.

SOARES, L.F.G.; RODRIGUES, R.F.; MORENO, M.F.; *Ginga-NCL: The Declarative Environment of the Brazilian Digital TV System. Journal of the Brazilian Computer Society*. Porto Alegre – RS, 2007.

SOUZA, A.C.; MACHADO, L.C.S.; SAMPAIO, R.L.; RAIMUNDO, P.O.; *"TV Digital: Limites e Possibilidades Tecnológicas".* 3º Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica – 2008.

Wikipédia – *http://en.**wikipedia**.org*. Acessado em 2009.

ZUFFO, Marcelo Knörich. *TV digital aberta no Brasil: políticas estruturais para um modelo nacional.* Departamento de Engenharia de Sistemas Eletrônicos Escola Politécnica - Universidade de São Paulo.

APÊNDICE

Jogos Digitais para TV Digital Brasileira

Poliane Teixeira Brito[1]

[1]Centro Tecnológico - Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-970 – Florianópolis – SC – Brasil

poliane@inf.ufsc.br

Resumo. *Estudo sobre as linguagens utilizadas pelo Middleware Ginga no seu subsistema Ginga-NCL e o desenvolvimento de um jogo para o mesmo.*

Abstract. *A study of the languages used by the Ginga Middleware in your subsystem Ginga-NCL and developing a game for it.*

1. Introdução

A televisão é o meio de comunicação audiovisual mais difundido nos dias de hoje, pela sua fácil acessibilidade. Com a TV Digital, o Sistema Brasileiro de TV Digital proporcionará uma experiência totalmente nova aos telespectadores e às emissoras – a interatividade. Uma campanha publicitária, por exemplo, poderá interagir diretamente com o seu público alvo, podendo ser utilizada como uma pesquisa de mercado muito mais abrangente e com dados mais precisos.

O foco dos estudos será sobre o middleware Ginga, no seu subsistema Ginga-NCL e a linguagem de script Lua. Ao final, será elaborada uma aplicação multimídia – um jogo – baseado no middleware Ginga, para demonstrar parte do estudo aqui apresentado.

2. Proposta de Trabalho

Neste documento serão mostradas as linguagens utilizadas pelo subsistema do middleware Ginga, o Ginga-NCL. Primeiro a linguagem hipermídia NCL, depois a Linguagem Lua. Por último os resultados do estudo.

3. NCL – Nested Context Language

A linguagem NCL – *Nested Context Language* – é uma linguagem declarativa para autoria de documentos hipermídia baseados no modelo conceitual NCM – *Nested Context Model*. A primeira versão de NCL foi especificada através de uma DTD – *Document Type Definition* – XML (W3C, 1998a).

O NCL provê facilidades para a especificação de aspectos de interatividade, sincronismo espaço temporal entre objetos de mídia adaptabilidade e suporte a múltiplos dispositivos.

Segundo Souza (2008), o uso da linguagem NCL, não é suficiente para o desenvolvimento de aplicações interativas mais exigentes, pois esses aplicativos exigem um controle de fluxo em tempo de execução. Para tal, estendeu-se a linguagem Lua com novas funcionalidades.

3.1. Extensões de NCLua

Para se integrar à NCL, estendeu-se a linguagem Lua, para atender exigências do *middleware* Ginga, tal como responder às teclas do controle remoto ou desenhar dentro da região NCL.

Os seguintes módulos estão disponíveis para scripts NCLua:

- Módulo *event*: permite que os objetos NCLua se comuniquem com o documento NCL e outras entidades externas, como o controle remoto e o canal de interatividade;

- Módulo *canvas*: oferece funcionalidades para desenhar objetos gráficos na região do NCLUA;
- Módulo *settings*: oferece acesso às variáveis definidas no objeto settings do documento NCL (objeto do tipo “application/x-ncl-settings”);
- Módulo *persistent*: exporta uma tabela com variáveis persistentes entre execuções de objetos imperativos.

4. Linguagem Lua

Lua é uma linguagem de script projetada e implementada em 1993, pelo Grupo de Computação Gráfica da PUC-Rio, na necessidade crescente das suas aplicações serem configuráveis externamente pelos usuários. Com isso, conseguiu-se fazer com que alterações simples – como alterações de cores - pudessem serem feitas sem a recompilação da aplicação. Poderosa e leve, a linguagem Lua funciona acoplada a programas hospedeiros implementados em linguagens compiladas.

