UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

2009.2

MOONDO: UM FRAMEWORK PARA DESENVOLVIMENTO DE APLICAÇÕES DECLARATIVAS NO SBTVD

TRABALHO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Aluno:	Thiago Monteiro Prota	{tmp@cin.ufpe.br}
Orientador:	Fernando da Fonseca de Souza	{fdfd@cin.ufpe.br}
Co-Orientador:	Bruno de Sousa Monteiro	{bruno84@gmail.com}

18 de janeiro de 2010

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MOONDO: UM FRAMEWORK PARA DESENVOLVIMENTO DE APLICAÇÕES DECLARATIVAS NO SBTVD

Thiago Monteiro Prota

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Fernando da Fonseca de Souza
Co-orientador: Bruno de Sousa Monteiro

18 de janeiro de 2010

"Uma frase inteligente não prova nada."

(Voltaire)

Agradecimentos

Primeiramente, gostaria de dizer que estes agradecimentos não são apenas para aqueles que contribuíram diretamente com este trabalho, mas para todos aqueles que, em qualquer etapa da minha vida, colaboraram para que eu pudesse chegar onde estou. Outra informação extremamente importante é o fato de que a ordem dos agradecimentos não implica em importância, não passa de uma mera organização estética textual.

Em especial, gostaria de agradecer a toda minha família, pelo apoio e paciência dedicados, principalmente aos meus pais, Lucyclere (Gorda) e Francisco (Chicão), meus irmãos, Felipe (Pipe) e Túllio (Tuca), meu tio e padrinho, Alexandre, e minha avó Dorinha (*in memoriam*), que mesmo não estando mais entre nós foi uma das que mais sonhou com o meu sucesso. Neste parágrafo familiar acho que já posso incluir a mais nova integrante, minha amiga e namorada, Roberta Nazário (Florzinho), que com certeza foi uma das que mais sofreu neste fim de processo, a você Florzinho, muito obrigado.

Gostaria também de agradecer aos meus orientadores, não por puro compromisso, mas por toda gratidão, Fernando Fonseca, meu eterno orientador de Iniciação Científica, Trabalho de Graduação e futuramente de Mestrado, e Bruno Monteiro, por todo embasamento teórico fornecido e pela parceira em trabalhos publicados que fizeram deste, um trabalho diferenciado. Pelo desenvolvimento deste, gostaria de agradecer também a Rodrigo Mateus (Bigo) por todo auxílio criativo e gráfico. No âmbito profissional, gostaria exprimir meus agradecimentos aos superiores e amigos, Luciano Guedes, Antônio Noberto (Tôca) e Joaquim Calado, pelos conselhos e compreensão, e aos não superiores, mas também amigos, os integrantes das equipes do GPF e do antigo Pool, entre outros integrantes do e-Fisco: Dourado, Fabiana, Celso, Hiroshi, Aline, Carioca, Senninha, Pacheco, Mário, George, Abelardo, Ailton, Sacha, Assis e Marcela.

Durante esta fase passei por várias equipes, grupos e times, na área acadêmica, profissional e pessoal, e gostaria de exprimir meus agradecimentos também a cada um deles: iTeam, Ingá, xTeam (iTeam + Alana), Pool, GPF, Garotos de Programa F.C., CinBolando (o time com o "i" vermelho, O Campeão), Los Cablocos Lokos F.C., Gangue e Equipe Desmantelo. Gostaria de agradecer também aos meus amigos do colégio (Visão e Contato) que mesmo afastados foram importante para o meu desenvolvimento pessoal, Gustavo (Guga), Rodrigo (Digão), André (Dedé), Patchola (Antônio), Paulo, Betinho, Bidoba, Fernando, Galego, Karen e Laiz.

E por ultimo, e com certeza não menos importante gostaria de agradecer a todos aqueles que não me fizeram virar um estudante de computação clichê. Aos inicialmente companheiros de turma, porém, hoje, companheiros da vida, presente em todos os dias (e algumas noites!): Petrônio (Pet), Marcio, Hazin, Luiz (Lula), Hugo (Haole), Morato e Leandro (Shark); a galera de engenharia: Guilherme, Cleunio, Felipe, Fernando, Gringo, Renan e Perazzo; e as meninas: Aninha, Marcela e Alana. E aos mais recentes companheiros, não só de festas e viagens, a galera da Gangue: Roberta, Lázaro, Kety, Felipe, Bruno, July e Tham.

Se mesmo depois de todos estes nomes você não foi citado e se sente injustiçado, só me resta pedir desculpas.

A todos vocês, muito obrigado!

Resumo

Por não existirem padrões para o desenvolvimento de aplicações declarativas, surge-se a necessidade de criar uma arquitetura de referência para o desenvolvimento de aplicações declarativas para TVD. Além de prover uma orientação arquitetural ao desenvolvedor, predefinindo o fluxo de controle da aplicação, é necessário que se tenha um modelo com classes básicas e constantes, a fim de promover o reuso, para minimizar o esforço do desenvolvimento. Este trabalho tem como objetivo formalizar um *framework* para o desenvolvimento de aplicações declarativas em NCL e Lua que facilitará o desenvolvimento destas aplicações, bastando ao desenvolvedor customizar o *framework* para uma aplicação particular. Além disso, disponibilizará mecanismos para permitir que o desenvolvedor se conecte às diversas funções, as estenda ou implemente aquelas que ainda não estão presentes.

Palavras-chave: Televisão Digital, Framework, Desenvolvimento de Aplicações, SBTVD, Lua, NCL.

Abstract

Nowadays there are no ideal standards in the development of Digital TV (DTV) declarative applications. Then comes the necessity to create a reference architecture to develop this kind of applications. Besides providing architectural guidance to application control flow, it's necessary to have a model with basic classes and constants to provide reuse. This research aims to formalize a framework to develop declarative applications that use NCL and Lua language. It intends to facilitate the development process, allowing the developer only to customize a particular application context inside the framework. In addition, it also enables the developer to connect various functions, to extend or implement new functionalities.

Keywords: Digital TV, Framework, Application Development, SBTVD, Lua, NCL.

Lista de Figuras

Lista de Quadros

Índice

1 Introdução

Atualmente, é grande a quantidade de informações que circula nos meios de comunicação a respeito da TV Digital (TVD). Porém, o processo de digitalização do sistema de TV não é novidade e já é bastante explorado pelas empresas de televisão por assinatura (e.g. NET e SKY). Entre as principais vantagens da TVD, frente à TV analógica tradicional, estão: melhor qualidade do vídeo e áudio; interatividade; otimização do espectro de freqüência; mobilidade e multiprogramação [BECKER e MONTEZ, 2004].

Com relação às aplicações interativas, segundo Nuno Bernardo [BERNADO, 2002], os serviços oferecidos para TVD podem ser divididos em 10 categorias: EPG (*Electronic Program Guide*), Comércio eletrônico, Banco eletrônico, E-mail, Internet, Portais de televisão interativa, Aplicações transversais aos canais, Programas interativos, Publicidade interativa e Jogos.

No contexto da TVD, a preocupação com usabilidade é primordial, pois as aplicações para TVD são completamente diferentes daquelas que os desenvolvedores estão acostumados a construir para PC (*Personal computer*). Neste aspecto, vale destacar as peculiaridades dos dispositivos de entrada e saída que devem ser consideradas durante o desenvolvimento. Portanto, como forma de ajudar os desenvolvedores de aplicação nessa nova realidade, algumas organizações (e.g. BBC [BBCi, 2002]) buscam escrever manuais e guias com dicas e informações úteis, que se seguidos, possibilitam o desenvolvimento de interfaces gráficas mais agradáveis e intuitivas.

Por não haver um estilo unificado de interação para aplicações na TVD, este tema está sendo fortemente estudado por empresas e instituições de pesquisa, na busca de padrões que permitam bons níveis de usabilidade. Por outro lado, estudos voltados especificamente aos dispositivos de interação também permitem novas possibilidades no campo da usabilidade, por exemplo, novos controles remotos, teclados virtuais, conectividade com celulares e PDA [SILVA *et. al.*, 2007].

No âmbito das aplicações para a TVD, existem três possíveis classificações quanto ao tipo de execução do software: Procedural, Declarativo e Híbrido. As procedurais possibilitam que o programador seja capaz de estabelecer todo o fluxo de controle e execução de seu programa [SOARES *et. al.*, 2007]. As declarativas têm como foco o sincronismo de mídias, a adaptabilidade, produção de conteúdo e suporte a múltiplos dispositivos. Já as Híbridas necessitam de engenhos que dê suporte a apresentação (Declarativo) e execução (Procedural) representam a união dos dois outros grupos, procedurais e declarativas. O *middleware* dispõe de facilidades para o desenvolvimento sob este paradigma, oferecendo APIs que visam dar suporte a integração destes grupos.

Diante deste cenário, surge a necessidade de criar soluções que auxiliem o desenvolvimento de aplicações declarativas para TVD. Para que isto seja feito de forma eficiente, é necessário que se proponha uma arquitetura de referência para que as aplicações possam ser criadas sobre ela, visando o ganho de qualidade e desempenho no processo. Além de prover uma orientação arquitetural ao desenvolvedor, predefinindo o fluxo de controle da aplicação, é necessário que se tenha um modelo com classes básicas e constantes que promovam o reuso, a fim de minimizar o esforço do desenvolvimento [SILVA et al., 2006 ; SILVA e PRICE, 1997].

Diante deste desafio, o presente trabalho tem como objetivo descrever o processo de concepção de um *framework* de desenvolvimento de aplicações declarativas para plataforma de TV Digital (TVD), intitulado MoonDo. Johnson et al. (1995) define um framework como um conjunto de classes cooperativas que compõem um modelo reusável para uma classe de software específica. O framework MoonDo visa abstrair o nível de complexidade técnica, durante a fase de codificação, sem prejudicar a qualidade da interatividade das aplicações.

1.1 Motivação

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. A escassez de materiais de referência é bastante visível, principalmente aqueles voltados a questões específicas, como modelos arquiteturais e padrões de projeto.

Por não possuir um modelo bem definido para o desenvolvimento de aplicações declarativas, é de extrema importância a criação de um *framework* que auxilie a autoria destas aplicações, baseado nos conceitos de produtividade de software, onde aplicações para TVD serão construídas com vistas para o reuso, otimizando o tempo e reduzindo os custos do desenvolvimento. Particularmente, um *framework* que forneça um modelo arquitetural modularizado, estruturado e bem definido, aumenta a qualidade, eficácia e eficiência do processo.

Com este recurso grande parte da complexidade técnica é abstraída, durante a fase de codificação, o que permite uma maior participação dos especialistas do domínio do problema no desenvolvimento das aplicações, promovendo a difusão desta tecnologia e potencializando as vantagens da TV Digital Interativa.

1.2 Objetivos

Esta seção visa descrever os objetivos deste trabalho, primeiramente de forma mais geral para mostrar a dimensão do projeto e posteriormente detalha os principais objetivos a fim de descrever cada contribuição proposta.

1.2.1 Objetivo geral

O presente trabalho tem por objetivo a criação e validação de um *framework* para o desenvolvimento de aplicações declarativas para o SBTVD. Visa oferecer uma solução que forneça um modelo arquitetural modularizado, estruturado e bem definido, para que seus componentes (gráficos e funcionais) possam ser reutilizados e parametrizados de acordo com as características da aplicação que está sendo desenvolvida. Dentre as principais funcionalidades deste *framework* estão: tratamento das ações dos usuários; funções de escrita e leitura de arquivos (metadados);

manipulação de componentes de interface gráfica; e permitir que o desenvolvedor modifique ou crie novos componentes gráficos e funcionalidades utilitárias (candidatas ao reuso).

1.2.2 Objetivos específicos

Especificamente, pretende-se:

I. Definir um *framework* para utilizar no processo de concepção e desenvolvimento de aplicações declarativas para TVD;

II. Definir uma orientação arquitetural adequada para o desenvolvimento de aplicações declarativas para TVD, com o objetivo de facilitar o desenvolvimento das mesmas;

III. Definir os componentes necessários para o domínio de aplicações na TVD a partir de recursos gráficos disponíveis como HAVi [PAES, 2005], Lwuit [Lwuit, 2009], Flex [Flex, 2009], entre outras tecnologias;

IV. Desenvolver (estender) os componentes gráficos idealizados, com base nos dados da análise de requisitos; e

V. Elaborar uma aplicação exemplo com o *framework* desenvolvido e avaliar o resultado.

1.3 Metodologia

Nesta seção são descritas as principais fases da metodologia sugerida. Para que o trabalho aqui proposto possa ser desenvolvido com sucesso, serão executadas as seguintes atividades:

1. **Levantamento bibliográfico:** Fase responsável pela identificação de trabalhos relacionados, compreensão dos detalhes tecnológicos envolvidos, e revisão do estado da arte.

2. **Levantamento de requisitos:** Esta etapa visa coletar as necessidades e possibilidades do desenvolvimento do framework. Sendo tarefa do pesquisador, explorar novas possibilidades que ainda não foram

vislumbradas, através da análise de sistemas correlatos, pelo estudo de comportamento dos usuários, e por limitações tecnológicas e sociais;

3. **Concepção:** Esta fase visa propor todo o framework, detalhando como seus componentes serão desenvolvidos e integrados com base nos requisitos elicitados anteriormente;
4. **Desenvolvimento:** Nesta etapa o framework será desenvolvido, juntamente com todos os seus componentes, sendo orientado pelos artefatos gerados pela fase anterior;
5. **Avaliação:** Esta etapa foca na análise do desenvolvimento de uma aplicação real. Desta forma, busca-se validar se o framework desenvolvido obteve um elevado grau de aceitação; e
6. **Melhorias e refinamentos:** Esta atividade buscará identificar e realizar refinamentos que melhorem os resultados até o momento obtidos.

1.3.1 Método de execução

A seguir a descrição dos métodos para a execução para este trabalho:

I. Documentar, avaliar e analisar os resultados obtidos;

II. Preparar artigos, resumos e apresentações a fim de publicar em congressos, periódicos e revistas;

III. Executar reuniões periódicas entre o aluno e os orientadores para discutir o progresso do projeto, além de avaliar se o projeto segue a proposta feita inicialmente.

1.4 Organização do texto

Este trabalho foi organizado considerando-se desde a contextualização dos principais conceitos envolvidos na pesquisa, até os resultados adotados na concepção do MoonDo e os detalhes técnicos referentes ao desenvolvimento. Com este objetivo, ele encontra-se organizada, além deste, nos seguintes capítulos:

- **Capítulo 2 – TV Digital Interativa.** Neste capítulo são descritas as principais características de um sistema de TVD e as tecnologias

envolvidas nos sistemas atualmente adotados no cenário mundial, com destaque aos padrões do Sistema Brasileiro de TVD.

- **Capítulo 3 – Frameworks de Desenvolvimento.** Explicita a as características dos frameworks de desenvolvimento e os seus processos de concepção e desenvolvimento, além de descrever técnicas e metodologias importantes para estes processos.
- **Capítulo 4 – Trabalhos Relacionados.** Nesta seção são descritos alguns esforços relacionados a este trabalho, e posteriormente é feita uma análise comparativa destes, com o intuito de identificar algumas características que puderam ser incorporadas pelo MoonDo.
- **Capítulo 5 – Desenvolvimento do Framework MoonDo.** Descreve todo o processo de concepção do MoonDo, mostra quais padrões de projetos foram adotados no desenvolvimento e detalha os componentes gráficos desenvolvidos. Além de descrever, como estudo de caso, o desenvolvimento de uma aplicação fazendo uso do framework MoonDo.
- **Capítulo 6 – Conclusão.** São apresentadas as contribuições e conclusões das atividades desenvolvidas durante o projeto, bem como perspectivas de trabalhos futuros.

2 TV Digital Interativa

Apesar da crescente quantidade de informação que circula nos meios de comunicação a respeito da TV Digital, o conceito e o processo de digitalização do sistema de TV não é novidade. Esta tecnologia já é bastante explorada pelas empresas de televisão por assinatura, porém, utilizam padrões proprietários no suporte às aplicações interativas e adotam diferentes meios de transmissão (cabo e satélite). Entretanto, no cenário mundial, na década de 1990, iniciou-se a corrida dos padrões de TV Digital aberta, resultando em três sistemas consolidados atualmente: o *Advanced Television Systems Committee* (ATSC), desenvolvido nos Estados Unidos [ATSC, 2008]; o *Digital Video Broadcasting* (DVB), utilizado na maioria dos países europeus [DVB, 2009] e o *Integrated Services Digital Broadcast* (ISDB), desenvolvido no Japão [ISDB, 2008]. Cada um desses padrões apresenta características específicas referentes à modulação, ao transporte do sinal, à compressão dos dados e à especificação do *middleware*. No Brasil, a preocupação com a desigualdade na distribuição de tecnologia e acesso à informação levou o Governo Federal a buscar as melhores soluções aplicáveis à realidade brasileira, na digitalização da TV aberta, através do projeto SBTVD [SBTVD, 2009].

