NCLFORMS: UMA API PARA DESENVOLVIMENTO DE GUIS EM APLICAÇÕES INTERATIVAS DE TV DIGITAL

Renan Ribeiro de Vasconcelos

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores: Cláudia Maria Lima Werner
Wagner Schau de Castro

Rio de Janeiro
Fevereiro de 2012

NCLFORMS: UMA API PARA DESENVOLVIMENTO DE GUIS EM APLICAÇÕES INTERATIVAS DE TV DIGITAL

Renan Ribeiro de Vasconcelos

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinada por:

_____

Prof. Cláudia Maria Lima Werner, D.Sc.

_____

Prof. Wagner Schau de Castro, M.Sc.

_____

Prof. Débora Christina Muchaluat Saade, D.Sc.

_____

Prof. Marcelo Schots de Oliveira, M.Sc.

RIO DE JANEIRO, RJ – BRASIL

FEVEREIRO de 2012

Vasconcelos, Renan Ribeiro de

NCLForms: Uma API para Desenvolvimento de GUIs em Aplicações Interativas de TV Digital/ Renan Ribeiro de Vasconcelos. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2012.

XIII, 113 p.: il.; 29,7 cm.

Orientadores: Cláudia Maria Lima Werner e Wagner Schau de Castro

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2012.

Referências Bibliográficas: p 106-111.

1. Aplicações de TV Digital 2. Interface Gráfica 3. TV Digital 4. Reutilização de Software 5. Padrões de Projeto. I. Werner, Cláudia Maria Lima *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Computação e Informação. III. Título.

Aos meus pais, Admilton e Maria da Graça,

ao meu irmão, Rodrigo,

à minha avó, Ilda,

e aos meus amigos.

Agradecimentos

A Deus, por tudo em minha vida.

Aos meus pais, que sempre estiveram ao meu lado e sempre buscaram o melhor para mim.

Ao meu irmão, pela disponibilidade constante de apoio.

À minha avó Ilda, pelo carinho durante toda a minha criação.

Aos meus amigos, todos eles, que me honraram com suas presenças em diversos momentos da minha vida.

Aos meus animais, gatos e cachorros, que sempre proporcionaram momentos de grande alegria.

Aos meus demais familiares, presentes desde o meu nascimento.

À professora Claudia Werner, minha orientadora, pela sabedoria, apoio, paciência e compreensão ao longo desse estudo. Sem tal ajuda, isso não seria possível.

Ao Wagner, meu coorientador, que participou desde a origem desse trabalho e colaborou em diversas situações.

À professora Débora Saade e ao Marcelo Schots, membros da banca, por aceitarem o convite e participarem da apresentação.

Aos colegas de Lab3D e LENS, em especial ao Sérgio Meyer, pelas conversas e dicas constantes.

Ao professor Guilherme Travassos, pelo apoio durante boa parte do curso.

Ao professor José Rezende, pela pronta ajuda em diversas questões durante a faculdade.

A todos aqueles que fizeram parte da minha vida em algum momento.

Resumo do Projeto de Graduação apresentação à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

NCLForms: Uma API para Desenvolvimento de GUIs em Aplicações Interativas de TV Digital

Renan Ribeiro de Vasconcelos

Fevereiro/2012

Orientadores: Cláudia Maria Lima Werner

Wagner Schau de Castro

Curso: Engenharia de Computação e Informação

Com o advento da TV Digital, novos serviços passaram a ser oferecidos. A interatividade cria novas oportunidades tanto para o telespectador, que passa a ter um papel mais ativo na relação com a TV, como para desenvolvedores, que podem planejar diferentes tipos de aplicações, adaptadas para esse formato de dispositivo. Nesse contexto, é importante se pensar em novos modelos de desenvolvimento, de acordo com as características do ambiente de TV Digital. O *middleware* de interatividade brasileiro Ginga e seu subsistema, Ginga-NCL, são estudados de forma a reconhecer novas alternativas de aplicação. Com esse objetivo, buscam-se na literatura conceitos que apoiem o estabelecimento de uma proposta que suporte o desenvolvimento voltado para esse padrão. Baseando o desenvolvimento em alternativas que viabilizem a reutilização de *software*, bibliotecas de código são reconhecidas como produtos de fácil adoção durante o projeto de aplicações com documentos NCL. De forma a adaptar tal proposta às características de aplicações interativas para TV Digital, este estudo busca inspiração nos padrões de projeto de interface voltados para essa área. Com isso, identifica-se uma dificuldade em se adicionar elementos de interface gráfica em aplicações implementadas com as linguagens NCL e NCLua. A proposta formulada neste trabalho visa desenvolver uma API para desenvolvimento de GUIs em aplicações interativas de TV Digital, referentes ao ambiente Ginga-NCL. Além disso, busca-se associar a implementação de soluções de alguns padrões de projeto com a API proposta, de forma a justificar sua adoção em projetos reais e apoiar o desenvolvimento no cenário de TV Digital.