Lua é tipada dinamicamente, é interpretada a partir de bytecodes para uma máquina virtual (engine) acoplada ao formatador NCL. Essas características fazem da mesma uma linguagem ideal para configuração, automação (scripting) e prototipagem rápida [SOARES, 2007].

4.1. Biblioteca Padrão

Além das funções que pertencem à biblioteca padrão, a distribuição oficial do Lua vem com bibliotecas que implementam várias funções importantes na composição de um programa.

As bibliotecas adicionais são:

- *string*: oferece funções para a manipulação de *strings*;
- *table*: funções para manipulação de tabelas, como inserção, remoção e ordenação de elementos;
- *math*: funções semelhantes à biblioteca matemática de C, como math.sqtr, math.sin, math.log;
- *os:* funções relacionadas ao sistema operacional;

- *debug:* oferece funções para depuração de códigos em Lua.

5. Resultados

Após o estudo realizado, desenvolveu-se então, um jogo para o middleware Ginga para o seu subsistema Ginga-NCL.

O jogo é um quebra-cabeça baseando-se no jogo Sokoban, cujo objetivo é levar todas as caixas aos lugares pré-determinados, passando por obstáculos existentes.

Utilizando-se da linguagem NCL implementou-se as telas inicial e de ajuda do jogo, cujas telas seguem:

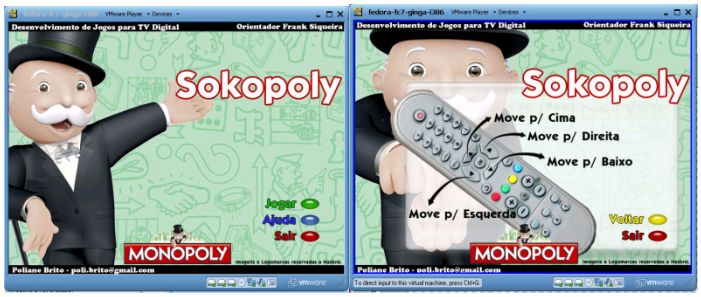

Fig. 1: Telas Inicial e de Auda, respectivamente.

Para a lógica do jogo, foi utilizada a linguagem Lua e as extensões do NCLua, por tratarem de fluxos mais complexos. A seguir a tela principal do jogo:

Fig.2: Tela principal do Jogo.

6. Conclusões

O aprofundamento no assunto é de grande valia, visto que grandes empresas estão investindo para que sejam pioneiras em aplicativos especialistas

ou mesmo genéricos, sendo notável o crescimento e o potencial da área e, conseqüentemente, exigindo mais profissionais aptos para tal.

Também o investimento na implementação de novas ferramentas que apóiem os desenvolvedores da área é de extrema importância, visto que as existentes deixam muito a desejar quando utilizadas, tornando o trabalho mais difícil e longo.

7. Referências Bibliográficas

IERUSALIMSCHY, R.; *Programming in Lua – Lua.org*. 2003.

SOARES, L.F.G.; RODRIGUES, R.F.; MORENO, M.F.; *Ginga-NCL: The Declarative Environment of the Brazilian Digital TV System. Journal of the Brazilian Computer Society*. Porto Alegre – RS, 2007.

SOUZA, A.C.; MACHADO, L.C.S.; SAMPAIO, R.L.; RAIMUNDO, P.O.; *"TV Digital: Limites e Possibilidades Tecnológicas"*. 3º Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica – 2008.