Apesar de todo conjunto de avanços tecnológicos, é importante voltar-se à realidade de países como o Brasil, onde a distribuição irregular de tecnologia e comunicação é um fato visível no acesso à Internet e ao uso do computador. Segundo pesquisa do Comitê Gestor da Internet no Brasil [CGI, 2008], apenas 25% da população possui computador, e dentro deste mesmo grupo, apenas 71% dos computadores estão conectados Internet. Em contrapartida, 97% dos brasileiros têm acesso a TV, de acordo com esta mesma pesquisa, confirmando que a TV é um recurso bastante favorável como meio de promoção do resgate social e transmissão de conteúdo instrutivo.

2.1 Estrutura geral

Todo o esforço gasto para digitalizar o sistema de televisão não é por acaso. Entre as principais vantagens da TVD, frente à TV analógica tradicional, estão:

melhor qualidade do vídeo e áudio; interatividade; otimização do espectro de freqüência; mobilidade e multiprogramação [BECKER e MONTEZ, 2004].

Diferentemente da TV convencional, no modelo digital, o fluxo recebido pelo usuário final é composto por três tipos de subfluxos: áudio, vídeo e dados (Figura 2.1). Estes subfluxos são multiplexados pelo Servidor de Geração de Conteúdo, que então monta o fluxo principal que será enviado aos receptores (Figura 2.2) [MORRIS, 2005]. O subfluxo de dados pode carregar pacotes de controle como também aplicações, que apresentam algumas limitações de processamento e de interface com o usuário. Estas restrições exigem que os desenvolvedores tomem certos cuidados, tais como usabilidade, escolha adequada das cores, uso correto de imagens, tamanho de fontes, posicionamento dos elementos gráficos e sobrecarga dos recursos de hardware [MORRIS, 2005].

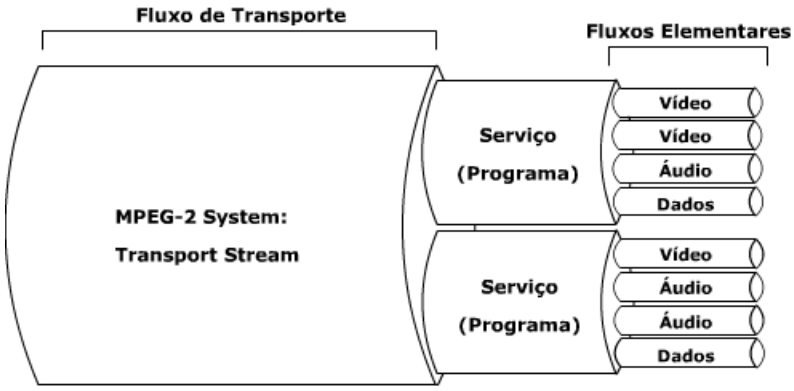


Figura 2.1 Fluxo de transporte e seus sub-fluxos [MONTEIRO, 2009]

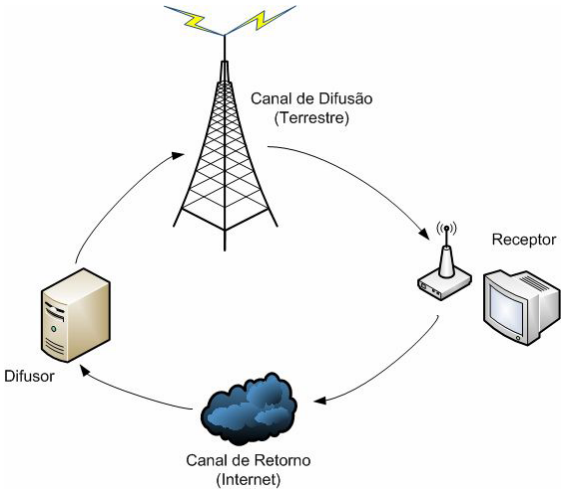


Figura 2.2 Modelo simplificado da infra-estrutura da TV Digital Interativa com meio de transmissão terrestre [MONTEIRO, 2009]

De modo geral, um sistema de TV digital interativa pode ser decomposto em três grandes partes:

I. difusor, responsável por prover o conteúdo a ser transmitido, e dar suporte às interações com os telespectadores;

II. receptor que recebe e apresenta o conteúdo e possibilita ao telespectador interagir com o difusor; e

III. meio de difusão, composto por canal de difusão e canal de retorno (ou canal de interatividade), que habilita a comunicação entre difusor e receptor.

Entretanto, com relação à arquitetura, o sistema de TVD é dividido em cinco camadas: Modulação, Transporte, Compressão, *Middleware* e Aplicações (Figura 2.3).

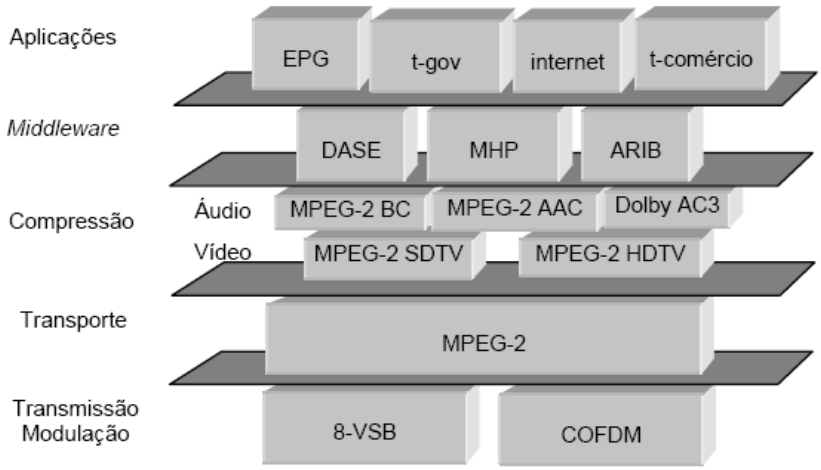


Figura 2.3 Arquitetura em camadas da TV Digital [MONTEZ e BECKER, 2005]

2.2 Aplicações e Serviços para TVD

Entre as especificações de *middleware* dos sistemas de televisão digital abertos, Java [SUN, 2009] é a linguagem predominante no desenvolvimento de aplicações, juntamente com o suporte a linguagens declarativas. Assim, graças à API Java-DTV [Java-DTV, 2009], implementada pela Sun Microsystems [SUN, 2009], os desenvolvedores podem ter acesso às funcionalidades disponibilizadas pelo receptor STB (*Set-Top-Box*) para construir aplicações (*Xlets*) que serão executadas sobre uma

Máquina Virtual. Esta metodologia garante a compatibilidade entre as aplicações que utilizam esse padrão, além de abstrair detalhes de baixo nível [BATISTA, 2006].

Com relação ao canal de retorno utilizado, a interação de uma aplicação pode ser classificada como:

- **Interação Local** - quando não há canal de retorno, o usuário faz uso apenas dos dados que estão instanciados no STB, recebidos através da rede broadcast;
- **Canal "half-duplex"** - o canal de retorno é utilizado apenas nos momentos que é necessário enviar informações para o servidor da aplicação; ou
- **Canal "full-duplex**" - o canal de retorno é utilizado de forma integral, tanto para o envio de informações para o servidor, quanto para receber informações, como por exemplo, um vídeo sob demanda.

Além de serem classificadas quanto à forma de interatividade, as aplicações também pode ser classificadas quanto à sua finalidade, são divididas em 10 categorias [Bernardo, 2002]: EPG (Electronic Program Guide), Comércio eletrônico, Banco eletrônico, E-mail, Internet, Portais de televisão interativa, Aplicações transversais aos canais, Programas interativos, Publicidade interativa e Jogos. Embora algumas dessas aplicações possam parecer bastante atrativas, como por exemplo, o serviço de e-mail e banco eletrônico, eles não tiveram grande sucesso na Europa. Constatou-se que aplicações intrínsecas à realidade da Internet, não foram bem aceitas na TVD. Este fato mostra a visível diferença entre a realidade das aplicações para computadores e TV, e as peculiaridades intrínsecas a cada uma dessas plataformas [BECKER, 2005].

2.3 SBTVD

Com o estabelecimento de um sistema brasileiro de televisão digital (SBTVD), firmaram-se as regras e especificações que servirão de parâmetro para o novo sistema. O SBTVD rege e normatiza itens como: *middleware*, canal de interatividade e técnicas de compressão, multiplexação, modulação, transmissão e recepção de vídeo, áudio e dados.

A especificação do SBTVD foi baseada no padrão de TV digital japonês, com modificações na camada de compressão e na camada de *middleware*. No caso da compressão de vídeo, todos os padrões de TV Digital Terrestre empregam o MPEG-2 [BRACKMANN, 2008]. O Brasil, no entanto, emprega uma técnica de compressão de vídeo mais recente e mais eficiente, denominada de H.264 ou MPEG-4 AVC (*Advanced Video Coding*, ISO/IEC 14.496 Parte 10) [BRACKMANN, 2008]. O MPEG-4 utiliza a mesma estrutura básica do algoritmo de codificação utilizada no MPEG-2, porém, incorpora novas funcionalidades que proporcionam um ganho significativo na razão taxa de bits por distorção. Com esta técnica de compressão de vídeo, é possível manter a qualidade de imagem, porém reduzindo sensivelmente a taxa de *bits, e*ste ganho de desempenho de compressão resulta em um uso mais eficiente do espectro.

Para a codificação do áudio, o SBTVD utiliza o MPEG-2:AAC bem como o MPEG-4:AAC, para que possa dar suporte ao som estéreo e ao recurso do surround [TOME et al., 2007]. Composto por um sistema que utiliza o método SBR (*Spectral Band Replication*) para alta eficiência de codificação de áudio, tanto em receptores fixos, assim como portáteis. Esta tecnologia leva em conta o modelo psicoacústico humano, resultando um áudio de alta qualidade e a geração de baixa taxa de bits [BRACKMANN, 2008].

A camada de transmissão, no SBTVD, fica sob a responsabilidade do padrão MPEG-2 System: Transport Stream, que é o mesmo adotado pelos demais sistemas mundiais. O padrão de modulação especificado no sistema brasileiro de TVD é o BST-OFDM (*Band Segmented Transmission-Orthogonal Frequency Division Multiplexing*), que é baseado no padrão de modulação do sistema Japonês, permite que mesmo com interferências, somente uma pequena parte da informação transmitida seja perdida e por isso tecnologia OFDM é mais imune a interferências do ambiente. O BST é responsável pela segmentação de banda, ou seja, torna viável o particionamento do espectro em três partes: vídeo, áudio e dados. Esta modulação prioriza a transmissão terrestre e recepção em dispositivos móveis [BRACKMANN, 2008].

Com relação ao middleware padrão, a solução adotada no Brasil, denominada Ginga, é um produto de tecnologia nacional, sendo composto por um ambiente procedural, denominado Ginga-J, e um módulo declarativo, o Ginga-NCL [MENDES, 2007]. Desde sua concepção, o Ginga levou em consideração a necessidade de inclusão social/digital e a obrigação do compartilhamento de conhecimento de forma livre. O middleware Ginga leva em consideração a importância da televisão, presente na quase totalidade dos lares brasileiros, como um meio complementar à inclusão social/digital.Permitindo levar ao cidadão todos os meios para que ele obtenha acesso à informação, educação à distância (EaD) e serviços sociais apenas usando sua TV, o meio de comunicação onipresente no país [BRACKMANN, 2008].

Algumas características tornam o sistema digital brasileiro único no mundo. A modulação especificada permite a transmissão simultânea de sinal digital tanto para aparelhos de TV quanto para dispositivos móveis como celulares e PDA (*Personal Digital Assistants*) [SILVA et. al., 2007]. Quanto à compressão de vídeo, todos os padrões de TV Digital Terrestre do mundo, exceto o brasileiro, utilizam o MPEG-2. No Brasil, a técnica empregada, H.264, mais conhecida como MPEG-4, faz uso de um algoritmo mais recente do que aqueles em vigência em outros países. Outra característica relevante no SBTVD está relacionada à tecnologia empregada no *middleware*, que dá suporte tanto à linguagem declarativa (NCL) quanto à procedural (Java) para o desenvolvimento de aplicações interativas. Na Figura 2.4 podem-se visualizar hierarquicamente os componentes da arquitetura do SBTVD.

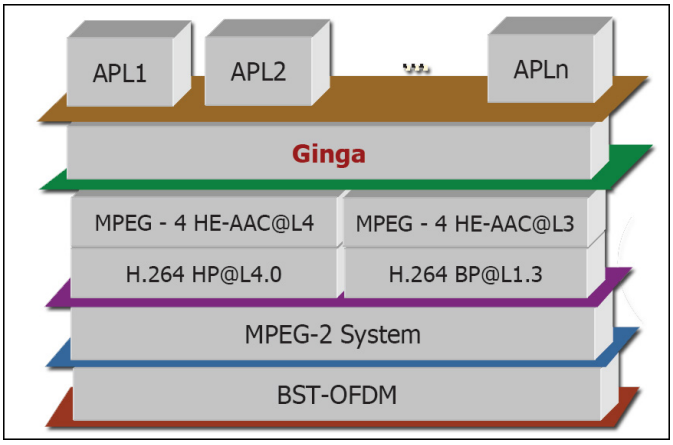


Figura 2.4 Camadas do Sistema Brasileiro de TV Digital [SOUZA, 2008]

2.4 Middleware Ginga

O middleware é uma camada de *software* que fornece uma abstração da heterogeneidade da rede, *hardware* e sistema operacional. Esta camada fornece um conjunto de serviços e modelo computacional uniforme para ser usado pelos programadores de aplicações [COULOURIS et al., 2007]. No contexto da TVD, este componente é responsável por executar as aplicações interativas que são enviadas juntamente com o fluxo televisivo da emissora, que é formado por subfluxos de dados, áudio e vídeo [TOME et al., 2007]. Com relação ao *middleware* padrão, a solução adotada no Brasil, denominada Ginga, é um produto de tecnologia nacional, sendo composto por um ambiente procedural, denominado Ginga-J, e um módulo declarativo, o Ginga-NCL com suporte a linguagem de *script* Lua [SOARES, 2007].

2.4.1 Ambientes de Programação

No âmbito das aplicações para a TVD, existem três possíveis classificações quanto ao tipo de execução do *software*:

- **Procedural** – necessita de uma plataforma de execução (máquina virtual) e no caso do middleware Ginga este módulo é denominado Ginga-J. Por utilizar a linguagem de programação Java, possibilita que o programador seja capaz de estabelecer todo o fluxo de controle e execução de seu programa;
- **Declarativo** – necessita de um engenho de apresentação (Browser) e é apresentada similarmente como uma página HTML (*HyperText Markup Language*) podendo conter scripts e folhas de estilo. No middleware Ginga, este módulo é denominado Ginga-NCL, que utiliza como base o NCL, linguagem que define uma separação entre a estrutura e o conteúdo. Geralmente as aplicações declarativas fazem uso de conteúdos em script, que no caso do NCL há o suporte à linguagem Lua; e
- **Híbrido** – representa a união dos dois grupos, procedural e declarativo. Esta arquitetura se faz necessária, pois as aplicações de TVD são usualmente desenvolvidas utilizando estes dois paradigmas de programação. Entretanto, estes ambientes de programação não estão precisamente disjuntos, ou seja, um

pode utilizar as facilidades do outro através das API (*Application Programming Interface*) contidas no middleware. Esta característica possibilita a criação de aplicações híbridas, que são suportadas pela arquitetura do Ginga, ilustrada pela Figura 2.5 [SOARES *et. al.*, 2007].

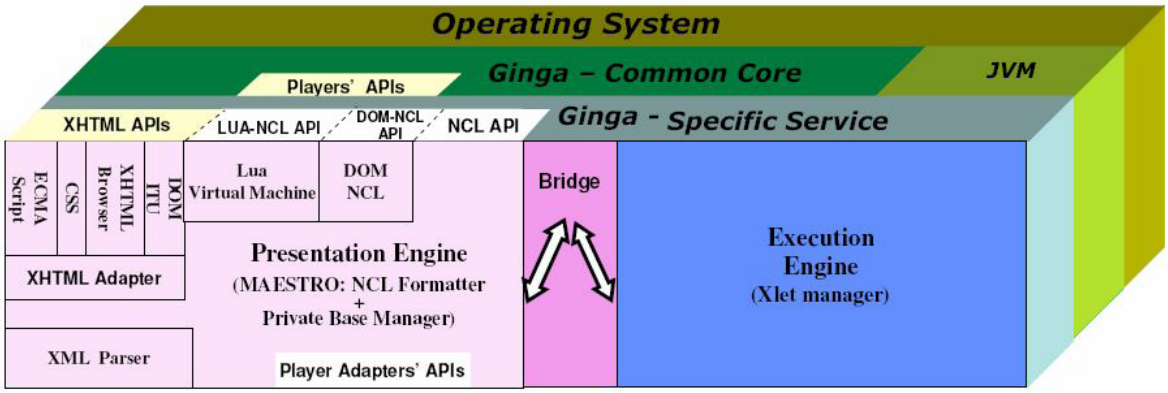


Figura 2.5 Arquitetura do Middleware Ginga [SOARES *et. al.*, 2007]

2.4.1.1 Ambiente Procedural: Ginga-J

Os ambientes procedurais se caracterizam pelo fato do programador possuir um maior controle sobre o código e um maior poder de expressão, devido ao nível de abstração ser mais baixo, se comparado com linguagens declarativas. Desta forma, ele torna-se capaz de definir todo o fluxo de controle e execução da aplicação. Entretanto, possui um elevado custo de implementação, devido ao aumento da complexidade do código fonte, e não são adequadas para o desenvolvimento de programas não-lineares. O Ginga-J é o responsável pelo processamento de aplicações procedurais escritas utilizando a linguagem Java [SUN, 2009]. Ele é dividido em três módulos: a Máquina Virtual Java, o núcleo e suas API [BARBOSA e SOARES, 2008]. O conjunto de API é dividido em três categorias:

- **Verde** – mantém a compatibilidade com o sistema americano e europeu;
- **Amarela** – suporte a múltiplos usuários, dispositivos e redes; e
- **Vermelha** – suporte às aplicações que podem ser recebidas, armazenadas e executada.