Palavras-chave: Aplicações de TV Digital, Interface Gráfica, Reutilização de *Software*, Padrões de Projeto.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

NCLForms: An API for the GUI Development in Interactive Digital Television Applications

Renan Ribeiro de Vasconcelos

February/2012

Advisors: Cláudia Maria Lima Werner

Wagner Schau de Castro

Major: Computer and Information Engineering

With the advent of Digital TV, new services began to be offered. The interactivity creates new opportunities for viewers, which assume an active role in their relation with TV, and for developers, which can plan different applications types, adapted to such device format. Within this context, it is important to think about new models of development, according to the characteristics of Digital TV environment. The interactive Brazilian middleware Ginga and its subsystem, Ginga-NCL, are studied in order to recognize new alternatives of application. With this objective, concepts are sought in the literature to support the establishment of a proposal to support the development toward this model. With the development based on alternatives that allow the software reuse, code libraries are recognized as products of easy adoption during the project of applications with NCL documents. In order to adapt such proposal to the characteristics of interactive Digital Television applications, this study looks for inspiration on interface design patterns toward this area. Thus, a difficulty in adding GUI elements in applications implemented with the languages NCL and NCLua is identified. The proposal aims at developing an API to build GUIs in interactive Digital TV applications, related to the Ginga-NCL environment. In addition, this study tries to associate the implementation of some design patterns solutions with the proposed API, in order to justify its adoption in real projects and to support the development in the Digital TV scenario.

Keywords: Digital TV Applications, Graphical Interface, Software Reuse, Design Patterns.