ANEXOS

Código Fonte da Aplicação Desenvolvida

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

<head>

<regionBase>
      <region id="rginicio" left="0" top="0" width="640" height="480"
zIndex="1" />
      <region id="rgLua" left="0" top="0" width="640" height="480"
zIndex="2" />
      <region id="rgajuda" left="0" top="0" width="640" height="480"
zIndex="3" />
</regionBase>

<descriptorBase>
    <descriptor id="dinicio" region="rginicio" focusIndex="1" />
    <descriptor id="dsLua" region="rgLua" focusIndex="2" />
    <descriptor id="dajuda" region="rgajuda" focusIndex="3" />
</descriptorBase>

<connectorBase>

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

      <causalConnector id="onBeginStopN">
            <simpleCondition role="onBegin"/>
            <simpleAction role="stop" max="unbounded" qualifier="seq"
/>
      </causalConnector>

      <causalConnector id="onEndStartN">
            <simpleCondition role="onEnd"/>
            <simpleAction role="start" max="unbounded" qualifier="par"
/>
      </causalConnector>

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

```
            </compoundAction>
      </causalConnector>

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

</connectorBase>

</head>

<body>
    <port id="init" component="inicio"/>

      <media id="programSettings" type="application/x-ginga-settings"
>
            <property name="service.currentKeyMaster" value="1" />
      </media>

      <media id="inicio" type="image/png" src="media/inicio.png"
descriptor="dinicio" />
      <media id="ajuda" type="image/png" src="media/ajuda.png"
descriptor="dajuda" />

    <media id="lua" type="application/x-ginga-NCLua" src="sokoban.lua"
descriptor="dsLua" >
    </media>

      <link xconnector="onBeginStart">
        <bind role="onBegin" component="inicio" />
      </link>

      <link xconnector="onEndStartN">
            <bind role="onEnd" component="lua" />
            <bind role="start" component="inicio"/>
      </link>

      <link xconnector="onKeySelectionStopNStartN">
            <linkParam name="keyCode" value="GREEN" />
            <bind component="inicio" role="onSelection" />
            <bind role="stop" component="inicio"/>
            <bind role="start" component="lua"/>
      </link>

      <link xconnector="onKeySelectionStopNStartN">
            <linkParam name="keyCode" value="BLUE" />
            <bind component="inicio" role="onSelection" />
            <bind role="stop" component="inicio"/>
            <bind role="start" component="ajuda"/>
      </link>

      <link xconnector="onKeySelectionStopNStartN">
            <linkParam name="keyCode" value="YELLOW" />
            <bind component="ajuda" role="onSelection" />
            <bind role="stop" component="ajuda"/>
            <bind role="start" component="inicio"/>
      </link>
```

```
      <link xconnector="onKeySelectionStopN">
            <linkParam name="keyCode" value="RED" />
            <bind component="inicio" role="onSelection" />
            <bind role="stop" component="inicio"/>
            <bind role="stop" component="lua"/>
      </link>

      <link xconnector="onKeySelectionStopN">
            <linkParam name="keyCode" value="RED" />
            <bind component="ajuda" role="onSelection" />
            <bind role="stop" component="ajuda"/>
            <bind role="stop" component="lua"/>
      </link>


</body>

</ncl>



-- Documento NCLUA

local playeri
local playerj
local xis
local ypy
local box_soltas
local level = {}
level[1] =
{'F','F','F','F','#','#','#','#','#','F','F','F','F','F','F','F','F','
F','F'}
level[2] =
{'F','F','F','F','#','','','','#','F','F','F','F','F','F','F','F','F',
'F','F'}
level[3] =
{'F','F','F','F','#','o','','','#','F','F','F','F','F','F','F','F','F'
,'F','F'}
level[4] =
{'F','F','#','#','#','','','o','#','#','F','F','F','F','F','F','F','F'
,'F'}
level[5] =
{'F','F','#','','','o','','o','','#','F','F','F','F','F','F','F','F','
F','F'}
level[6] =
{'#','#','#','','#','','#','#','','#','F','F','F','#','#','#','#','#',
'#','#'}
level[7] =
{'#','','','','#','','#','#','','#','#','#','#','#','','','.','.','#'}
level[8] =
{'#','','o','','','o','','','','','','','','','','','.','.','#'}
level[9] =
{'#','#','#','#','#','','#','#','#','','#','@','#','#','','','.','.','
#'}
level[10] =
{'F','F','F','F','#','','','','','','#','#','#','#','#','#','#','#','#
'}
level[11] =
{'F','F','F','F','#','#','#','#','#','#','#','F','F','F','F','F','F','
F','F'}
```

```
function desenhaLevel()
	fundo = canvas:new('media/fundo.png')
	canvas:compose(0, 0, fundo)
	xis=160
	ypy=160
	box_soltas=0
	for i=1,11 do
		for j=1,19 do
			peca = level[i][j]
			if peca == '#' then

				imgParede = canvas:new('media/wall.png')
				parede = {img=imgParede, x=xis, y=ypy}
				canvas:compose(parede.x, parede.y, parede.img)