As aplicações de natureza procedural desenvolvidas em Java são denominadas "Xlets" [BATISTA, 2006]. Estas aplicações possuem uma interface que

permite que uma fonte externa inicie, pare, ou controle de várias outras formas sua execução [PICCIONI e MONTEZ, 2004]. A linguagem Java também dá suporte a aplicações para TV Digital principalmente através de sua API Java-DTV [Java-DTV, 2009] que fornece as seguintes funcionalidades:

- Fluxo de áudio e vídeo;
- Acesso aos dados nos canais de transmissão;
- Acesso aos dados do *Service Information*;
- Controle do sintonizador de canais;
- Sincronização da mídia; e
- Gerenciamento do ciclo de vida das aplicações.

2.4.1.2 Ambiente Declarativo: Ginga-NCL

O Ginga-NCL é o ambiente do *middleware* Ginga responsável pelo processamento das aplicações declarativas que utilizam a linguagem NCL (*Nested Context Language*) [GINGA-NCL, 2008]. Esta especificação é baseada em XML (*eXtensible Markup Language*) [W3C, 2009] para autoria de documentos hipermídia e tem como vantagem adicional a utilização da linguagem de *script* Lua [LUA, 2009] para a manipulação de suas variáveis, por adotar o paradigma imperativo, ser eficiente, rápida, leve e projetada para estender aplicações.

NCL define o modo com os objetos de mídia são estruturados e relacionados, no tempo e espaço, proporcionando um controle maior sob o conteúdo. Outra característica importante de NCL é a facilidade de reuso das especificações, ou seja, todos os documentos podem importar elementos já declarados em outros documentos. NCL também oferece um nível mais alto de abstração para a autoria de programas. Além disso, é adequada no desenvolvimento de aplicações não-lineares por se tratar de uma linguagem para integração e sincronização de mídias, ou seja, facilita a sincronização temporal e espacial das mídias, dispensando muitas vezes a programação por *scripts* (ou outras estratégias de programação algorítmica) [BARBOSA e SOARES, 2008; SOARES e SOUZA FILHO, 2007].

No Ginga-NCL, uma aplicação de TVD pode ser gerada ou modificada ao vivo, através de comandos de edição, pois ela oferece uma separação entre o conteúdo e a estrutura da apresentação [BARBOSA e SOARES, 2008]. Com as linguagens declarativas, o programador fornece apenas o conjunto das tarefas a serem realizadas, não necessitando, em geral, de tantas linhas de código para definir certa tarefa, se comparado a implementações algorítmicas. Elas provêm facilidades para a especificação de aspectos de interatividade e sincronismo espaço-temporal entre objetos de mídia. Estes objetos de mídia podem ser: imagens, vídeos, áudio, texto e objetos de execução (Xlet e Lua). As linguagens declarativas têm como foco o sincronismo de mídias, a adaptabilidade, edição e produção de conteúdo e o suporte a múltiplos dispositivos.

Neste trabalho foi adotado o paradigma declarativo utilizando o ambiente Ginga-NCL para autoria de documentos hipermídia, porém, fazendo uso de uma linguagem de script imperativa. A linguagem Lua agrega uma série de vantagens: permite a manipulação de suas variáveis; é imperativa, eficiente, rápida e leve; é projetada para estender aplicações; e adiciona flexibilidade ao desenvolvimento das aplicações interativas que seriam difíceis de alcançar se fosse utilizada apenas a linguagem NCL [Soares, 2008].

2.5 Usabilidade pra TVD

De acordo com o padrão ISO 9241-11 (1998), o conceito de usabilidade é definido como "o alcance pelo qual um produto e/ou serviço pode ser usado por usuários específicos para atingir metas específicas com eficácia, eficiência e satisfação em um específico contexto de uso". Em aplicações para TVD, este tema deve ser estudado com bastante afinco, principalmente para desenvolvedores e todos os demais interessados em desenvolver sistemas interativos para TVD.

A possibilidade de interagir com aplicações através da televisão por meio do uso do controle remoto, antes usado em um número limitado de funcionalidades, requer a criação de novos estilos de interação. Para atingir tal objetivo, são necessários estudos detalhados do comportamento dos usuários neste contexto, além

de criatividade para a concepção de soluções que satisfaçam requisitos de usabilidade.

Em interfaces consideradas consistentes, o usuário forma um modelo coerente da interface, o que permite generalizar sua experiência em uma interface específica para as demais. Desta forma, com um conjunto pequeno de regras universalmente aplicáveis, o usuário pode utilizar novas interfaces com uma curva de aprendizagem mínima o que aumenta a facilidade de aprendizagem [BARROS, 2006].

Do ponto de vista do usuário final, a consistência traz benefícios diretos, tais como: facilidade de aprendizagem, facilidade de uso, menor taxa de erros, proporciona uma sensação de domínio e melhora a autoconfiança do usuário. Para desenvolvedores de software, observam-se também vantagens importantes: Reduz custos de suporte e de desenvolvimento, aumenta a expectativa de utilização e de vendas, tem potencial de melhorar os aspectos estéticos da interface e traz benefícios do ponto de vista de marketing [NIELSEN, 2002].

Pelo fato de não haver um estilo unificado de interação para aplicações na TVD, este tema está sendo fortemente estudado por empresas e instituições de pesquisa, na busca de padrões que permitam bons níveis de usabilidade. Por outro lado, estudos voltados especificamente aos estudos de dispositivos de interação, como, por exemplo, controles remotos, teclados virtuais, conectividade com celulares e PDA, também permitem novas possibilidades no campo da usabilidade [SILVA *et. al.*, 2007].

2.5.1 Restrições e diferenças diante do computador.

Para os desenvolvedores e designers de novos produtos de software para TVD, sob processo de concepção impõe diversos desafios que precisam ser contornados. O mais preocupante deles é o fato de que as aplicações para TV não podem ser cópias daquelas desenvolvidas no contexto Web ou para o computador pessoal. Pois, em relação a um PC, a TV interativa apresenta uma série de diferenças importantes; tela de menor resolução e com área periférica sujeita a distorções, não oferece rolagem horizontal, propõe dispositivos bastante limitados para o controle da

entrada de dados. O Quadro 2.1 apresenta as principais diferenças técnicas e culturais entre os contextos de aplicações para computador e televisão. Estas características implicam em um grande impacto nas decisões de design de aplicações para estas plataformas.

Quadro 2.1 Televisão x Computador [PICCOLO & BARANAUSKAS, 2006]

Característica	Televisão	Computador
Resolução da tela (quantidade de informação exibida)	Relativamente pobre (640 x 480 pontos)	Varia entre telas médias e grandes (de 800 x 600 a 1280 x 1024 pontos, por exemplo)
Dispositivos de entrada	Controle remoto e, no melhor caso, teclado sem fio	Mouse e teclado, situados em posição fixa
Distância de visualização	Alguns metros	Alguns centímetros
Postura do usuário	Relaxado, reclinado	Ereto, sentado
Ambiente	Sala de estar, quarto (ambientes que sugerem o relaxamento)	Escritório (ambientes que sugerem trabalho)
Oportunidades de integração com outras coisas no mesmo dispositivo	Vários programas de TV	Atividades pessoais, atividades de trabalho
Número de usuários	Normalmente, muitas pessoas estão na sala enquanto a TV está ligada. Uso social e coletivo	Normalmente o uso é individual (poucas pessoas podem ver a tela)
Envolvimento do usuário	Passivo: A emissora seleciona e envia a informação apresentada. O usuário somente a recebe.	Ativo: Usuário comanda e o computador obedece

2.5.2 Padrões adotados pra TVD (*guidelines*)

Como forma de ajudar os desenvolvedores de aplicação nessa nova realidade, algumas organizações, como por exemplo, a BBC (2002), buscam escrever manuais e guias, que são regras e princípios documentados na literatura e que podem ser utilizados como orientação no processo de avaliação e design de aplicações. E se aplicados corretamente, possibilitam o desenvolvimento de interfaces gráficas mais agradáveis e intuitivas. A seguir são listadas seis publicações relevantes no contexto da TVD [BARROS, 2006]:

- BBCi Interactive Television Style Guide [BBCi, 2006];
- A Guide for Digital TV Service Producers [ARVID, 2004];
- Interactive Television Design Guide [LIBERATE, 2002];
- Tiresias.org Television Guidelines [TIRESIAS, 2009];
- Interactive Television Production [GAWLINSKI, 2003]; e
- Case Study: The Usability of Electronic Programme Guides [DALY-JONES, 2003].

Com base nestes guias, a seguir, serão descritas algumas "boas práticas" de desenvolvimento de aplicações para TVD que auxiliam os desenvolvedores na busca por melhores níveis de usabilidade. Estas práticas são divididas nos seguintes aspectos: tamanho e aproveitamento da tela; correto uso de cores; textos; dispositivos de interação; e navegabilidade.

Tamanho da Tela

Na maioria dos televisores que existem atualmente a relação entre largura e altura da tela é 4:3. No entanto, a tendência atual é que cada vez mais aparelhos de TV tenham o formato wide screen, em que a tela é mais retangular e a relação entre largura e altura são 16:9. Como forma de contornar a diferença entre estes dois formatos, utiliza-se normalmente uma borda ao redor do conteúdo ou mesmo através do corte da imagem original [BBCi, 2006].

Com relação à exibição de uma aplicação, ela pode sobrepor o conteúdo televisivo ou redimensioná-lo. Caso se pretenda sobrepor o vídeo com a aplicação, esta deve ocupar as bordas da tela e possuir fundo de tela com um percentual de transparência, com o objetivo de não atrapalhar a visão do telespectador. Por outro lado, se a escolha for redimensionar o vídeo, este não pode ocupar menos de um quarto da tela.

Cores

Entre a televisão e o monitor do computador, existe uma diferença entre seus respectivos *gamute*, que é o conjunto das cores com suporte no dispositivo. Por isso, recomenda-se não utilizar cores muito saturadas ou com luminosidade muito alta [GAWLINSKI, 2003]. Em outras palavras, as cores utilizadas devem ter valores, no sistema RGB, não menores que 16 e não maiores que 236 (na escala de 0 a 256) [BBCi, 2006;TIRESIAS, 2009].

Textos

O conteúdo televisivo tem naturalmente como linguagem principal a exploração da imagem e do áudio, tornando este meio de comunicação tão popular. Logo, os telespectadores não estão acostumados a ler muitos textos, o que torna o uso deste recurso um grande desafio, aliado também ao fato da definição de tela da maioria dos aparelhos de TV ser muito reduzida. Estas limitações fazem com que seja recomendável que o tamanho dos caracteres não seja menor que 18 pontos, sendo ideal o uso de 22 pontos ou maior. Com o objetivo de aumentar a legibilidade dos textos na TV, BECKER et al., (2006) sugere a utilização de 36 pt para Títulos, 20 pt para Menus, 22 pt para textos diversos e 18 pt para Botões (Figura 2.6).

O espaçamento entre linhas e entre caracteres deve também ser maior do que o utilizado para impressão, e a quantidade máxima de palavras em toda a tela não deve passar de 90 [BARROS, 2006].

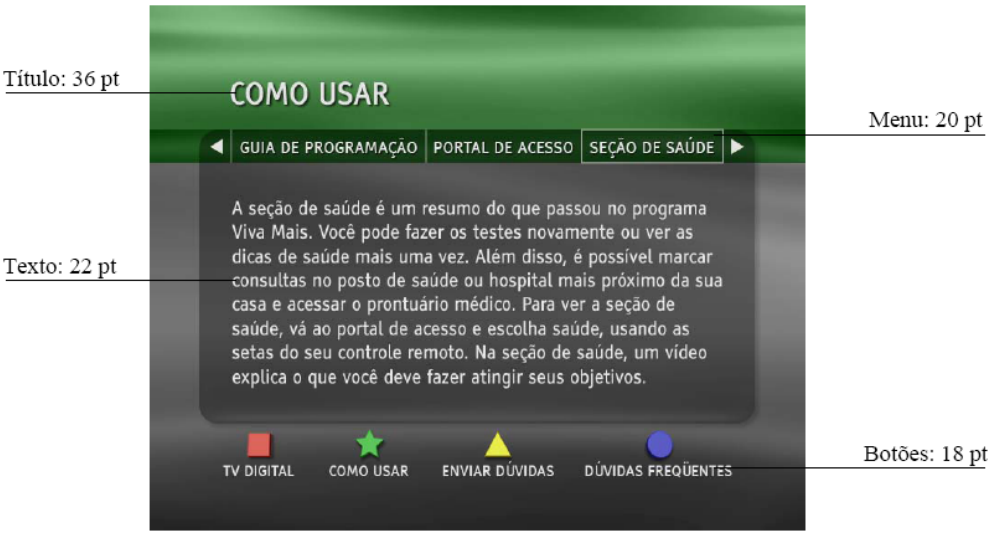


Figura 2.6 Tamanhos de textos sugeridos para a interface do portal [BECKER et al., 2006]

Entre as fontes recomendadas para inserção de textos em aplicações na TVD, estão: Tiresias, Gill Sans, Gill Sans Bold, Zurich, Trebuchet e Univers [TIRESIAS, 2009; BBCi, 2006; LIBERATE, 2002; GAWLINSKI, 2003; ARVID, 2004].O uso de texto claro sobre um fundo escuro também é uma recomendação [TIRESIAS, 2009; ARVID, 2004]. Em Gawlinski (2003) é especialmente recomendado o uso de textos branco sobre fundo azul.

Dispositivo de Interação

No contexto da TVD, um aspecto específico de consistência entre terminais de acesso é a questão de um conjunto mínimo de teclas que deve existir em todos os controles remotos. A escolha do conjunto de botões, pela equipe de desenvolvimento, também afeta a consistência da interface. O conjunto de teclas funcionais deve ser o menor possível, e suas funções devem ser semelhantes, mesmo em diferentes telas da interface. Além disto, o uso de opções claramente disponíveis na tela é recomendável, ao invés da utilização de teclas dedicadas do controle remoto, com funcionalidades específicas [DALY-JONES, 2003]. Estas recomendações evitam que o telespectador tenha que mudar o foco, entre a TV e controle, muitas vezes, pois pode tornar a interação cansativa [GAWLINSKI, 2003; BARROS, 2006].

Navegação

Em termos de navegação, o guia da BBC [BBCi, 2006] apresenta duas possibilidades interessantes: o uso de setas direcionais mais uma tecla de confirmação ou o uso de setas direcionais sem a tecla de confirmação. O cursor utilizado deve ser proeminente e claramente destacado do resto da interface [GAWLINSKI, 2003] e deve-se tomar um cuidado especial quando existem apenas duas opções, pois pode não ficar claro qual item está selecionado e qual não está [ARVID, 2004].

Os menus de opções devem ser circulares, ou seja do primeiro item é possível ir para o último e vice-versa. Também é desejável que as instruções de navegação sejam apresentadas na tela, pois muitos usuários não estão acostumados a estes mecanismos [BBCi, 2006].

De acordo com o estudo desenvolvido por Brecht et. al. (2005), durante o levantamento de requisitos para uma aplicação jornalística, são confirmadas estas necessidades. Durante entrevistas em grupos focais, os usuários destacaram que a facilidade de interação está diretamente relacionada com o uso efetivo e a satisfação. Outro requisito que foi levantado é referente à separação clara entre elementos de conteúdo e navegação.

2.6 Considerações Finais

Este capítulo teve como objetivo alertar e instruir os programadores que se interessam pelo desenvolvimento de aplicações para TVD. Primeiramente, é preciso lembrar que parte do público alvo desta nova geração de aplicações é formada basicamente por pessoas que não possuem computador e que estão acostumadas em apenas mudar o volume e o canal da TV.

3 Frameworks de desenvolvimento

Atualmente no contexto da Engenharia de Software, o reuso tem sido uma prática constante no processo de melhoraria da qualidade dos artefatos, otimização do tempo e redução do custo de desenvolvimento. O reuso pode ser explorado das seguintes formas: concentrar-se nos modelos de arquitetura para solucionar problemas semelhantes; fazer uso de padrões de projetos para solucionar problemas de implementação; e reutilizar componentes de software já desenvolvidos, que é a estratégia mais comum [SILVA, 2006].