Sumário

Lista de Figuras xi

Lista de Tabelas xiii

1. Introdução 1

2. Aplicações Interativas de TV Digital 4

2.1. Televisão 4

2.2. Aplicações Interativas 5

2.2.1. Serviços permanentes 6

2.2.2. Aplicações de TV aprimorada 7

2.3. Usabilidade 7

2.4. Dispositivos 8

2.5. Modelo brasileiro 10

2.5.1. Ginga 10

2.5.2. Ginga-NCL 11

2.5.3. Ginga-J 18

3. Reutilização e Desenvolvimento de *Graphical User Interfaces* 19

3.1. APIs 19

3.2. *Frameworks* 21

3.3. Elementos de Interface Gráfica 23

3.3.1. Classificação de Elementos de Interface 24

3.3.2. GUIs 27

4. *Design Pattern* para TV Digital Interativa 32

4.1. *Design Patterns* 32

4.2. Desenvolvendo *Design Patterns* 35

4.2.1. Métodos de desenvolvimento 37

4.3. Coleção de *Design Patterns* 45

4.3.1. Linguagem de *Patterns* 45

4.3.2. Padrões de Projeto de Interface 47

5. NCLForms - Apresentação e Contexto de Uso 61

5.1. A API 61

5.1.1. Reutilização 63

5.1.2. *Widgets* 63

5.2. Utilização 82

5.3. Adoção em *Design Patterns* 96

5.3.1. Estendendo a linguagem de *patterns* 97

6. Conclusão 103

6.1. Contribuições 103

6.2. Limitações 104

6.3. Trabalhos Futuros 105

Referências Bibliográficas 107

Lista de Figuras

Figura 1. Arquitetura do *middleware* Ginga. 11

Figura 2. Estrutura da linguagem NCL (adaptada de (RATAMERO, 2007)). 15

Figura 3. Exemplo de documento NCL. 15

Figura 4. Aplicação com formulário implementado através da API NCLForms. 62

Figura 5. Modelo conceitual com todos os elementos propostos na API. 64

Figura 6. Representação gráfica dos estados do *widget statebutton*. 66

Figura 7. Representação gráfica do elemento *textbox*. 67

Figura 8. Representação gráfica do elemento *passwordbox*. 70

Figura 9. Representação gráfica do *widget checkbox*. 73

Figura 10. Representação gráfica do *widget radio buttons*. 75

Figura 11. Representação gráfica do *widget list box*. 78

Figura 12. Representações gráficas do elemento legenda. 80

Figura 13. Incorporando script Lua em um documento NCL. 83

Figura 14. Passagem de parâmetros para o *script* NCLua. 83

Figura 15. Recepção de parâmetros pela GUI com a função *nclHandler*. 84

Figura 16. Criando as estruturas dos elementos na função *formParser*. 86

Figura 17. Estrutura da metatabela do elemento botão de estados. 86

Figura 18. Função *redraw* para exibir cada *widget* na tela. 89

Figura 19. Exemplo de parâmetros para o botão de estados. 92

Figura 20. Exemplo de parâmetros para a caixa de texto. 92

Figura 21. Exemplo de parâmetros para a caixa de senha. 92

Figura 22. Exemplo de parâmetros para a caixa de seleção. 92

Figura 23. Exemplo de parâmetros para o botão de opção. 92

Figura 24. Exemplo de parâmetros para a caixa de listagem. 92

Figura 25. Exemplo de parâmetros para a legenda..... 93

Figura 26. Dados de saída retornados..... 93

Figura 27. Documento NCL exemplificando o uso da GUI NCLForms. 93

Figura 28. Aplicação usando a API NCLForms com apresentação de parâmetros..... 95

Lista de Tabelas

Tabela 1. Classificação de *widgets*. 26

Tabela 2. Problemas de *design* referentes a “Leiaute de página”..... 38

Tabela 3. Problemas de *design* referentes a “Navegação”. 39

Tabela 4. Problemas de *design* referentes a “Teclas de controle remoto”. 39

Tabela 5. Problemas de *design* referentes a “Funções básicas”. 40

Tabela 6. Problemas de *design* referentes a “Apresentação de conteúdo”. 41

Tabela 7. Problemas de *design* referentes a “Participação do usuário”..... 42

Tabela 8. Problemas de *design* referentes a “Entrada de texto”. 42

Tabela 9. Problemas de *design* referentes a “Ajuda”. 43

Tabela 10. Problemas de *design* referentes a “Acessibilidade e personalização”. 43

Tabela 11. Correspondências entre padrões e elementos gráficos da API. 97

1. Introdução

Os aparelhos de televisão deixaram de ser apenas telas exibindo os programas transmitidos pelas emissoras. Com o avanço tecnológico, as TVs ganharam novas funções, reproduzindo e permitindo novas formas de conteúdo. Assim como ocorreu com os telefones móveis, que passaram a ser chamados de *smartphones*, vem ganhando volume o conjunto de televisões que já recebem o nome de *smart* TVs.

Com a adoção do sinal digital, novas funcionalidades passaram a ser possíveis em aparelhos de TV. Um componente já existente em outras mídias, a interatividade, abriu caminho no campo da televisão. O telespectador pode ter um papel mais ativo em sua relação com a TV. Pode-se pensar até em outra denominação, onde o espectador torna-se um usuário de TV. No entanto, apenas uma pequena parte do potencial de uso da interatividade em TV Digital foi explorada (JÄÄSKELÄINEN, 2001), devendo-se sempre buscar novas formas de ampliar a oferta e a qualidade de serviços.

Com a chamada TV Digital interativa, aplicações podem ser executadas diretamente na tela da televisão. Do ponto de vista do desenvolvimento, novos paradigmas devem ser seguidos ao se pensar em uma aplicação para TV, além de aspectos como domínios de aplicações, tipos de usuários, interface de interação, dentre outros.

Ao se pensar em novos serviços para TV Digital, é importante conhecer o padrão adotado por cada país, a fim de reconhecer os requisitos estipulados por cada um deles visando o desenvolvimento de novos projetos. O padrão de transmissão digital adotado no Brasil, o Sistema Brasileiro de Televisão Digital (SBTVD), é baseado no modelo japonês, o *Integrated Services Digital Broadcasting – Terrestrial* (ISDB-T).

No que diz respeito às aplicações, é ainda mais importante conhecer as características do *middleware* utilizado. O modelo brasileiro é chamado de Ginga, contando com dois módulos, o Ginga-J e o Ginga-NCL (ABNT, 2008). Por ser um modelo relativamente recente – as primeiras transmissões digitais no Brasil começaram em 2007 –, ainda há bastante espaço para o desenvolvimento de novas ferramentas e tecnologias que colaborem na expansão da interatividade em TVs no país.

Uma forma de apoiar o desenvolvimento de aplicações é por meio de bibliotecas de código. Tal alternativa permite colocar em prática técnicas de reúso a fim de acelerar a construção de tais aplicações. Outro caminho muito comum em desenvolvimento de *software* diz respeito aos *design patterns*, que fornecem orientações na forma de melhores práticas para a implementação de projetos em domínios específicos.