				xis=xis+20
			elseif peca == '' then
				xis=xis+20
			elseif peca == 'F' then
				xis=xis+20
			elseif peca == '@' then
				playeri=i
				playerj=j

				imgPlayer = canvas:new('media/player.png')
				player = {img=imgPlayer, x=xis, y=ypy}
				canvas:compose(player.x, player.y, player.img)

				xis=xis+20
			elseif peca == 'o' then
				imgBox = canvas:new('media/box.png')
				box = {img=imgBox, x=xis, y=ypy}
				canvas:compose(box.x, box.y, box.img)
				box_soltas=box_soltas+1

				xis=xis+20
			elseif peca == '.' then
				imgFim = canvas:new('media/end.png')
				fim = {img=imgFim, x=xis, y=ypy}
				canvas:compose(fim.x, fim.y, fim.img)

				xis=xis+20
			elseif peca == '@.' then
				playeri=i
				playerj=j

				imgPlayer = canvas:new('media/player.png')
				player = {img=imgPlayer, x=xis, y=ypy}
				canvas:compose(player.x, player.y, player.img)

				xis=xis+20
			elseif peca == 'o.' then
				imgBox = canvas:new('media/box.png')
				box = {img=imgBox, x=xis, y=ypy}
				canvas:compose(box.x, box.y, box.img)

				xis=xis+20
			end
```

```
            end
            xis=160
            ypy=ypy+20
    end
      canvas:flush()
end


-- Funcao de tratamento de eventos:
function handler (evt)

      -- apenas eventos de tecla me interessam
      if evt.class == 'key' and evt.type == 'press'
      then
            -- apenas as setas movem o player
            if evt.key == 'CURSOR_UP' then

                  if (level[playeri-1][playerj]~='#') and
(level[playeri-1][playerj]~='o') and (level[playeri-1][playerj]~='.')
and (level[playeri-1][playerj]~='o.') then
                        level[playeri-1][playerj]='@'

                        if (level[playeri][playerj]=='@.') then
                              level[playeri][playerj]='.'
                        else
                              level[playeri][playerj]=''
                        end

                  end

                  if (level[playeri-1][playerj]=='.') then
                        level[playeri-1][playerj]='@.'

                              if (level[playeri][playerj]=='@.') then
                                    level[playeri][playerj]='.'
                              else
                                    level[playeri][playerj]=''
                              end

                  end


                  if ((level[playeri-1][playerj]=='o') or
(level[playeri-1][playerj]=='o.')) and (level[playeri-
2][playerj]~='#') and (level[playeri-2][playerj]~='F') and
(level[playeri-2][playerj]~='o') and (level[playeri-2][playerj]~='o.')
then

                        if (level[playeri-2][playerj]=='.') then
                              level[playeri-2][playerj]='o.'
                        else
                              level[playeri-2][playerj]='o'
                        end

                        if (level[playeri-1][playerj]=='o.') then
                              level[playeri-1][playerj]='@.'
                        else
                              level[playeri-1][playerj]='@'
                        end

                        if (level[playeri][playerj]=='@.') then
```

```
                              level[playeri][playerj]='.'
                        else
                              level[playeri][playerj]=''
                        end

                  end

            elseif evt.key == 'CURSOR_DOWN' then

                  if (level[playeri+1][playerj]~='#') and
(level[playeri+1][playerj]~='o') and (level[playeri+1][playerj]~='.')
and (level[playeri+1][playerj]~='o.') then
                        level[playeri+1][playerj]='@'

                        if (level[playeri][playerj]=='@.') then
                              level[playeri][playerj]='.'
                        else
                              level[playeri][playerj]=''
                        end
                  end

                  if (level[playeri+1][playerj]=='.') then
                        level[playeri+1][playerj]='@.'
                              if (level[playeri][playerj]=='@.') then
                                    level[playeri][playerj]='.'
                              else
                                    level[playeri][playerj]=''
                              end
                  end

                  if ((level[playeri+1][playerj]=='o') or
(level[playeri+1][playerj]=='o.')) and
(level[playeri+2][playerj]~='#') and (level[playeri+2][playerj]~='F')
and (level[playeri+2][playerj]~='o') and
(level[playeri+2][playerj]~='o.') then

                        if (level[playeri+2][playerj]=='.') then
                              level[playeri+2][playerj]='o.'
                        else
                              level[playeri+2][playerj]='o'
                        end

                        if (level[playeri+1][playerj]=='o.') then