Segundo Johnson et al. (1995), um framework é um conjunto de classes cooperativas que compõem um modelo reutilizável para um modelo específico de software. Os frameworks promovem o reuso de código e de projeto fazendo uso dos padrões de projetos para a obtenção de estruturas que sejam mais extensíveis e facilmente modificadas [SILVA, 2006]. Nas subseções seguintes serão apresentados os aspectos gerais dos *frameworks*, a descrição das etapas do processo de concepção mostrando técnicas e padrões usualmente utilizados, além de detalhar os benefícios e as formas de utilização.

3.1 Aspectos Gerais

Para prover as funcionalidades requeridas por um domínio de aplicações, um *framework* engloba as funcionalidades comuns às aplicações deste domínio, porém, para ser útil no desenvolvimento de aplicações, deve possuir a capacidade de adaptação. Um *framework* provê uma orientação arquitetural ao desenvolvedor ao apresentar um modelo com classes abstratas e definir suas responsabilidades e colaborações. O desenvolvedor o customiza para uma aplicação particular, pois este fornece uma implementação para as funções básicas e constantes e inclui um mecanismo para permitir que o desenvolvedor se conecte às diversas funções, as estenda ou implemente aquelas que ainda não estão presentes.

Os *hot spots* configuráveis garantem que os frameworks possuam a capacidade de adaptação, para que este atue de diversas formas não previstas, enquanto preserva outras partes. São as partes de um framework que estão abertas à extensão e

customização [SILVA, 2000] e especificam aspectos do framework que não podem ser (ou não se quer que sejam) totalmente antecipados [SILVA e OLIVEIRA, 2006].

3.2 Classificação dos Frameworks

Os frameworks podem ser classificados em 2 tipos, quanto a forma de utilização (Quadro 5.1) e quanto a finalidade do uso (Quadro 5.2). Quando se refere aos problemas que os frameworks buscam solucionar podem ser classificados em horizontais, verticais ou de infra-estrutura. Os frameworks horizontais destinam-se a resolver apenas uma parte do problema da aplicação e por este fato conseguem atender a uma grande parcela das mesmas. Já os verticais encapsulam os conhecimentos aplicáveis às aplicações de um domínio particular de problema. Destinam-se a resolver todo ou boa parte do problema, podendo gerar aplicações inteiras [SILVA, 2000 ; SILVA e OLIVEIRA, 2006]. Os de infra-estrutura, no entanto buscam solucionar problemas ao nível de infra-estrutura (e não de aplicação).

Quadro 3.1 Características dos tipos dos frameworks com relação a forma de utilização

Tipo	Característica
Caixa Branca	• Foca na herança de classes; • Estende ou modifica funcionalidades pela definição de subclasses e pela sobreposição de métodos;
Caixa Preta	• Foca na composição dos componentes; • Usa a funcionalidade já presente no framework; • As entidades internas do framework não podem ser vistas ou alteradas; • As instanciações e composições feitas determinam as particularidades da aplicação;
Híbrido (Caixa Cinza)	• A maioria dos frameworks apresenta uma organização híbrida; • Também são conhecidos por frameworks de caixa-cinza; • Possuem funcionalidades prontas e aquelas que podem ser criadas ou alteradas • A grande maioria deles são de caixa-branca com algumas funcionalidades já prontas (caixa-preta);

Os frameworks podem ser classificados, também, por sua forma de utilização, caixa branca, caixa preta ou híbrido. Os frameworks de caixa branca são focados na herança de classes, estendem ou modificam uma funcionalidade pela definição de subclasses com sobreposição de métodos [SILVA, 2000]. Já nos caixa preta, os desenvolvedores devem saber apenas quais objetos estão disponíveis e as regras para combiná-los, estes objetos não podem ser vistos ou alterados. Neste tipo de framework o reuso se dá pelas conexões entre os componentes, não havendo preocupação em saber como eles realizam as tarefas individuais, as instanciações e composições feitas determinam as particularidades da aplicação [SILVA, 2000]. A maioria dos frameworks apresenta uma organização híbrida, são também conhecidos por frameworks de caixa-cinza, pois existem funcionalidades prontas e aquelas que podem ser criadas ou alteradas [Silva, 2000 ; SILVA e OLIVEIRA, 2006].

Quadro 3.2 Características dos tipos dos frameworks com relação a finalidade

Tipo	Características
Suporte	• São raros • Provê serviços de nível de infra-estrutura (e não de aplicação) · Acesso a arquivos · Computação distribuída · Device drivers
Aplicação	• Também chamado de framework horizontal • Encapsula conhecimento ("expertise") • Aplicável a uma vasta gama de aplicações • Resolve apenas uma fatia do problema da aplicação • **Exemplo:** framework para construção de interface GUI
Domínio	• Também chamado de framework vertical; • Encapsula conhecimento ("expertise"); • Aplicável a aplicações pertencendo a um domínio particular de problema; e • Resolve boa parte da aplicação. • **Exemplo:** framework para construir aplicações de controle de manufatura

3.3 Projeto e Desenvolvimento de Frameworks

Um framework é uma abstração de um domínio de aplicações, especializada em aplicações deste domínio. A principal característica ao se desenvolver um framework é a generalidade em relação a conceitos e funcionalidades do domínio tratado. Além disso, é fundamental que a estrutura produzida seja flexível, isto é, apresente as características de alterabilidade e extensibilidade [SILVA e PRICE, 1997].

Existem atualmente poucas metodologias para o desenvolvimento de frameworks, as mais conhecidas são "Projeto Dirigido por Exemplo" (*example-driven design*) e "Projeto Dirigido por Pontos de Flexibilização" (*hot spot driven design*). O primeiro enfatiza a análise da maior quantidade de exemplos do domínio tratado, para obtenção de conhecimento do domínio, para que assim possa identificar as especificações e generalidades de cada aplicação. Já o segundo enfatiza a busca por pontos de flexibilização (*Hot Spots*), que segundo [SILVA, 2000] são partes da estrutura do *framework* que devem ser mantidas flexíveis, com o objetivo de possibilitar a sua adaptação a diferentes aplicações do domínio. Ambos seguem os seguintes passos para a produção do *framework*:

I. Análise do domínio (principalmente das aplicações já desenvolvidas para o domínio);

II. Definição de uma hierarquia de classes que possa ser especializada para as aplicações do domínio (modelagem do framework); e

III. Teste do framework através do desenvolvimento de exemplos.

A análise do domínio provê ao desenvolvedor do *framework* entendimento detalhado dos conceitos do domínio, assim como entendimento detalhado sobre as funcionalidades e *hot spots* que ele deve prover. Um detalhe muito importante deve ser observado durante este processo, o desenvolvimento de uma aplicação utilizando o *framework* não pode demandar mais esforço do que seu desenvolvimento sem seu uso.

À medida que se obtém novos conhecimentos sobre o domínio de aplicação, também é necessária a manutenção no *framework*, para fins de refinamento. Um *framework* está sempre em desenvolvimento. Isto faz com que sua estrutura passe por um processo contínuo de evolução iterativa, causada pela busca de adaptação da estrutura de classes aos aspectos de generalidade, extensibilidade e flexibilidade. Seus usuários são os principais responsáveis pelo o amadurecimento do *framework*, englobando através de suas experiências novas funcionalidades e abordagens.

Durante a fase de projeto do framework, surgem algumas questões chaves [SILVA, 2000]:

I. **Quais classes –** quais classes o usuário do framework deve desenvolver e quais ele pode reutilizar;

II. **Quais métodos –** ao gerar uma classe concreta a partir de uma classe abstrata do framework, deve-se definir que métodos poderão ser reutilizados completamente, quais devem ser definidos e quais devem ser sobrepostos; e

III. **O que os métodos fazem –** esta questão relaciona-se ao objetivo da aplicação. Deve-se definir a responsabilidade de cada método para que este produza o comportamento esperado e realize a interação entre diferentes objetos.

Nesta questão é relevante delimitar três categorias de classes: as concretas, as abstratas e as que necessitam serem criadas para gerar a aplicação. Os métodos por sua vez também são divididos em três categorias, métodos Abstratos, Templates e Bases [SILVA, 2000]. Um método Abstrato tem apenas a sua assinatura definida, sendo que seu algoritmo deve ser definido nas subclasses. Um método Template é um método que possui sua estrutura definida por chamadas a outros métodos (abstratos). Já os métodos Base são métodos que têm seu corpo claramente definido, não sendo preciso alterá-lo, mas oferece a possibilidade de sobreposição.

Frameworks são um tipo complexo de software, e necessitam fazer uso de modelos e padrões já consagrados na literatura que detenham de uma boa

documentação para assegurar a corretude e eficiência de suas implementações. Além disto, é importante que se tenha uma boa documentação, ela deve prover informações suficientes para auxiliar as tarefas de adaptação e evolução, tais como, seu propósito, como utilizar o *framework* e suas características estruturais. Com o objetivo de suprir as deficiências em termos da documentação é interessante o uso de modelos gráficos adequados, que abrangem conceitos do contexto de *frameworks*. Nas subseções seguintes serão detalhados os principais padrões de projetos utilizados na concepção de *frameworks* e um modelo gráfico adequado para complementar a sua documentação.

3.3.1 Padrões Utilizados

Segundo Silva (2006), os padrões de projeto (*design patterns*) [JOHNSON, 1995] utilizados durante a modelagem e implementação dos *frameworks*, são referências importantíssimas no desenvolvimento de aplicações orientadas a objetos. A motivação para se utilizar padrões de projeto é o desejo de reuso de modelos de qualidade com uma boa fonte de documentação. Um padrão de projeto descreve um problema, sua solução, quando aplicar a solução e suas conseqüências. Os padrões de projeto adicionam flexibilidade ao projeto de um *framework* por portar a solução de um problema comum que pode ser adaptado. Abaixo serão descritos os principais padrões utilizados no projeto de *frameworks* e suas respectivas utilizações, baseados na descrição apresentada em [JOHNSON, 1995]

Abstract Factory

Abstract Factory é um padrão que permite a criação de famílias de objetos relacionados ou dependentes, através de uma única interface e sem que a classe concreta seja especificada. Define-se um responsável por criar objetos quando existem considerações especiais, tais como uma lógica de criação complexa, ou para separar as responsabilidades de criação de objetos, para melhorar a coesão. O objeto responsável inclui métodos utilitários para se obter uma instância do objeto associado à fábrica, e esta se encarrega do executar o carregamento dinâmico da mesma.

A utilização deste padrão no desenvolvimento de frameworks é interessante, pois com ele pode-se facilmente incluir novas classes concretas no modelo, sem que o usuário da *factory* tenha que saber que outra implementação foi incluída.

Singleton

Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto de acesso global e único ao objeto. Neste contexto, este padrão foi usado para tornar a interface de cada módulo da arquitetura única e acessível de forma global. Muitos projetos necessitam que algumas classes tenham este aspecto. A solução proposta pelo padrão *Singleton* é tornar a própria classe responsável pela sua única instância. A classe que implementa o padrão *Singleton* garante o acesso à sua instância e ainda intercepta as requisições para criação de novos objetos, garantindo que nenhuma outra instância seja criada. No desenvolvimento de *frameworks* este padrão é extremamente útil, pois assegura o acesso de forma única e global a cada módulo da arquitetura.

Strategy

O padrão *Strategy* é utilizado quando é necessário ter diferentes implementações, políticas, estratégias para um mesmo problema. A solução está em se definir cada estratégia em uma classe separada, com uma interface comum. No projeto e desenvolvimento de *frameworks* este padrão pode ser utilizado em conjunto com o *Abstract Factory*, onde uma classe pode ser responsável por criar todas as estratégias necessárias para a aplicação. A fábrica pode ler o nome da classe que contém a estratégia a partir de uma propriedade externa e instanciá-la. O padrão *Strategy* oferece suporte a mudanças dinâmicas de políticas, ou seja, algoritmos variantes. Este padrão permite que o algoritmo possa variar independentemente dos clientes que o utilizam.

Composite

Composite é um padrão de projeto de software utilizado para representar um objeto que é constituído pela composição de objetos similares a ele. O padrão resolve este problema definindo classes para objetos compostos e atômicos que

implementem uma mesma interface. Isto faz com que o objeto composto possua um conjunto de outros objetos que estão na mesma hierarquia de classes a que ele pertence. A conseqüência disto é que os elementos contidos em um objeto composto são tratados como se fosse um único objeto. Desta forma, todos os métodos comuns às classes que representam objetos atômicos da hierarquia poderão ser aplicáveis também ao conjunto de objetos agrupados no objeto composto. Normalmente é utilizado para representar listas recorrentes - ou recursivas - de elementos, e também no desenvolvimento de componentes de interfaces gráficas.

Observer

O padrão *Observer* é aplicado quando diferentes tipos de objetos assinantes estão interessados nas mudanças de estado ou nos eventos de um objeto publicador e cada um quer reagir de sua própria maneira exclusiva quando o publicador gerar um evento. Sua implementação define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda o estado, todos seus dependentes sejam notificados e atualizados automaticamente. A solução consiste em definir uma interface "ouvinte" (*listener*) que será implementada pelos objetos assinantes.

Quando uma mudança a um objeto requer mudanças a outros e você não sabe quantos outros objetos devem mudar ou quando um objeto deve ser capaz de avisar outros sem fazer suposições sobre quem são os objetos. Em outras palavras, sem criar um acoplamento forte entre os objetos. Sua utilização fornece uma implementação muito flexível de acoplamento de abstrações. O uso predominante desse padrão é o tratamento de eventos de elemento de janela de GUI (*Graphical User Interface*).

Template Method

O padrão *Template Method* auxilia na definição de um algoritmo com partes, do mesmo, definidas por métodos abstratos. As subclasses devem se responsabilizar por estas partes abstratas, deste algoritmo, que serão implementadas, possivelmente de várias formas, ou seja, cada subclasse irá implementar a sua necessidade e oferecer um comportamento concreto construindo todo o algoritmo. Assim, as subclasses podem sobrepor os métodos variantes para adicionar seu próprio comportamento exclusivo em pontos de variabilidade.

3.3.2 Modelagem

O processo de modelagem tem por objetivo complementar a documentação do *framework*, sendo de extrema importância em projetos que carecem de documentação. No contexto de *frameworks*, a modelagem deve abranger tanto as visões estáticas quanto as dinâmicas, havendo assim técnicas de modelagem específicas para cada visão. É necessário que o modelo proposto englobe os diferentes tipos de classes e métodos dispostos no projeto de *frameworks*.

As classes abstratas representam um conjunto de conceitos gerais do domínio de aplicação, e a criação e os relacionamentos de suas subclasses têm como conseqüência a especificação de novas aplicações. Com isto é necessário que se tenha uma distinção gráfica entre as classes abstratas e concretas. Silva e Price (1997) sugere a seguinte notação para modelagem de *frameworks* (Figura 3.1).

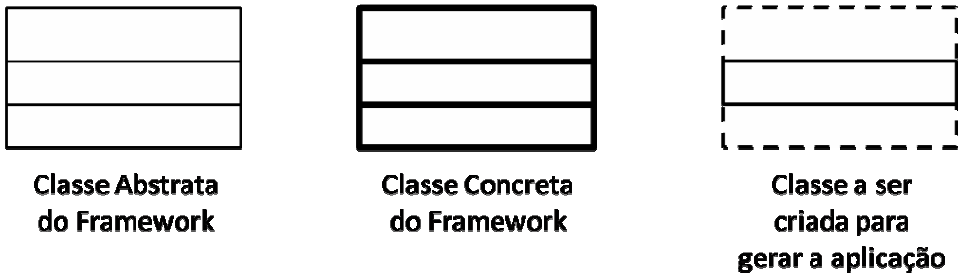


Figura 3.1 Distinção gráfica das classes de um framework [SILVA e PRICE, 1997]

Este modelo especifica a obrigatoriedade das classes e mostra quais estruturas podem ser estendidas a fim de especificar o comportamento de uma aplicação do domínio. A criação das classes por parte dos usuários poderá assumir uma das seguintes situações exemplificadas na Figura 3.2.

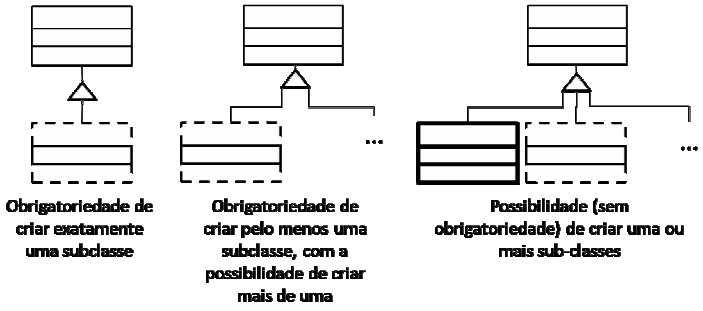


Figura 3.2 Possíveis situações das subclasses criadas pelos usuários do *framework* [SILVA e PRICE, 1997]

Esta técnica de modelagem tem por objetivo complementar as metodologias de desenvolvimento descritas anteriormente e visam auxiliar os aspectos de registros, classificação e organização, das informações do projeto. Este processo é extremamente útil durante a fase de desenvolvimento e manutenção, e serve como documentação para o entendimento básico das estruturas do projeto para os usuários do *framework*.