Assim como em qualquer área de *software*, aplicações interativas de TV Digital também contam com bibliotecas e *patterns*, porém ainda necessitam de novas propostas e principalmente da aplicação de tais propostas no cenário brasileiro, por não ter ainda atingido um estado de maturidade.

O presente estudo, inserido no contexto do projeto "AutorTVD-Reuse", estabelecido entre o Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ e os Departamentos de Ciência de Computação e de Engenharia de Telecomunicações da UFF, procura colaborar nessa discussão, reconhecendo a importância de bibliotecas e *design patterns* durante o processo de desenvolvimento de aplicações interativas para TV Digital. Além disso, ao se identificar uma dificuldade em se inserir elementos de interface gráfica em aplicações no ambiente Ginga-NCL, é apresentada uma biblioteca de código que propõe uma *Application Programming Interface* (API) para desenvolvimento de interfaces de usuário com elementos gráficos

em aplicações com componentes de formulários. Tal proposta é colocada sob a ótica da reutilização de *software* no contexto do ambiente Ginga-NCL, através dos recursos fornecidos pela integração entre as linguagens NCL e Lua. Com isso, o presente estudo busca colaborar no desenvolvimento de interfaces gráficas em aplicações de TV Digital no cenário brasileiro.

Como forma de promover a discussão, são propostas também formas de implementação de soluções a uma linguagem de *design patterns* de interação para aplicações em TV Digital. Tais propostas buscam também identificar possíveis aplicações da API desenvolvida, de forma a atender a alguns dos padrões estudados.

Dessa forma, o texto é organizado em seis capítulos, contando com esta introdução. O Capítulo 2 apresenta uma descrição dos tipos de aplicações interativas para TV Digital, além de destacar as principais características do modelo brasileiro para interatividade. O Capítulo 3 faz uma revisão sobre APIs e *frameworks* em termos de reutilização de *software*, abordando também características de elementos de interface gráfica, além de ferramentas voltadas para a construção de *Graphical User Interfaces* (GUIs). O Capítulo 4 realiza uma análise de *design patterns* voltados para interação em aplicações de TV. No Capítulo 5, é apresentada a API desenvolvida e é proposta a sua aplicação na implementação de soluções em uma linguagem de *patterns* estudada. O Capítulo 6 conclui o trabalho, apresentando as considerações finais.

2. Aplicações Interativas de TV Digital

Neste capítulo, é realizada uma descrição dos modelos de aplicação para Televisão Digital interativa, a fim de caracterizar o cenário atual e possibilitar uma análise mais aprofundada ao longo deste estudo. Após uma discussão de aplicações em geral, o texto passa, a partir da Seção 2.5, a focar o modelo brasileiro, com uma breve descrição do *middleware* Ginga e das ferramentas de desenvolvimento, com ênfase para as linguagens NCL e Java.

2.1. Televisão

Antes de se avançar a discussão em aplicações para TV Digital, é necessária uma breve descrição do dispositivo que é personagem principal nesse estudo: a televisão. Sem especificar tipos ou modelos, como digital ou analógica, a televisão em si não pode ser definida adotando o seu uso como referência. Os modos de acesso à televisão vêm se alterando intensamente nos últimos anos. Tal dispositivo deixou de ser apenas um aparelho disposto em algum cômodo de uma residência para ser acessível em diferentes formatos. A TV passou a ser portátil, integrada a outros dispositivos, conectada.

Diferentes formas de uso da televisão suscitam a dúvida sobre uma definição que abranja todos os seus modos de acesso. Hasebrink diz que a TV tradicional é uma forma de contato pessoal com comunicações audiovisuais que seguem alguns atributos, como padronização, onde todos os telespectadores recebem o mesmo conteúdo, dependência temporal, sendo simultânea para todos, apresentando apenas um sentido no envio de informação, e pública (HASEBRINK, 2001 *apud* KUNERT, 2009).

Uma definição usando esses termos é adequada para qualquer tipo de TV, desde que esta não possua a chamada interatividade. Aplicações interativas alteraram alguns dos atributos característicos da televisão tradicional, como a não exigência de padronização e o sentido da comunicação.

2.2. Aplicações Interativas

Apesar de a televisão ser anterior ao telefone celular, a área de aplicações interativas em aparelhos de telefonia móvel se desenvolveu de modo bem mais intenso, talvez pelo forte apelo da portabilidade trazida com os *smartphones*. Em termos de TV Digital, aplicações ainda podem ser consideradas como pertencentes a uma área nova, cujo desenvolvimento e popularização da tecnologia ainda são processos em ascensão.