                              level[playeri+1][playerj]='@.'
                        else
                              level[playeri+1][playerj]='@'
                        end

                        if (level[playeri][playerj]=='@.') then
                              level[playeri][playerj]='.'
                        else
                              level[playeri][playerj]=''
                        end

                  end

            elseif evt.key == 'CURSOR_LEFT' then

                  if (level[playeri][playerj-1]~='#') and
(level[playeri][playerj-1]~='o') and (level[playeri][playerj-1]~='.')
and (level[playeri][playerj-1]~='o.') then
```

```
                        level[playeri][playerj-1]='@'
                        if (level[playeri][playerj]=='@.') then
                              level[playeri][playerj]='.'
                        else
                              level[playeri][playerj]=''
                        end
                  end

                  if (level[playeri][playerj-1]=='.') then
                        level[playeri][playerj-1]='@.'
                              if (level[playeri][playerj]=='@.') then
                                    level[playeri][playerj]='.'
                              else
                                    level[playeri][playerj]=''
                              end
                  end

                  if ((level[playeri][playerj-1]=='o') or
(level[playeri][playerj-1]=='o.')) and (level[playeri][playerj-
2]~='#') and (level[playeri][playerj-2]~='F') and
(level[playeri][playerj-2]~='o') and (level[playeri][playerj-2]~='o.')
then

                        if (level[playeri][playerj-2]=='.') then
                              level[playeri][playerj-2]='o.'
                        else
                              level[playeri][playerj-2]='o'
                        end

                        if (level[playeri][playerj-1]=='o.') then
                              level[playeri][playerj-1]='@.'
                        else
                              level[playeri][playerj-1]='@'
                        end

                        if (level[playeri][playerj]=='@.') then
                              level[playeri][playerj]='.'
                        else
                              level[playeri][playerj]=''
                        end

                  end

            elseif evt.key == 'CURSOR_RIGHT' then

                  if (level[playeri][playerj+1]~='#') and
(level[playeri][playerj+1]~='o') and (level[playeri][playerj+1]~='.')
and (level[playeri][playerj+1]~='o.') then
                        level[playeri][playerj+1]='@'
                        if (level[playeri][playerj]=='@.') then
                              level[playeri][playerj]='.'
                        else
                              level[playeri][playerj]=''
                        end
                  end

                  if (level[playeri][playerj+1]=='.') then
                        level[playeri][playerj+1]='@.'
                              if (level[playeri][playerj]=='@.') then
                                    level[playeri][playerj]='.'
                              else
```

```
                                    level[playeri][playerj]=''
                              end
                  end

                  if ((level[playeri][playerj+1]=='o') or
(level[playeri][playerj+1]=='o.')) and
(level[playeri][playerj+2]~='#') and (level[playeri][playerj+2]~='F')
and (level[playeri][playerj+2]~='o') and
(level[playeri][playerj+2]~='o.') then

                        if (level[playeri][playerj+2]=='.') then
                              level[playeri][playerj+2]='o.'
                        else
                              level[playeri][playerj+2]='o'
                        end

                        if (level[playeri][playerj+1]=='o.') then
                              level[playeri][playerj+1]='@.'
                        else
                              level[playeri][playerj+1]='@'
                        end

                        if (level[playeri][playerj]=='@.') then
                              level[playeri][playerj]='.'
                        else
                              level[playeri][playerj]=''
                        end

                  end

            elseif evt.key == '1' then

                  event.post{class='ncl', type='presentation',
label='ReStart',action='start'}

            elseif evt.key == '3' then

                  event.post{class='ncl', type='presentation',
action='stop'}

            end

            if box_soltas==0 then

                  event.post{class='ncl', type='presentation',
label='fim', action='start'}

            end

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