3.4 Considerações Finais

O uso de um *framework* consiste na geração de aplicações a partir deste. A motivação de seu uso provém da característica de melhoria da qualidade e aumento da produtividade, devidos à reutilização de código e projeto. Entretanto para se usar um *framework* deve-se entender a sua estrutura e possivelmente estende-la, de modo que os requisitos da aplicação sejam atendidos.

Outra questão envolvida no uso de um *framework* é o fato de que o esforço despendido para entendê-lo e utilizá-lo na implementação de um aplicativo deve ser menor do que se este aplicativo fosse desenvolvido sem o seu uso. Desta forma o framework deve oferecer recursos que facilitem desde a sua compreensão inicial, de modo superficial, até elementos que permitam um estudo mais profundo do projeto do *framework*. Conclui-se que em via de regra, a adoção de mecanismos de mais alto nível de abstração reduzem a necessidade de análise de código, demandando menores esforços para a utilização do *framework*.

4 Trabalhos Relacionados

Atualmente existem alguns esforços que buscam facilitar o desenvolvimento de aplicações declarativas para TV Digital. Nesta seção serão descritos de forma resumida alguns destes esforços, e posteriormente é feita uma análise comparativa, com o intuito de identificar algumas características que podem ser incorporadas pelo MoonDo, além destacar de algumas falhas para que estas possam ser evitadas.

4.1 Composer

O Composer é uma ferramenta de autoria hipermídia voltada para o desenvolvimento de programas audiovisuais interativos em NCL, desenvolvido pelo Laboratório TeleMídia da Pontifícia Universidade Católica do Rio de Janeiro (PUCRio). No Composer, abstrações visuais são definidas com o objetivo de diminuir parte da complexidade de se programar em NCL. Para isso ele usa o conceito de visões onde um programa poderá ser construído sob uma dessas visões (Figura 4.1): Estrutural, Temporal, Leiaute ou Textual. Essas visões funcionam de maneira sincronizada a fim de oferecer um ambiente integrado de autoria [GUIMARAES, R. L, 2007].

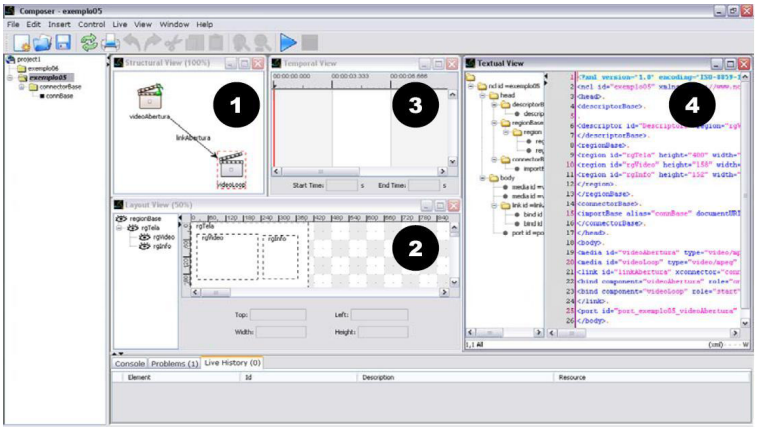


Figura 4.1 Interface do Composer (Janela 1 – Visão Estrutural; Janela 2 – Visão Leiaute; Janela 3 – Visão Temporal; e Janela 4 – Visão Textual) [GUIMARAES, R. L, 2007]

Apesar da pluralidade das visões ser um recurso interessante para o desenvolvimento de aplicação e sua sincronização facilitar a manipulação dos documentos NCL, a ferramenta apresenta algumas deficiências impactantes no

processo de desenvolvimento. Inicialmente podemos identificar como deficiências, a falta de suporte à linguagem Lua, problemas com as disposições dos componentes na interface gráfica, pouca especificação dos erros encontrados pela ferramenta, pouco poder de manipulação dos arquivos do projeto e o baixo desempenho [OLIVEIRA, 2009].

4.2 GingaWay

O Gingaway (Figura 4.2) é uma ferramenta especializada no desenvolvimento de aplicações interativas para a TV digital brasileira. É uma extensão para a plataforma Eclipse [ECLIPSE, 2009] que facilita na codificação de aplicações em Ginga-NCL com suporte de scripts em Lua. O GingaWay visa propor um ambiente de desenvolvimento integrado com uma gama de ferramentas que auxiliem o desenvolvimento das aplicações declarativas. Como o GingaWay estende alguns plugins do Eclipse, tais como o NCL Eclipse e o Lua Eclipse, ele engloba suas funcionalidades [BELTRÃO FILHO, 2008].

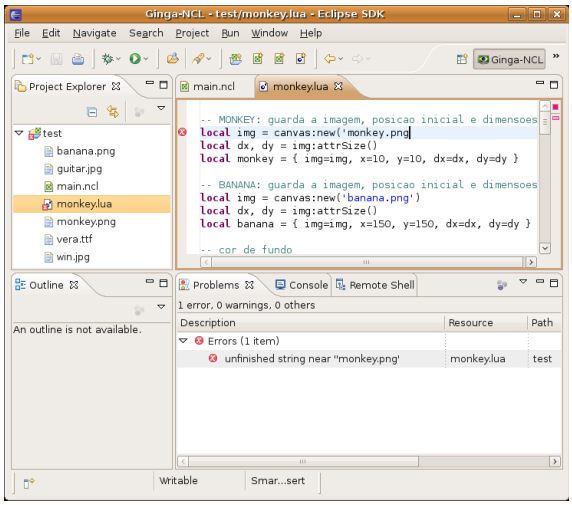


Figura 4.2 Interface do GingaWay [BELTRÃO FILHO, 2008]

Além das funcionalidades incorporadas através dos componentes estendidos, ele dispõe de assistentes de desenvolvimento e configuração e dá suporte a utilização dos demais simuladores conhecidos pela comunidade. Porém, suas soluções fornecem ambiente integrado com ferramentas CASE (*Computer-Aided Software Engineering*) para este domínio, sem focar em recursos essenciais para o desenvolvimento, tais como componentes gráficos e dispor de uma orientação

arquitetural. O GinagWay ainda carece de documentação para ser utilizado de forma eficiente para o desenvolvimento de aplicações.

4.3 LuaComp

Ferramenta de autoria para aplicações em Ginga-NCLua para a TV Digital interativa do Brasil. O LuaComp (Figura 4.3) é uma ferramenta de autoria que possibilita ao usuário a rápida criação de aplicações, explorando funcionalidades do LuaOnTV [SOUZA JÚNIOR, 2009], um framework para utilização dos componentes gráficos sob o paradigma da programação orientada a objetos. Esta ferramenta apresenta o conceito de visões, assim como o Composer, é baseado no conceito WYSIWYG (*What You See Is What You Get*) com o objetivo de auxiliar a orientação dos componentes gráficos das telas da aplicação, além de permitir a criação e utilização de *templates* em arquivos XML [SOUZA JÚNIOR, 2009].

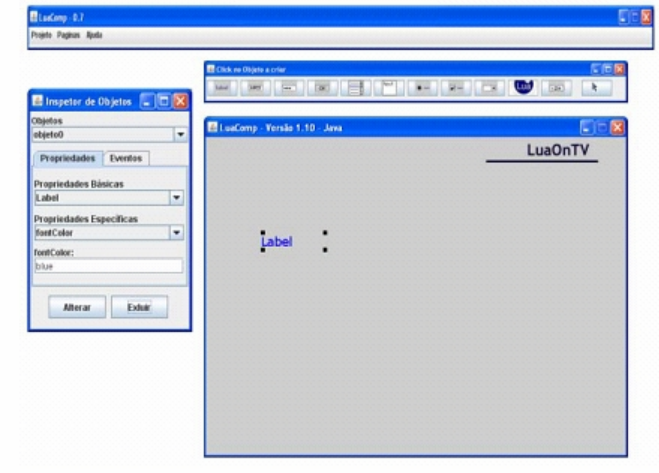


Figura 4.3 Interface do LuaComp [SOUZA JÚNIOR, 2009]

O LuaComp dispõe de uma interface gráfica para geração de códigos das aplicações (ncl, lua e xml), ainda fornece uma pequena quantidade de componentes para uma ferramenta que usa como principal característica o desenvolvimento gráfico no formato *drag and drop* – arrastar e soltar – que consiste no ato de selecionar componentes pré-estabelecidos e arrumar manualmente (através do mouse) no leiaute da aplicação. Alguns componentes gráficos utilizados não são indicados pelos guias de usabilidade [BECKER et al., 2006] para o contexto da TVD, mesmo que estes sejam adequados para o contexto web, como por exemplo, o *checkbox*. Mesmo

possuindo um recurso interessante de exportação das aplicações para o formato XML, que favorece a integração com outras ferramentas de autoria, este ambiente ainda está estritamente ligado ao LuaOnTV. Isto faz com que a integração desta ferramenta com outras soluções seja limitada pelo uso do LuaOnTV. Assim como outras ferramentas aqui descritas o LuaComp não facilita a integração de outros simuladores e não faz o controle dos eventos aplicados nos componentes gráficos, impactando no processo de desenvolvimento.

4.4 NCL Eclipse

O NCL Eclipse (Figura 4.4) é um plug-in para a IDE (*Integrated Development Environment*) Eclipse que oferece diversas funcionalidades centradas nas necessidades dos autores NCL, tais como a sugestão de código automática e contextual, validação de documentos NCL, coloração de elementos XML e de palavras reservadas da linguagem. Auxilia o desenvolvimento textual, enquanto que as abstrações gráficas são deixadas um pouco de lado, mas podem futuramente ser incorporadas. O Eclipse se mostra uma ferramenta natural para integrar as diversas necessidades, haja vista ser um ambiente que também oferece *plug-ins* para Lua – como o LuaEclipse [LUAECLIPSE, 2008] – e ser um dos mais utilizados para o desenvolvimento em Java NCL [AZEVEDO et al.,2008]. A Figura 4.4 mostra a interface do ambiente de desenvolvimento integrado com o NCL Eclipse. As Janelas A,B e C mostrar a sugestão automática de códigos enquanto que a Janela D mostra como é feito a orientação das regiões na tela, pelo NCL Eclipse.

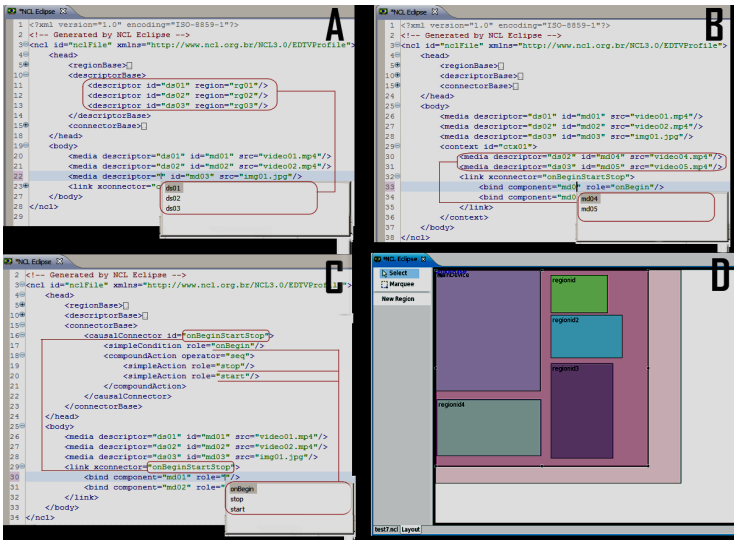


Figura 4.4 Interface do NCL Eclipse [AZEVEDO et al.,2008]

Por apresentar diversas facilidades para a edição dos documentos NCL e por integrar-se eficientemente com o LuaEclipse para suporta a linguagem Lua, esta ferramenta usualmente é utilizada como base para as demais soluções que buscam facilitar o desenvolvimento de aplicações declarativas. Porém por não definir uma orientação arquitetural e nem dispor de componentes gráficos para serem utilizados nas aplicações, esta ferramenta usada isoladamente se torna ineficiente para a produção de aplicações, tornando o desenvolvimento custoso em relação ao tempo e qualidade do processo.

4.5 TVision

TVision (Figura 4.5) é uma ferramenta que é executada em browsers via web, utilizando tecnologias do tipo RIA (*Rich Internet Application*), para fornecer uma interface que amigável e intuitiva. Tem por objetivo auxiliar o desenvolvimento de aplicações interativas, fazendo uso do formato *drag and drop*. Esta ferramenta visa auxiliar o desenvolvimento de aplicações declarativas no SBTVD, gerando código para serem executadas sobre o ambiente GINGA-NCL (aplicações escritas em NCL e com o suporte de uma linguagem de script Lua). O código gerado é baseado na orientação dos componentes gráficos nas telas do TVision via browser [OLIVEIRA, 2009].

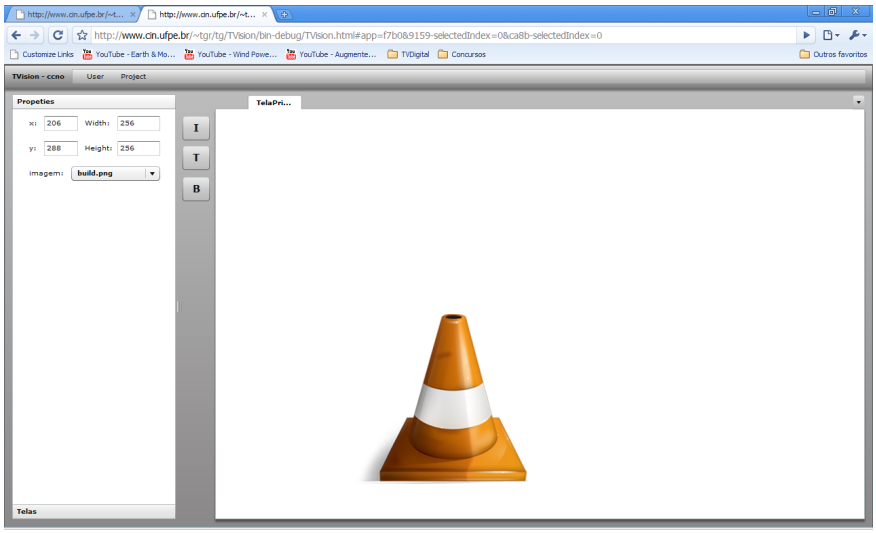


Figura 4.5 Interface do TVision [OLIVEIRA, 2009]

Mesmo dispondo de uma interface gráfica para geração de códigos das aplicações, esta ferramenta ainda está em fase de amadurecimento, pois ainda fornece uma pequena quantidade de componentes para uma ferramenta que usa como principal característica o desenvolvimento gráfico no formato *drag and drop*. O TVision ainda carece de documentação e de ferramentas e processos de testes e validações das aplicações, pois não apresenta uma ferramenta de simulação integrada, além de não dispor de um ambiente para desenvolvimento *offline*, obrigando ao desenvolvedor ter acesso a internet para ele possa desenvolver suas aplicações.

4.6 Considerações Finais

Através da análise destes trabalhos foi possível identificar algumas características que puderam ser incorporadas pelo MoonDo, além de tomar conhecimento de algumas falhas para que estas pudessem ser evitadas. Abaixo segue um quadro (Quadro 4.1) comparativo a fim de evidenciar os principais pontos positivos e negativos dos trabalhos relatados.

Quadro 4.1 Analise comparativa dos trabalhos relacionados

Ferramenta	Pontos Positivos	Pontos Negativos
Composer	▪ Edição de documentos NCL; ▪ Visões Múltiplas; ▪ Manipulação gráfica de componentes; ▪ Integração com ferramentas de simulação; ▪ Geração de código;	▪ Não possui edição de documentos Lua; ▪ Simulador não suporta Lua; ▪ Não é possível integrar novas ferramentas de simulação; ▪ Não provê uma orientação arquitetural ao desenvolvedor;
GingaWay	▪ Edição de documentos NCL; ▪ Edição de documentos Lua; ▪ Integração com ferramentas de simulação; ▪ Configuração de novas ferramentas de simulação;	▪ Não possui manipulação gráfica de componentes; ▪ Não possui componentes de interface gráfica configuráveis; ▪ Não provê uma orientação arquitetural ao

		desenvolvedor; ▪ Pouca documentação;
LuaComp	▪ Edição de documentos NCL; ▪ Edição de documentos Lua; ▪ Visões Múltiplas; ▪ Manipulação gráfica de componentes, para a criação das interfaces; ▪ Geração de código; ▪ Componentes de interface gráfica configuráveis; ▪ Integração com ferramentas de simulação; ▪ Facilita a integração com outras ferramentas (exporta aplicação para xml);	▪ Falta de qualidade dos componentes de interface gráfica; ▪ Não é possível integrar novas ferramentas de simulação; ▪ Não provê uma orientação arquitetural ao desenvolvedor; ▪ Pouca documentação;
NCL Eclipse	▪ Edição de documentos NCL; ▪ Edição de documentos Lua; ▪ Integração com ferramentas de simulação;	▪ Não possui manipulação gráfica de componentes; ▪ Não possui componentes de interface gráfica configuráveis; ▪ Não provê uma orientação arquitetural ao desenvolvedor;
TVision	▪ Edição de documentos NCL; ▪ Edição de documentos Lua; ▪ Manipulação gráfica de componentes, para criação das interfaces; ▪ Componentes de interface gráfica configuráveis; ▪ Ambiente de desenvolvimento online; ▪ Geração de código;	▪ Poucos componentes de interface gráfica disponíveis; ▪ Não possui integração com ferramentas de simulação; ▪ Não provê uma orientação arquitetural ao desenvolvedor; ▪ Não possui ambiente de desenvolvimento offline; ▪ Pouca documentação;

5 Desenvolvimento do Framework MoonDo

A iniciativa do desenvolvimento do *framework* MoonDo surgiu a partir de uma dissertação de mestrado [MONTEIRO, 2009]. Durante o desenvolvimento de um portal interativo na TVD integrado a um Sistema de Gestão de Aprendizagem, percebeu-se que alguns componentes foram reutilizados. A partir dai os esforços se concentraram em estabelecer uma metodologia que permitisse a formalização e amadurecimento do *framework* de desenvolvimento de aplicações para TVD, apresentado neste trabalho.