A fim de guiar o progresso da pesquisa, alguns pontos devem ser estabelecidos, de forma que o conceito de aplicações interativas em TV Digital não seja interpretado erroneamente. Nesse contexto, algumas características podem ser ressaltadas, como independência de conteúdo (o que diferencia aplicações interativas para TV de TV interativa ou programas de TV interativos) e de tipo de aplicação, e estar relacionado à televisão digital (KUNERT, 2009). Uma simples definição que abrange esses tópicos é a que aponta aplicações interativas de TV como serviços avançados ou interativos com TV Digital (EUROPEAN BROADCASTING UNION, 2004a *apud* KUNERT, 2009).

Quando se fala em aplicações interativas, deve-se pensar que existem diversas formas possíveis de interatividade em televisão. Nesse sentido, segundo PAGANI (2003 *apud* KUNERT, 2009), três níveis de interatividade podem ser identificados, sendo classificados como local, simples e completo.

A interação local ocorre em aplicações que não exigem canal de retorno. Muitas funções desse tipo podem ser encontradas em diversos modelos de TV, como clássicas opções de menu. As aplicações de interatividade simples, por sua vez, necessitam de um canal de retorno e os exemplos mais comuns estão em opções de *pay-per-view* e jogos. Por último, a interação completa se dá em aplicações de conferência, onde há ligações com vídeo por meio da própria televisão.

Uma outra forma de classificação de aplicações interativas para TV é baseada na relação entre a aplicação em si e a programação do canal de televisão. Aqui, as aplicações podem estar disponíveis a qualquer momento como serviços diferenciados ou disponibilizadas em conjunto com certos programas de TV, aprimorando a experiência do telespectador daquele programa com conteúdo adicional referente ao tratado naquele momento (KUNERT, 2009).

2.2.1. Serviços permanentes

Algumas aplicações interativas são consideradas serviços, por serem disponibilizadas de forma permanente, comumente chamadas de 24/7, ou seja, vinte e quatro horas por dia e por sete dias na semana. Tais aplicações são dedicadas e autossuficientes. Normalmente, seu conteúdo não está relacionado a nenhum programa de TV e são serviços comuns, como terminal eletrônico de compras, e aplicações de entretenimento, tais como jogos.

GAWLINSKI (2003 *apud* KUNERT, 2009) propõe uma divisão de possíveis aplicações para serviços permanentes, como: guia de programação eletrônico, tele texto,

walled garden[1] (jardim murado), acesso à Internet na TV, vídeo sob demanda e gravadores de vídeo pessoais.

2.2.2. Aplicações de TV aprimorada

A TV aprimorada é reconhecida pelo uso de serviços interativos sobre uma programação linear, permitindo ao telespectador intervir na mesma (EUROPEAN BROADCASTING UNION, 2004a *apud* KUNERT, 2009). Conforme observa GAWLINSKI (2003 *apud* KUNERT, 2009), os provedores de conteúdo costumam adicionar elementos gráficos e textos, além de diferentes camadas, sobre a programação, permitindo ao usuário assistir à televisão enquanto interage com o serviço multimídia.

2.3. Usabilidade

Identificar o contexto de uso de aplicações interativas em TV Digital é um aspecto de grande importância para o desenvolvimento de aplicações que atendam as necessidades e expectativas do usuário. Entender como os indivíduos fazem uso de aplicações interativas possibilita reconhecer possíveis deficiências de uma aplicação e propor, com isso, interfaces que tornem a experiência do usuário mais agradável.

Antes de considerar a usabilidade no contexto de aplicações interativas, deve-se buscar uma melhor definição para ela. A usabilidade está voltada para a tarefa de otimizar as interações que as pessoas têm com produtos interativos, permitindo a elas realizar suas atividades nos mais diferentes cenários. SHARP *et al.* (2007) identificam uma divisão de usabilidade em seis objetivos: eficaz para uso (eficácia), eficiente para

[1] *Walled garden* refere-se a um conjunto limitado de serviços oferecidos a usuários de forma que a estes não é permitido abandonar tal área de serviços.

uso (eficiência), seguro para uso (segurança), ter boa utilidade (utilidade), fácil de aprender (capacidade de aprendizado) e fácil de lembrar como usar (capacidade de recordação).

Na área de interface homem-máquina, a usabilidade é o critério de qualidade mais adotado. A literatura conta com métricas e métodos bem desenvolvidos para avaliar a usabilidade de um produto. A usabilidade também é adotada nesse texto como parâmetro de qualidade nas discussões posteriores.

A seguir, na Seção 2.4, são caracterizados alguns elementos comuns ao cenário de utilização de aplicações interativas de TV Digital.

2.4. Dispositivos

Um dos equipamentos fundamentais em ambientes de TV Digital é o *set-top box*. Ele é responsável por decodificar o sinal de TV Digital, incluindo as aplicações interativas. O *set-top box* pode ser um dispositivo separado da TV ou já vir incluído no conjunto, sendo interno ao aparelho. O *software* incluído no *set-top box* possui alguns componentes, sendo o *middleware* o componente central, funcionando como uma camada de interface entre o *software* do dispositivo e as aplicações. O *middleware* determina as linguagens de programação que podem ser utilizadas no desenvolvimento de aplicações interativas. Existem diversos padrões de *middleware*, tanto proprietários como de código aberto. Um desses padrões, o adotado no Sistema Brasileiro de Televisão Digital, será discutido na Seção 2.5. Para um equipamento como um *set-top box* ou uma TV possuir interatividade simples ou completa, precisa-se de um canal de retorno. Tal canal pode ser implementado de modos diferentes. Em sistemas de TV

terrestres ou por satélite, o canal pode fazer uso de uma conexão à linha telefônica. Já em sistemas de TV a cabo digital, o canal de retorno é feito pelo próprio cabo.

A tela é outro dispositivo que deve ser considerado no desenvolvimento de aplicações. Suas características, como resolução, proporção, áreas seguras, cores disponíveis, transparência etc., devem ser analisadas a fim de se obter o resultado desejado. Busca-se sempre a independência de dispositivos para uma aplicação, tendo como resultado a possibilidade de se exibir o mesmo conteúdo de TV em outras telas que não sejam de televisão, como monitores de computador e *smartphones*.

O outro equipamento que ainda não foi citado aqui é o controle remoto, meio pelo qual se dá a interação do usuário com a aplicação se dá. Apesar de o *design* desses aparelhos variar bastante, dependendo de sua marca, existe um certo padrão de teclas que devem ser incluídas, como as quatro teclas coloridas (vermelha, verde, amarela e azul), quatro teclas de navegação com setas (cima, baixo, esquerda, direita), uma tecla de confirmação (OK) e o teclado numérico (0-9). KUNERT (2009) critica o modelo atual de controles remotos, pois as possibilidades de interação ficam bastante limitadas pela falta de manipulação direta, como o estilo de interação existente nos *mouses* de computadores, e pela inexistência de entradas de texto, com um teclado alfanumérico.

Com essa breve descrição dos equipamentos comuns ao ambiente de TV Digital, além das diferentes categorias de aplicações interativas, esse estudo busca uma melhor caracterização do modelo brasileiro, com elementos como o *middleware* Ginga e as ferramentas utilizadas.

2.5. Modelo brasileiro

Conforme dito anteriormente, o modelo brasileiro de TV Digital é baseado no *middleware* Ginga. Tal sistema tem suporte a aplicações declarativas e a aplicações procedurais. O suporte a esses recursos é dividido em dois ambientes principais: um ambiente de apresentação, conhecido como Ginga-NCL, e um ambiente de execução, denominado Ginga-J (ABNT, 2008). No entanto, aplicações híbridas são igualmente possíveis, fazendo uso de recursos dos dois ambientes, além da possibilidade de usar uma linguagem procedural, como Lua, em conjunto com aplicações declarativas na máquina de apresentação Ginga-NCL. Essa seção irá caracterizar a arquitetura do *middleware*, focando, na sequência, os detalhes de cada um dos seus ambientes.

2.5.1. Ginga

Além dos dois módulos que compõem os ambientes de apresentação e de execução, a arquitetura do *middleware* Ginga também possui um módulo principal, conhecido como *Common Core*. Os dois módulos previamente citados usam recursos do módulo principal, como decodificadores e procedimentos, tanto para aplicações procedurais como declarativas.

Os subsistemas Ginga-J e Ginga-NCL compõem os chamados Serviços Específicos Ginga, pois as aplicações em execução, de acordo com a natureza da própria aplicação, utilizam os recursos de um dos subsistemas, podendo ser declarativa, fazendo uso de uma linguagem que enfatiza a descrição declarativa do problema ou procedural, que visa uma decomposição em uma implementação algorítmica (ABNT, 2008).

A Figura 1 ilustra a arquitetura em alto nível do *middleware* Ginga.