Neste processo, o levantamento de requisitos, a análise de riscos e a identificação de oportunidades devem considerar: usuário, contexto social e tecnológico [GOMES et al., 2008]. É evidente que a forma de interação com a TV é diferente do que se está acostumado em frente ao computador. Por esta razão, para que o *framework* MoonDo seja realmente eficiente e eficaz, foi necessário primeiramente compreender os principais conceitos relacionados ao desenvolvimento de aplicações para TVD. Neste capítulo, serão apresentados os requisitos que o MoonDo irá atender, a arquitetura base do sistema e detalhes de implementação da ferramenta.

5.1 Concepção do MoonDo

Segundo Johnson et al. (1995), um *framework* é um conjunto de classes cooperativas que compõem um modelo reutilizável para um modelo específico de software. Os frameworks promovem o reuso de código e de projeto fazendo uso dos padrões de projetos para a obtenção de estruturas que sejam mais extensíveis e facilmente modificadas [SILVA, 2006]. O framework MoonDo foi idealizado para fornecer um modelo arquitetural modularizado, estruturado e bem definido, para que seus componente (gráficos e funcionais) possam ser reutilizados e parametrizados de acordo com as características da aplicação que está sendo desenvolvida. Entre as principais funcionalidades deste framework estão: tratamento das ações dos usuários; disponibiliza funções de escrita e leitura de arquivos (metadados); dispõe de componentes de interface gráfica; e permite que o

desenvolvedor modifique ou crie novos componentes gráficos e funcionalidades utilitárias (candidatas ao reuso).

O MoonDo é classificado como um *framework* de aplicação [SILVA, 2000], pois se concentra no domínio de aplicações para TV Digital. A concepção do MoonDo foi orientada através da metodologia, Projeto Dirigido por Exemplo, onde a abstração do domínio é obtida a partir da análise e generalização de exemplos concretos [SILVA e PRICE, 1997], passando pela seguinte seqüência de etapas:

1. Análise do domínio (aplicações, padrões e guias existentes);
2. Definição da uma hierarquia de classes que possa ser especializada para abranger os exemplos; e
3. Teste do *framework* através do seu uso no desenvolvimento de exemplos (desenvolvimento de um objeto de aprendizagem).

Após a análise de algumas aplicações existentes neste domínio, percebeu-se que não havia um modelo arquitetural definido para o desenvolvimento destas aplicações. Desta forma buscou-se identificar as principais necessidades das aplicações para que o MoonDo pudesse fornecer todos os elementos básicos para o desenvolvimento das mesmas. A arquitetura do MoonDo foi idealizada como resultado desta fase de análise. Inicialmente foi definido que o MoonDo teria como foco a linguagem Lua, bastando ao NCL a responsabilidade de iniciar as aplicações desenvolvidas em Lua (Figura 5.1).

Figura 5.1 Relacionamento entre as entidades e o framework MoonDo

A Figura 5.2 mostrar o modelo arquitetural definido, seguindo o modelo proposto por Silva [SILVA e PRICE, 1997], contendo os relacionamentos entre as entidades, para que seus componentes (gráficos e funcionais) possam ser reutilizados e parametrizados de acordo com as características da aplicação que está sendo desenvolvida. Sob o ponto de vista da forma de utilização, ele é um *framework*

híbrido, pois possui entidades com característica de "caixa preta" (quando abstraem os detalhes internos de implementação) e "caixa branca" (quando utiliza herança e sobreposição de métodos para estender ou modificar uma funcionalidade) [SILVA, 2000].

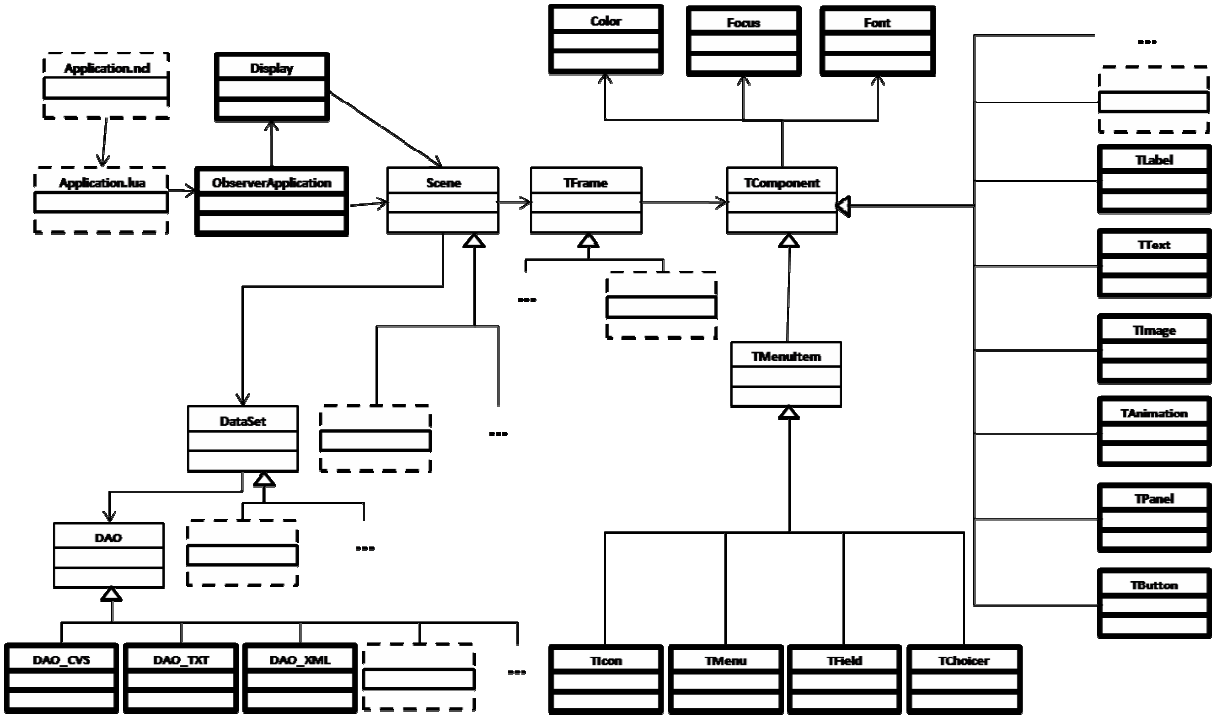


Figura 5.2 Representação arquitetural do *framework* MoonDo

Abaixo, segue a descrição das principais entidades e a sua classificação em relação à forma de utilização no desenvolvimento:

- **Application.ncl** – entidade que necessita ser criada para gerar a aplicação, responsável por definir a região e inicializar a aplicação no documento NCL;
- **Application.lua** – entidade que necessita ser criada para gerar a aplicação, ou seja, a aplicação propriamente dita e, mídia executada pelo documento NCL. Nesta enteidade importa-se e inicializa as *Scene's* da aplicação;
- **ObserverApplication** – entidade "caixa preta" responsável por repassar os eventos do controle remoto para as entidades competentes. O desenvolvedor faz uso deste componente para manipular a seleção das telas da aplicação;

- **Scene** – entidade "caixa branca" responsável por agrupar as telas (TFrame) dependentes, compartilhando dados e funções. O desenvolvedor estende este componente a fim de definir funções e dados comuns a um conjunto de telas;
- **TFrame** – entidade "caixa branca" que representa uma única tela da aplicação. O desenvolvedor estende este componente para que possa adicionar os componentes gráficos e definir os eventos do controle remoto;
- **DataSet** – entidade "caixa branca" que provê uma interface de acesso e manipulação de dados externos para diferentes estruturas de arquivos. O desenvolvedor estende este componente com o objetivo de definir quais os dados que serão lidos, e implementar funções auxiliares para a manipulação desses dados; e
- **TComponente** – entidade "caixa branca", porém suas sub-classes possuem abordagem híbrida. Ela representa a abstração de um componente gráfico, por isso todo componente gráfico precisa estendê-la. O desenvolvedor pode fazer uso dela para definir novos componentes; porém devido à praticidade, suas subclasses gráficas, já definidas no *framework*, são comumente instanciadas.

Com relação à interface com o usuário, foram idealizados componentes gráficos comuns para as aplicações de TVD. Pelo fato do controle remoto limitar a interação, muitas aplicações fazem uso de menus para organizar hierarquicamente suas funcionalidades. Com base nas características peculiares desta plataforma, foram desenvolvidos componentes gráficos genéricos e intuitivos para dar suporte a esta forma de organização, sem perder a qualidade do *design*. Alguns componentes são de caráter abstrato, por exemplo, o TMenuItem, que indica que suas sub-classes podem representar itens em uma instância de menu, e o TAnimation que indica a presença de uma animação na tela. Além dos componentes arquiteturais e gráficos, o *framework* disponibiliza bibliotecas com funções específicas de entrada e saída, de desenho, de importação de *scripts* e de manipulação dos tipos básicos da linguagem Lua [SOARES, 2008].

5.2 Padrões de projeto Utilizados

Segundo Silva [SILVA, 2006], os padrões de projeto (*design patterns*) [JOHNSON, 1995] utilizados durante a modelagem e implementação dos *frameworks*, são de extrema importância no desenvolvimento de aplicações orientadas a objetos. O MoonDo fez uso de algumas soluções descritas nestes padrões para resolver os problemas de implementação enfrentados. O Quadro 5.1, mapeia os padrões de projetos utilizados nas entidades da arquitetura.

Quadro 5.1 Padrões de projeto adotados na arquitetura do framework MoonDo

Padrão de Projeto	Descrição do Padrão de Projeto	Entidade
Observer	Permite que quando um objeto tem algum atributo modificado, todos seus objetos dependentes são notificados e atualizados automaticamente.	ObserverApplication; TComponent
State	Permite que o comportamento de um objeto seja alterado em tempo de execução dependendo do seu estado (conjunto de atributos).	ObserverApplication; Scene
Chain Responsibility	Representa um encadeamento de objetos receptores para o processamento de diferentes solicitações.	ObserverApplication
Sigleton	Garante a existência de apenas uma instancia da classe.	ObserverApplication; Display
Data Acess Object (DAO)	Separa as regras de negócio da tecnologia de persistência de dados.	DAO
Bridge	Define uma abstração que pode ser utilizada independente das suas implementações.	Cadastro
Template Method	Permite que métodos abstratos sejam implementados por subclasses usadas nos algoritmos da classe pai.	TComponent
Mediator	Possibilita o desacoplamento e gerencia as colaborações entre um grupo de objetos.	TComponent; TFrame;

Dentre os elementos da arquitetura do MoonDo, a entidade ObserverApplication agrega uma importância especial, pois é responsável por centralizar os eventos de ação do controle remoto, para que através do seu estado

atual, possa identificar a forma correta de tratá-lo e assim notificar o Display, caso haja alguma alteração gráfica. Por esta razão, o desempenho desta entidade foi otimizada graças ao uso dos padrões de projeto, exibidos no Quadro 5.1.

5.3 Ferramentas e Tecnologias

As ferramentas e tecnologias utilizadas na construção framework MoonDo são de natureza Freeware ou Open Source. Além de apresentarem a qualidade desejada, possibilitaram ainda a redução dos custos de desenvolvimento. Segue a lista dos recursos de software adotados:

- IDE *(Interface Development Environment)* Eclipse 3.3.1.1 [ECLIPSE, 2009];
- *Plugin* Lua Eclipse, responsável por reconhecer a sintaxe dos scripts Lua [LUAECLIPSE, 2008];
- Plugin NCL Eclipse, responsável por reconhecer a sintaxe NCL [AZEVEDO et al., 2008]; e
- Ginga-NCL Emulator 1.1.1 [GINGA-NCL, 2008];

5.4 Componentes gráficos desenvolvidos

Esta seção tem como objetivo descrever mais detalhadamente cada componente gráfico implementado, sob a orientação dos guias de usabilidade, para esta primeira versão do *framework* MoonDo.

Todos os componentes gráficos do MoonDo estendem a classe abstrata TComponent, que define atributos e métodos básicos para os componentes, além de criar métodos abstratos para serem implementados obrigatoriamente por suas sub-classes. Alguns destes componentes estendem a classe abstrata TMenuItem, pois podem ser usados com itens de menu pela entidade TMenu. A classe abstrata TMenuItem, obriga a implementação de um método abstrato (*action*) para tratar o comando de ação no menu, pois os itens de menu sofrem ação por meio do controle remoto (pressionando a tecla OK). Para a implementação deste método pode-se fazer uso de uma característica especial do MoonDo, que permite que os métodos abstratos

possam ser implementados após a instanciação do objeto. Abaixo segue uma breve descrição de cada componente gráfico implementado.

- **TPanel –** Subclasse do TConponent, funciona como um container para os outros componentes. Cada TPanel pode ser instanciado e adicionado em diferentes TFrame. Orientado pelas coordenadas do TFrame, pode ser posicionado em qualquer em lugar, e também pode-se alterar sua cor de fundo;
- **TLabel –** Subclasse do TComponent, bastante utilizado no contexto Web e Desktop e também indicado para o uso em aplicações para TVD quando segue os *guidelines*. Cada TLabel pode ser instanciado e adicionado em diferentes TFrame. Orientado pelas coordenadas do TFrame, pode ser posicionado em qualquer em lugar, e também pode-se parametrizar seu texto, fonte e cor;
- **TImage –** Subclasse do TComponent, estrutura responsável por importar uma imagem externa para a aplicação. Cada TImage pode ser instanciado e adicionado em diferentes TFrame. Orientado pelas coordenadas do TFrame, pode ser posicionado em qualquer em lugar, e também pode-se parametrizar o tamanho da imagem, porém é necessário informar o caminho do arquivo de imagem;
- **TIcon –** Subclasse do TMenuItem, basicamente formada por uma imagem (TImage) e um titulo (TLabel). Além de puder ser instanciada e adicionada em qualquer TFrame, pode-se adicioná-la a um objeto de menu (TMenu). Nos TFrame podem ser posicionada em qualquer lugar. No caso de ser adicionada no menu é necessário implementar seu método de ação (*action*). O desenvolvedor tem a possibilidade de customizá-lo informando o a imagem, o label e a posição de seu título (título em cima, em baixo, à esquerda ou à direita da imagem);
- **TText –** Subclasse do TComponent, utilizada para inserção de textos na aplicação. Cada TText pode ser instanciado e adicionado em diferentes TFrame. Orientado pelas coordenadas do TFrame, pode ser posicionado em qualquer em lugar, e também pode-se parametrizar seu texto, fonte e cor,

além de puder especificar a largura limite do texto. Também identifica o token de quebra de linha ('\n') forçando a quebra da linha do texto na exibição;

- **TButton –** Subclasse do TComponent, bastante utilizado no contexto Web e Desktop e também indicado para o uso em aplicações para TVD quando segue os *guidelines*. Cada TButton pode ser instanciado e adicionado em diferentes TFrame. Orientado pelas coordenadas do TFrame, pode ser posicionado em qualquer em lugar, e também pode-se parametrizar sua cor e seu ícone (TIcon). Mesmo não sendo um componente de menu (TMenuItem), é necessário que se implemente o método de ação, *action,* pois diferentemente de outras sub-classes do TComponent este componente sofre ação por meio do controle remoto (pressionando a tecla OK);
- **TField –** Subclasse do TMenuItem, representa uma caixa de texto e é bastante utilizado no contexto Web e Desktop. Além de puder ser instanciada e adicionada em qualquer TFrame, pode-se adicioná-la a um objeto de menu (TMenu). Nos TFrame podem ser posicionada em qualquer lugar. É necessário implementar seu método de ação (*action*) independente de onde for utilizá-lo, TFrame ou TMenu, pois este componente sofre ação por meio do controle remoto (pressionando a tecla OK) nos dois casos. O desenvolvedor tem a possibilidade de customizá-lo informando o label e a posição de sua legenda (legenda em cima, em baixo, à esquerda ou à direita do campo), além de especificar o limite de caracteres, a fonte a ser utilizada e se possui mascara;
- **TMenu –** Subclasse do TMenuItem, no contexto TVD é um dos componentes mais importantes para interação do usuário. Além de puder ser instanciada e adicionada em qualquer TFrame, pode-se adicioná-la a um outro objeto de menu. Nos TFrame podem ser posicionada em qualquer lugar. Para sua utilização é necessário adicionar itens de menu. O desenvolvedor pode customizá-lo informando sua forma de orientação (horizontal ou vertical), o tamanho de sua janela (quantidades de itens a serem exibidos por vez) e se o mesmo é cíclico ou não;

- **TChoicer** - Subclasse do TMenuItem, é uma espécie de menu de escolha, ele deixa visível uma única opção (a selecionada) ao usuário, mas permite que este visualize e selecione as outras opções fazendo uso das setas direcionais. Além de puder ser instanciada e adicionada em qualquer TFrame, pode-se adicioná-la a um objeto de menu (TMenu). Nos TFrame podem ser posicionada em qualquer lugar. É necessário implementar seu método de ação (*action*) independente de onde for utilizá-lo, TFrame ou TMenu, pois este componente sofre ação por meio do controle remoto (pressionando a tecla OK) nos dois casos; e

- **TAnimation –** Subclasse do TComponent, representa a presença de animação no frame. Este objeto é responsável por atualizar a animação, faz com que a tela sofra um reload automático para que a atualização seja feita. Na sua instanciação é necessário a implementação do método abstrato *process(evt)*, que é chamando em toda iteração e recebe como parâmetro o evento da animação, podendo ser passados parâmetros através deste evento. A qualquer momento da execução pode-se iniciar ou parar a animação, chamando o método *start()* e *stop()* do objeto, repectivamente.