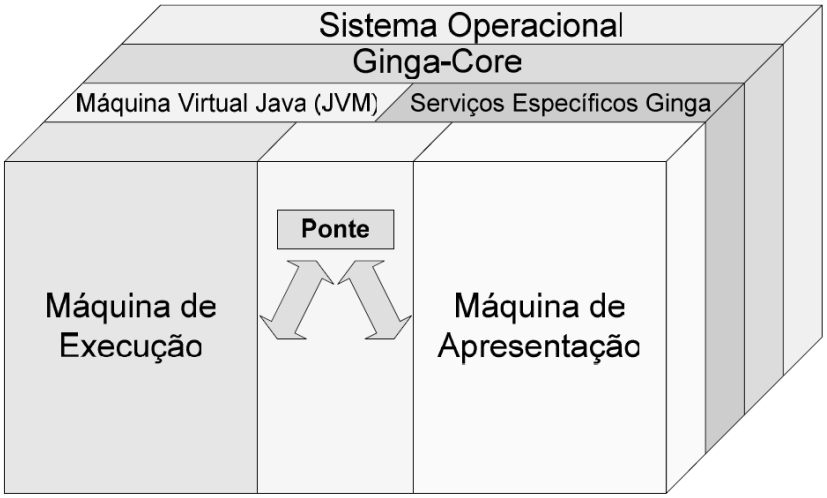

Figura 1. Arquitetura do *middleware* Ginga.

2.5.2. Ginga-NCL

O módulo Ginga-NCL faz uso da linguagem declarativa NCL (*Nested Context Language*). As características da linguagem e o suporte a outros recursos são descritos a seguir.

2.5.2.1. Linguagem NCL

A linguagem NCL, baseada no modelo NCM (*Nested Context Model*), é descrita por SOARES *et al.* (2007) como uma aplicação XML que segue uma abordagem de modularização. Nessa forma de abordagem, cada módulo está associado a um conjunto de elementos XML com atributos, que representam uma unidade de funcionalidade. Para o padrão ISDB-Tb, versão brasileira do padrão japonês ISDB-T, o sistema Ginga-NCL adota dois perfis, que são combinações de módulos. Os dois perfis de linguagem são: BDTVProfile (*Basic Digital TV Profile*), que conta com um conjunto básico de

módulos, e EDTVProfile (*Enhanced Digital TV Profile*), que conta com alguns módulos extras além do perfil básico.

NCL é uma linguagem que apresenta uma divisão bem clara entre conteúdo e estrutura. Nessa linha, uma mídia em si não é definida em um documento NCL, mas sim como objetos de mídia estruturados e relacionados, tanto em termos espaciais como temporais. Os tipos de mídias não são limitados pela linguagem, podendo haver objetos de imagem, vídeo, áudio, texto, execução, que são, no caso, *scripts* em linguagem procedural (Xlet, Lua), etc.

Algumas das características de NCL são explicadas pelas funções de seus elementos. O sincronismo espaço-temporal generalizado é definido por elos (*links*). A adaptabilidade é definida pelos elementos *NCL switch* e *descriptor*. Outro recurso, o suporte a múltiplos dispositivos de exibição, deve-se aos elementos *NCL regions* (SOARES *et al.*, 2007).

A fim de descrever a estrutura geral da linguagem, alguns de seus elementos são citados, seguindo a hierarquia estabelecida pelos módulos.

O elemento raiz `<ncl>` tem como filhos `<head>` e `<body>`, assim como outros padrões W3C (*World Wide Web Consortium*).

O elemento `<head>` pode ter os seguintes elementos como filhos:

- `<importedDocumentBase>`

  Informa documentos NCL a serem importados.

- `<ruleBase>`

  Contém elementos `<rule>` com operadores de comparação e valores para a adaptação de conteúdo ao longo da execução de uma aplicação.

- <transitionBase>

  Inclui elementos <transition> para definir efeitos de transição entre objetos de mídia. Tal especificação é referenciada nos elementos <descriptor>.

- <regionBase>

  Permite definir em que classe de dispositivos os objetos de mídia serão exibidos. Inclui elementos <region> definindo os locais de exibição para cada mídia.

- <descriptorBase>

  Inclui elementos <descriptor> para definir parâmetros de exibição de objetos de mídia em determinadas regiões. É possível reutilizar essa base em outros documentos NCL.

- <connectorBase>

  Pode incluir elementos <causalConnector>, para definir conectores, estabelecendo sincronismo e o comportamento de interatividade entre os objetos, e elementos <importBase>, para importar uma base de conectores de outro arquivo.

- <meta>

  Permite definir metadados simples, com informações sobre o conteúdo a ser exibido.