Na Figura 5.3, é mostrada uma tela com alguns componentes gráficos. Já na Figura 5.4 são mostrados algumas aplicações construídas com os componentes gráficos do MoonDo.

Figura 5.3 Componentes Gráficos presentes no MoonDo

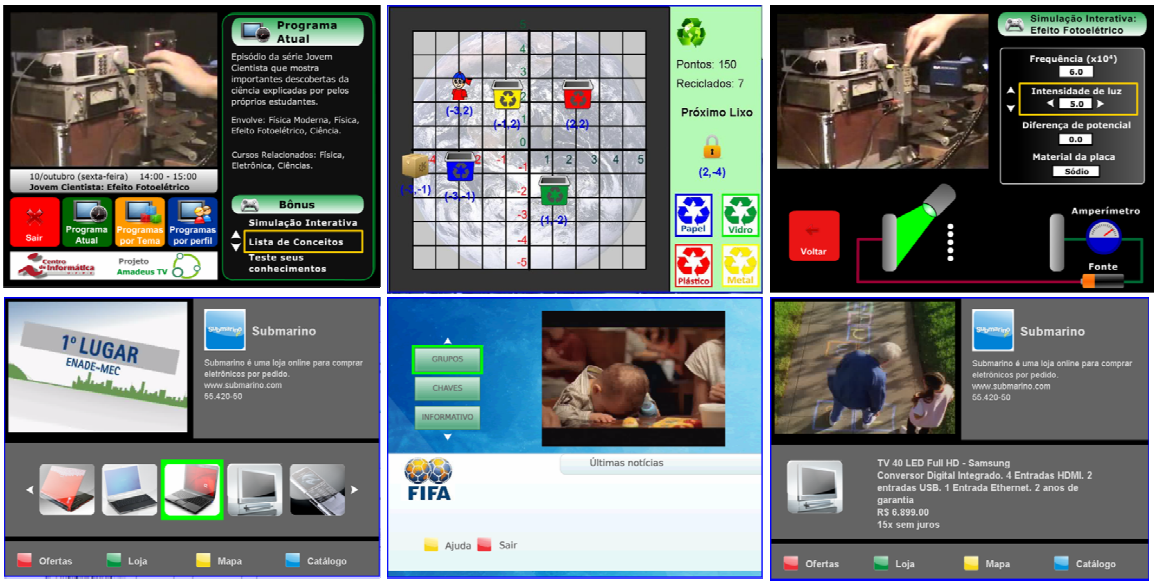


Figura 5.4 Telas de algumas aplicações desenvolvidas com o Moondo

5.5 Estudo de caso: Reciclagem Cartesiana

Diante da relevância do ensino de matemática e da importância da conscientização ecológica, para este estudo de caso foi desenvolvido um Objeto de Aprendizagem (OA), utilizando o *framework* MoonDo, que trata destes temas, de forma interdisciplinar. Ele consiste basicamente de um jogo educacional com o intuito de exercitar os princípios da reciclagem e ao mesmo tempo aplica os conhecimentos matemáticos de plano cartesiano, conforme pode ser observado na Figura 5.5.

O objetivo do jogo consiste em posicionar a lixeira correta no local da queda do respectivo material reciclável, o que exige do aprendiz raciocínio rápido e senso de orientação dentro do plano cartesiano. A jogabilidade foi especialmente adaptada ao contexto da TVD. O personagem se movimenta no plano através das teclas direcionais e a escolha da lixeira correta é feita através das teclas coloridas (verde, azul, amarela e vermelha), que seguem os padrões estabelecidos na resolução N. 275, 25 de abril de 2001, do Conselho Nacional do Meio Ambiente [CONAMA, 2001].

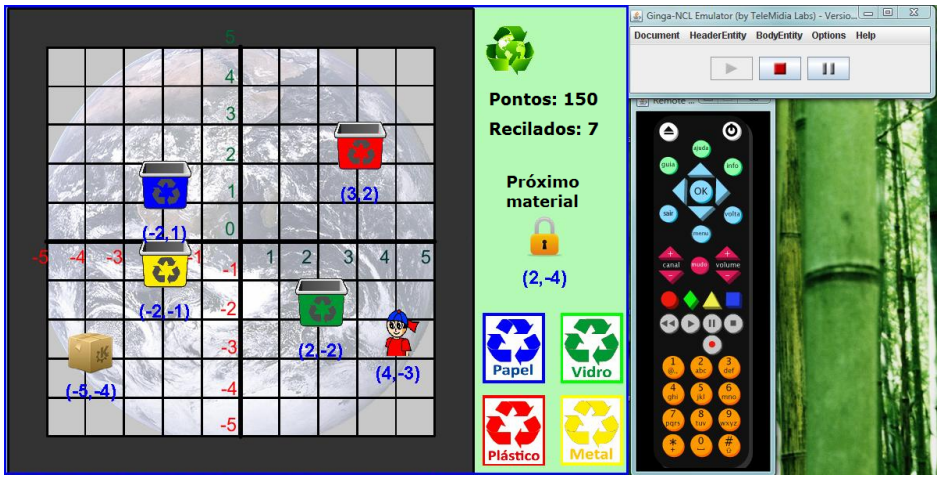


Figura 5.5 Objeto de Aprendizagem que trata do conceito de plano cartesiano sob a temática da reciclagem

O desenvolvimento desta aplicação demonstrou primeiramente a eficiência durante a codificação, resultando em um menor esforço, pois toda a arquitetura da aplicação já estava definida e os componentes gráficos utilizados, implementados. Isto fez com que o desenvolvedor apenas se preocupasse com a lógica do objeto de aprendizagem. Em segundo lugar, os componentes gráficos disponibilizados pelo

framework e o controle de eventos trataram adequadamente a interação do usuário, sem prejudicar a qualidade do design e da usabilidade. Além de facilitar a implementação, a utilização do MoonDo, também reflete na qualidade da manutenabilidade, por deixar a aplicação mais modularizada.

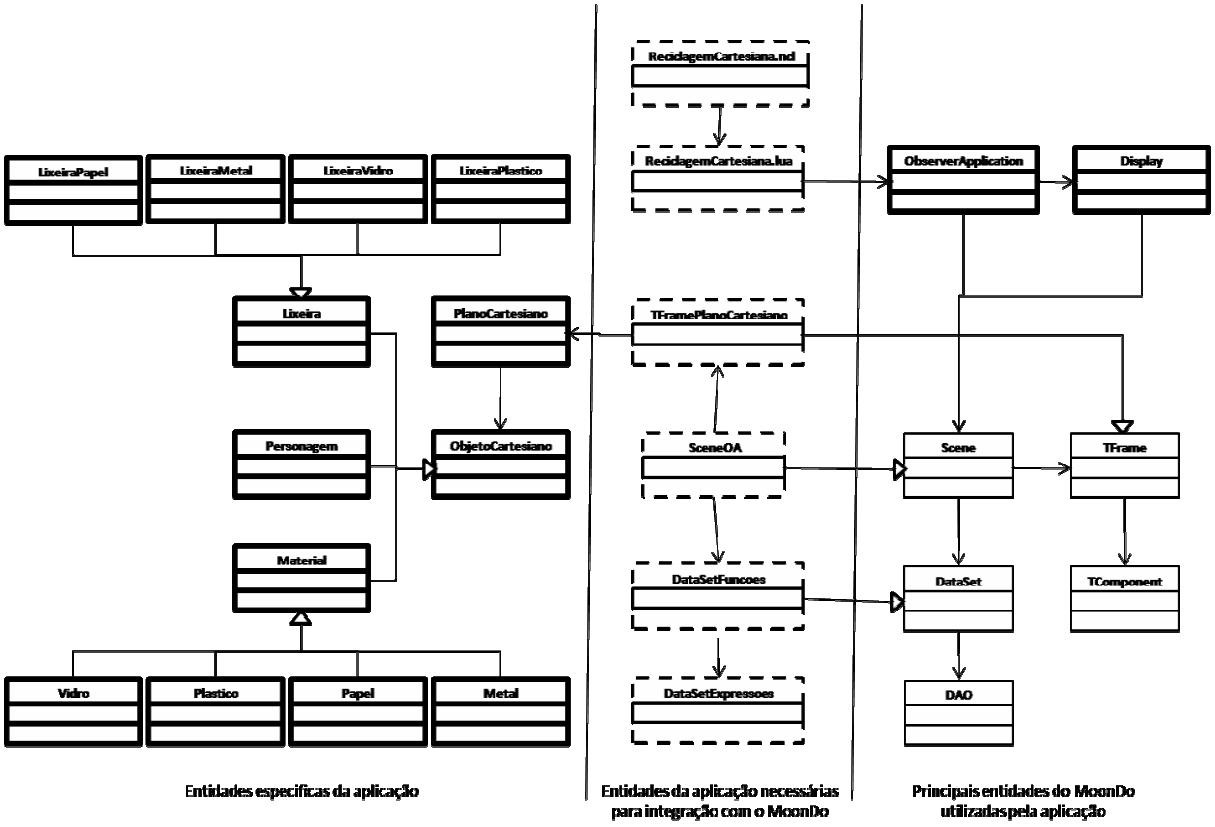


Figura 5.6 Mapeamento das principais entidades da aplicação e os componentes do framework utilizados neste estudo de caso

A aplicação desenvolvida fez uso da arquitetura do *framework* estendendo e combinando seus componentes. A Figura 5.6 mostra a modelagem da arquitetura da aplicação, classificando como: entidades específicas da aplicação, entidades necessárias para integração com o MoonDo e principais entidades do MoonDo utilizadas na aplicação. Segundo a arquitetura do MoonDo (Figura 5.2) é necessário a criação do "Application.ncl" e do "Application.lua" para gerar a aplicação. Este OA representou estas entidades por "ReciclagemCartesiana.ncl" e "ReciclagemCartesiana.lua", respectivamente. Outra obrigatoriedade do MoonDo é que o desenvolvedor estenda pelo menos uma entidade "Scene" e uma entidade "TFrame", que juntas representam as telas da aplicação. Para suprir esta necessidade

foram criadas as entidades “SceneOA” e “TFramePlanoCartesiano”, estendendo as entidades obrigatórias. Como a aplicação fez uso de leitura de arquivos para receber os dados, foi necessário criar classes de manipulação de arquivos (“DataSetFuncoes” e “DataSetExpressões”), estendendo a entidade “DataSet”. Outras entidades básicas que caracterizam o domínio do Objeto de Aprendizagem foram implementadas com o objetivo de modularizar a aplicação. Abaixo, na Figura 5.6, segue a diagramação em UML da arquitetura do objeto de aprendizagem construído e as respectivas entidades do framework utilizadas.

5.6 Considerações Finais

O produto aqui descrito, o Framework MoonDo, torna suas contribuições livremente acessíveis a sociedade, por ser distribuído sob a licença de software livre GPL 2.0. Ele pode ser encontrado, juntamente com sua documentação na página do projeto http://moondo.sourceforge.net/. No Apêndice A pode ser encontrado também um pequeno tutorial que orienta a sua utilização.

6 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. Este conhecimento também precisa ser acessível aos profissionais de outras áreas do conhecimento, aonde estas tecnologias serão aplicadas. Por esta razão, este trabalho concentra suas contribuições na criação de uma nova alternativa no sentido de facilitar e popularizar o desenvolvimento de aplicações interativas para TVD.

Seguindo este objetivo, o produto aqui descrito, o Framework MoonDo, torna suas contribuições livremente acessíveis à sociedade, por ser distribuído sob a licença de software livre GPL 2.0. Esta característica reduz a dependência por soluções proprietárias, reduz o custo de desenvolvimento, e permite o contínuo aprimoramento pela comunidade.

Além das informações técnicas, os resultados obtidos também puderam ser confrontados através de um estudo de caso, que descreve o desenvolvimento de um objeto de aprendizagem sob uma temática pertinente nos dias de hoje: ensino de matemática e conscientização ambiental. Este estudo demonstra principalmente: a flexibilidade do MoonDo na concepção de aplicações de diversos temas; a possibilidade para que a equipe possa concentrar seus esforços no processo criativo; e a facilidade da comunicação entre profissionais da informática e da educação, pois atua sob conceitos de alto-nível, e não em detalhes da linguagem de programação adotada.

Como trabalho futuro, está sendo organizado um experimento com programadores de diferentes perfis, incluindo aqueles com experiência no desenvolvimento de aplicações para TVD, e também profissionais com experiência apenas em plataformas *Desktop* ou *Web*. Através deste experimento, pretende-se identificar dificuldades na utilização do *Framework* MoonDo, calcular a curva de aprendizagem, e identificar oportunidades de melhorias, com o objetivo de agregar

novos recursos que promovam a eficiência do esforço de codificação e aumente a qualidade das aplicações desenvolvidas.

Como outro trabalho futuro, sugere-se estudar a viabilidade do desenvolvimento de uma ferramenta integrada a IDE (*Integrated Development Environment*) utilizada. Isto vai permitir um aumento da visão do programador a respeito das facilidades disponibilizadas, e conseqüentemente, aumento de eficiência.

7 Referências

ABNT (2009). Associação Brasileira de Normas Técnicas. Disponível em: http://www.abnt.org.br – Acessado em: 18 de novembro de 2009.

APPLE Computer (1992). Macintosh Human Inteface Guidelines. New York: Addison Wesley, 1992.

ARVID (2004). A Guide for Digital TV Service Producers. Finland: Arvid, 2004.

ATSC (2008). Advanced Television Systems Committee. Disponível em: http://www.atsc.org – Acessado em: 20 de dezembro de 2008.

AZEVEDO, R. G. de A.; TEIXEIRA, M. M.; SOARES NETO, C. de S., NCL Eclipse: Ambiente Integrado para o Desenvolvimento de Aplicações para TV Digital Interativa em Nested Context Language. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), Recife, 2009

BARBOSA, Simone Diniz Junqueira; SOARES, L. F. G. (2008). TV digital interativa no Brasil se faz com Ginga: Fundamentos, Padrões, Autoria Declarativa e Usabilidade. Congresso da Sociedade Brasileira de Computação. (Org.). Jornada de Atualização em Informática, Belém, 2008.

BARROS, Gil Garcia (2006) A Consistência da Interface com o usuário para a TV Interativa. Dissertação de Mestrado. Universidade de São Paulo, 2006.

BATISTA, C. E. (2006) TV Digital: Java na Sala de Estar. Mundo Java nº 17. pp 30-39. Curitiba, PR, 2006.

BBC (2008). British Broadcasting Corporation. Disponível em: http://www.bbc.co.uk – Acessado em: 20 de dezembro de 2008. Referências Bibliográficas 85

BBCi (2006). Interactive Television Design.Designing for Interactive Television v 1.0. 2006.

BECKER, V. MONTEZ, C. (2004) TV Digital Interativa. Conceitos, desafios e perspectivas para o Brasil. Florianópolis, I2TV, 2004.

BECKER, Valdecir; FORNARI, Augusto; HERWEG FILHO, Günter H.; MONTEZ, Carlos. Recomendações de usabilidade para TV digital interativa. In: II WORKSHOP DE TV DIGITAL. Título. Curitiba: Biblioteca Digital Brasileira da Computação, 2006. 12 p. 55 e 56

BECKER, Valdecir; PICCIONI, Carlos Alexandre; MONTEZ, Carlos; HERWEG FILHO, Gunter H. (2005). Datacasting e Desenvolvimento de Serviços e Aplicações para TV Digital Interativa. In: TEIXEIRA, Cesar Augusto Camilo; BARRÉRE, Eduardo; ABRÃO, Iran Calixto. (Org.). Web e Multimídia: Desafios e Soluções. Poços de Caldas, 2005.

BEUTRAO FILHO, M. F. H. (2008). Gingaway – Uma ferramenta para criação de aplicações GINGA-NCL interativas para TV Digital, trabalho de graduação, Centro de Informática – Universidade Federal de Pernambuco, 2008.

BERNARDO, Nuno. “O guia prático da produção de televisão interactiva”. Centro Atlântico, Porto, 2002.

BRACKMANN, Christian Puhlmann. Sistema Brasileiro de TV Digital. Pelotas. 2008

BRASIL. (2003) Decreto-lei n. 4.901, de 26 de novembro de 2003. Institui o Sistema Brasileiro de Televisão Digital - SBTVD, e dá outras providências, Diário Oficial da República Federativa do Brasil, Pág. 7, Brasília, 27 de novembro 2003.

CGI: Comitê Gestor da Internet no Brasil. Pesquisa sobre o uso das Tecnologias da Informação e da Comunicação (2006). Online: http://www.nic.br/indicadores/usuarios/ – Acessado em: 05/08/2007.

CONAMA, Conselho Nacional de Meio ambiente (2001). Resolução nº 275, de 25 de abril de 2001. Diário Oficial da República Federativa do Brasil, Brasília, Pág 1.

COULOURIS, G.; DOLLIMORE, J.; KINDERG, T. (2007). Distributed Systems: Concepts and Design. Addison-Wesley, 2007.

DALY-JONES, Owen (2003). Case study: The usability of electronic programme guides. In: Gawlinski, M., Interactive Television Production, Focal Press, 2003. 288 p.

DVB (2009). Digital Video Broadcasting. Disponível em: http://www.dvb.org - Acessado em: 20 de novembro de 2009.

ECLIPSE (2009). Eclipse Foundation. Disponível em: http://www.eclipse.org – Acessado em: 5 de novembro de 2009.

FLEX (2009). Disponível em: http://www.adobe.com/br/products/flex/ – Acessado em: Acessado em: 22 de abril de 2009.

GAWLINSKI, Mark (2003). Interactive Television Production. Focal Press, 2003.

GINGA-NCL (2008). Ambiente declarativo do middleware Ginga. Disponível em: http://www.gingancl.org.br - Acessado em: 20 de janeiro de 2008.

GOMES, Alex Sandro; MONTEIRO, Bruno de Sousa; Melo, Cássio; Arcoverde, Daniel; Frota, Carina (2008). Design da Interação de Novos Produtos para TVD: Abordagens Qualitativas. Simpósio Brasileiro de Fatores Humanos em Sistemas Computacionais (IHC). Porto Alegre, 2008.

GUIMARAES, R. L. (2007). Composer: um ambiente de autoria de documentos NCL para TV digital interativa, dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, 2007.

ISDB (2008). Integrated Services Digital Broadcast. DiBEG. Digital Broadcasting Experts Group. Disponível em: http://www.dibeg.org - Acessado em: 20 de dezembro de 2008.

ISO 9241-11. (1998) Ergonomics requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability. International Standard.

JAVA-DTV (2009). Disponível em: https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=javadtv-1.0-oth-JPR@CDS-CDS_Developer . Acessado em: 28 de novembro de 2009

JOHNSON R.E. VLISSIDES J. GAMMA E., HELM R. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

LIBERATE (2002). Interactive Television Design Guide. Liberate Technologies, 2002.

LUA (2009). The Programming Language Lua. Disponível em: http://www.lua.org – Acessado em: 20 de novembro de 2009.

LUAECLIPSE (2008). “Luaeclipse: An integrated development environment for the lua programming language”. http://luaeclipse.luaforge.net/.

LWUIT (2009). Disponível em: https://lwuit.dev.java.net/ – Acessado em: 22 de novembro de 2009.

MENDES, Luciano Leonel (2007). SBTVD – Uma visão sobre a TV digital no Brasil. T&C Amazônia, Ano V, Número 12, Outubro de 2007.

MONTEIRO, Bruno de Sousa (2009). Amadeus-TV: Portal Educacional na TV Digital Integrado a um Sistema de Gestão de Aprendizado. Dissertação de Mestrado, Centro de Informática, Universidade Federal de Pernambuco, 2009.

MONTEIRO, Bruno de Sousa; PROTA, Thiago Monteiro; SOUZA, Fernando da Fonseca de; GOMES, Alex Sandro (2008). Desenvolvimento de Objetos de Aprendizagem para TVDi. Simpósio Brasileiro de Informática na Educação, Fortaleza, 2008.

MONTEZ, Carlos; BECKER, Valdecir (2005). TV Digital Interativa. Conceitos, desafios e perspectives para o Brasil. Editora da UFSC, 2a edição, Florianópolis, 2005.

MORRIS, Steven; SMITH-CHAIGNEAU, Anthony (2005). Interactive TV Standards. Focal Press, 2005.

NCL (2009). Disponível em: http://www.ncl.org.br/ – Acessado em: Acessado em: 30 de novembro de 2009.

NIELSEN, Jakob (2002). Coordinating UI for Consistency. Morgan Kaufmann. 156 p. 2002.

OLIVEIRA, Caio César Neves de (2009). TVision – Ferramenta Gráfica para Desenvolvimento de Aplicações Para TV Digital no Formato GINGA-NCL, trabalho de graduação, Centro de Informática – Universidade Federal de Pernambuco, 2009.

PAES, Alexandre, ANTONIAZZI, Renato , “Padrões de Middleware para TV Digital”, UFF – Universidade Federal de Fluminense, 2005

PICCIONI, C. A.; MONTEZ, C. B. (2004) Um Estudo sobre Emuladores de Aplicações para a Televisão Digital Interativa. I WorkComp Sul, Palhoça, 2004.

PICCOLO, L.; BARANAUSKAS, M. C. C. (2006). Desafios de Design para a TV Digital Interativa. Simpósio Brasileiro de Fatores Humanos em Sistemas Computacionais (IHC). 2006.

SBTVD (2009). Sistema Brasileiro de TV Digital. Disponível em: http://sbtvd.cpqd.com.br - Acessado em: 26 de novembro de 2009.

SILVA, Ricardo P. e, PRICE, R. T. (1997). O uso de técnicas de modelagem no projeto de frameworks orientados a objetos. In: Proceedings of 26th International Conference of the Argentine Computer Science and Operational Research Society (26th JAIIO) / First Argentine Symposium on Object Orientation (ASOO'97). Buenos Aires, Argentine: aug. 1997. p.87-94.

SILVA, Anderson Fér Ribeiro; OLIVEIRA, Marcos Roberto Martins de (2006). Projeto de um framework de desenvolvimento de interfaces para TV digital. Trabalho de Conclusão de Curso - Universidade de Brasília, Instituto de Ciências Exatas, Departamento de Ciência da Computação. Brasília, 2006

SILVA, Lincoln D. N.; BATISTA, Carlos E. C. F.; LEITE, Luiz E. C.; SOUZA FILHO, Guido L. de (2007). Suporte para desenvolvimento de aplicações multiusuário e multidispositivo para TV Digital com Ginga. Revista T&C Amazônia, Ano V, Número 12, Outubro de 2007.

SILVA, R. P (2000). Suporte ao desenvolvimento e uso de frameworks e componentes. Tese de doutorado. Porto Alegre, UFRGS/II/PPGC, 2000.

SOARES, L. F. G.; RODRIGUES, R. F.; MORENO, M. F. (2007). Ginga- NCL: the Declarative Environment of the Brazilian Digital TV System. Journal of the Brazilian Computer Society. n.4, v. 12, Março, 2007.

SOARES, Luiz Fernando Gomes; SOUZA FILHO, Guido Lemos de (2007). Interactive Television in Brazil: System Software and the Digital Divide. European Conference on Interactive TV (EuroITV), Áustria, 2007.

SOUZA JÚNIOR, Paulo José de (2009). LuaComp: Ferramenta de Autoria de Aplicações para TV Digital. Dissertação de Mestrado, Departamento de Engenharia Elétrica, Universidade de Brasília, 2009.

SOMMERVILLE, Lan (2007). Engenharia de Software. Pearson Addison-Wesley, 2007.

SUN (2009). Sun Microsystems Inc. Disponível em: http://www.sun.com – Acessado em: 24 de novembro de 2009.

TIRESIAS (2009). Guidelines: Television. Disponível em: http://www.tiresias.org – Acesso em : 20 de novembro de 2009.

TOME, Takashi; MONTEIRO, Marcelo S. M.; PICCOLO, Lara S. G.; NISHIHARA, Ricardo M.; LAMAS, Amilton da Costa (2007). Abordagem Sistêmica no Sistema Brasileiro de Televisão Digital (SBTVD). Disponível em: http://cpqd.com.br/file.upload/07_artigoforum_sbtvd.pdf - Acessado em: novembro de 2009.

W3C (2009). World Wide Web Consortium. Extensible Markup Language. Disponível em: http://www.w3.org/XML - Acessado em: 20 de novembro de 2009.

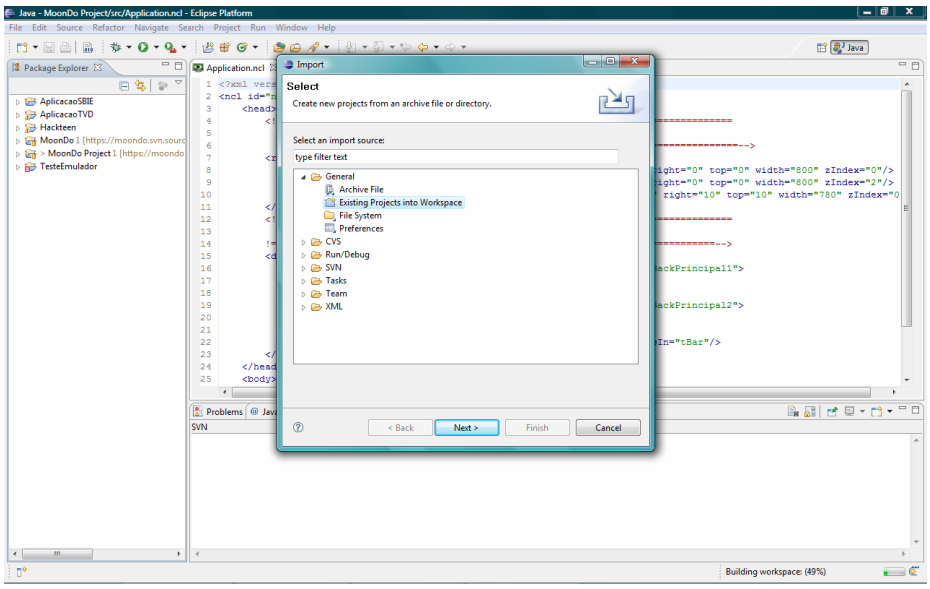
Apêndice A – Tutorial MoonDo

O objetivo desde documento é descrever como utilizar o Framework MoonDo para o desenvolvimento de aplicações declarativas no Windows.

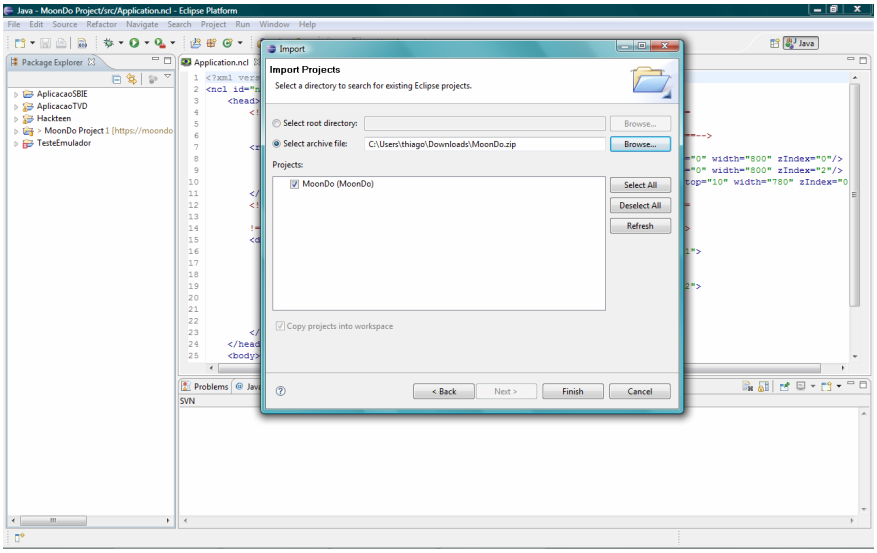
Pré-Requisitos

- Eclipse 3.3.1.1 (http://www.eclipse.org/);
- Java Standard Edition Development Kit 1.6 (http://www.sun.com/);
- Plugin Lunar Eclipse, responsável por reconhecer a sintaxe dos scripts Lua (http://luaeclipse.luaforge.net/);
- Plugin NCL Eclipse, responsável por reconhecer a sintaxe NCL (http://laws.deinf.ufma.br/~ncleclipse/);
- Lua for Windows (http://luaforwindows.luaforge.net/); e
- MoonDo (http://moondo.sourceforge.net/).

1. **Criando um Projeto**
    a. Após todos os download e instalação de todos os pré-requisitos;
    b. Execute o eclipse já configurado com os plugins (Lunar Eclipse e NCL Eclipse);
    c. Importe o projeto do MoonDo
    d. Clique em "File"->"Import...";
    e. Selecione "Existing Project into Workspace":



f. Localize o arquivo do MoonDo:



g. Alterar o nome do projeto para o nome de sua aplicação (Ex.: HelloWorld)

2. Desenvolvendo a aplicação

a. Após a criação do projeto, criar os componentes obrigatórios do framework, o MoonDo obriga o desenvolvedor a implementar os seguintes arquivos:

- [nome_aplicação].NCL (Ex.:HelloWorld.ncl), localizado na raiz da pasta "src" no projeto. Responsável por definir a região da aplicação no documento NCL;

- [nome_aplicação].Lua (Ex.:HelloWorld.lua), localizado na raiz da pasta "src" no projeto. Aplicação propriamente dita, mídia executada pelo documento NCL;
- As TScene's da Aplicação (Ex.:SceneHelloWorld.lua) , localizado e organizado na pasta "src/scene". A aplicação poderá ser dividida logicamente em Scene's que são registradas e inicializadas pela aplicação. Cada aplicação necessita ter no mínimo uma Scene;
- Os TFrame's das Scenes (Ex.:TFrameHelloWorld.lua), localizado e organizado na pasta "src/scene". Os TFrame's são as telas da aplicação e cada Scene poderá ter seus próprios TFrame's, onde serão organizados os componentes gráficos e tratados os eventos do controle. Cada Scene deverá ter pelo menos um TFrame;

b. Criar o "HelloWorld.ncl", contendo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="nclHello" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
	<head>
		<regionBase>
			<region bottom="0" height="600" id="rgBackPrincipal"
left="0" right="0" top="0" width="800" zIndex="0"/>
		</regionBase>
		<descriptorBase>
			<descriptor focusIndex="ixLua" id="dsBackPrincipal"
region="rgBackPrincipal"/>
		</descriptorBase>
	</head>
	<body>
		<media descriptor="dsBackPrincipal" id="app"
src="HelloWorld.lua" type="application/x-ginga-NCLua"/>
		<port component="app" id="pInicio"/>
	</body>
</ncl>
```

c. Criar o “HelloWorld.lua”, contendo:

```
require("framework.framework_nclua");
BibliotecaAuxiliarScript.execute("scene.SceneHelloWorld");

ObserverApplication.register(SceneHelloWorld,SceneHelloWorld.id);
SceneHelloWorld:inicialize();
```

d. Criar o ”SceneHelloWorld.lua”, contendo:

```
BibliotecaAuxiliarScript.execute('framework.src.util.app.Scene');
BibliotecaAuxiliarScript.execute('scene.TFrameHelloWorld');

SceneHelloWorld = Scene.new();
SceneHelloWorld.id = 'SceneHelloWorld';
```

e. Criar o “TFrameHelloWorld.lua”, contendo:

```
BibliotecaAuxiliarScript.execute('framework.src.gui.TFrame');
BibliotecaAuxiliarScript.execute('framework.src.gui.TLabel');

TFrameHelloWorld = TFrame.new();
TFrameHelloWorld.id = "TFrameHelloWorld";

function TFrameHelloWorld:inicialize()

        TFrameHelloWorld:setAltura(600);
        TFrameHelloWorld:setLargura(800);
        TFrameHelloWorld:setCorFundo(Cor.new({r=0,g=0,b=255}));

        -- Fonte
        local fonte = Fonte.new({nome='Arial', tamanho=30});
        fonte.cor = Cor.new({r=255,g=255,b=255});
        fonte.is_negrito = true;

        --TLabel
        TFrameHelloWorld.label = TLabel.new();
        TFrameHelloWorld.label:setTexto("Hello World");
        TFrameHelloWorld.label:setPx(10);
        TFrameHelloWorld.label:setPy(100);
        TFrameHelloWorld.label:setFonte(fonte);

        TFrameHelloWorld:addComponent(TFrameHelloWorld.label,1);
end
```

3. Executando a aplicação
    a. Após escrever o código da aplicação pode-se simular sua execução
    b. Abrir o arquivo HelloWorld.ncl
    c. Selecionar a opção “Run”