- <metadata>

  Permite definir metadados mais estruturados, em forma de árvores.

Já o elemento <body> pode incluir os seguintes elementos em sua hierarquia:

- <port>

Atua como ponto de interface de um contexto para oferecer acesso externo ao conteúdo. Deve haver pelo menos uma porta em um documento NCL a fim de indicar qual o objeto inicial.

- `<property>`

  Serve para definir um atributo de um nó ou grupo de nós.

- `<media>`

  Específica objetos de mídia ou conteúdo, fazendo referências aos arquivos de origem da mídia e ao descritor apropriado.

- `<context>`

  Permite agrupar objetos de mídia, `<switch>` ou de contexto, além de elos.

- `<switch>`

  Elemento composto por nós alternativos, dentre os quais um será selecionado. A decisão sobre a seleção fica a cargo de regras definidas em elementos `<bindRule>`. Comportamentos *default* também podem ser implementados com elementos `<defaultComponent>`, caso alguma regra não seja satisfeita para selecionar algum nó.

- `<link>`

  Representa elos na aplicação, associando objetos por meio de conectores. Se a condição de um conector relacionado ao elo for satisfeita, ações são disparadas.

A Figura 2 apresenta a estrutura de um documento com a linguagem NCL e a Figura 3 mostra um exemplo de um documento NCL, ilustrando o uso de alguns dos elementos destacados acima. Tal documento NCL representa uma aplicação que executa vídeos específicos em sequência e em *loop*. Uma descrição mais detalhada desses e de

outros elementos da linguagem NCL pode ser encontrados nos trabalhos de (SOARES & BARBOSA, 2009) e (SOARES *et al.*, 2007). Através de um dos elementos <media>, é possível executar *scripts* Lua por meio de uma aplicação em NCL. Uma breve descrição dessa técnica é realizada na seção a seguir.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="exemplo01"
 xmlns="http://www.telemidia.puc-rio.br/specs/xml/NCL23/profiles"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.telemidia.puc-
rio.br/specs/xml/NCL23/profiles/NCL23.xsd">
      <head>
            <regionBase>
                  <!-- regiões da tela onde as mídias são
apresentadas -->
            </regionBase>
            <descriptorBase>
                  <!-- descritores que definem como as mídias são
apresentadas -->
            </descriptorBase>
            <connectorBase>
                  <!-- conectores que definem como os elos são
ativados e o que eles disparam -->
            </connectorBase>
      </head>
      <body>
            <port id="pInicio" component="ncPrincipal"
interface="iInicio"/>
                  <!-- contextos, nós de mídia, elos e outros
elementos -->
      </body>
</ncl>
```

Figura 2. Estrutura da linguagem NCL (adaptada de (RATAMERO, 2007)).

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<ncl id='videoList' title='videoList'
xmlns='http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd'>
      <head>
            <regionBase>
                  <region id='rgTV' width='1280' height='720'/>
            </regionBase>
            <descriptorBase>
                  <descriptor id='dVideo' region='rgTV'/>
            </descriptorBase>
```

Figura 3. Exemplo de documento NCL.

```
            <connectorBase>
                  <causalConnector id='onEndStart'>
                        <simpleCondition role='onEnd'/>
                        <simpleAction role='start'/>
                  </causalConnector>
            </connectorBase>

      </head>
      <body>
            <port id='pStart' component='cVideos'/>
            <context id='cVideos'>
                  <port id='pVideo0' component='video0'/>
                  <media id='video0' src='video0.mp4'
type='video/mpeg' descriptor='dVideo'/>
                  <media id='video1' src='video1.mp4'
type='video/mpeg' descriptor='dVideo'/>
                  <media id='video2' src='video2.mp4'
type='video/mpeg' descriptor='dVideo'/>
                  <media id='video3' src='video3.mp4'
type='video/mpeg' descriptor='dVideo'/>
                  <link id='startNext0' xconnector='onEndStart'>
                        <bind role='onEnd' component='video0'/>
                        <bind role='start' component='video1'/>
                  </link>
                  <link id='startNext1' xconnector='onEndStart'>
                        <bind role='onEnd' component='video1'/>
                        <bind role='start' component='video2'/>
                  </link>
                  <link id='startNext2' xconnector='onEndStart'>
                        <bind role='onEnd' component='video2'/>
                        <bind role='start' component='video3'/>
                  </link>
                  <link id='startFirst' xconnector='onEndStart'>
                        <bind role='onEnd' component='video3'/>
                        <bind role='start' component='video0'/>
                  </link>
            </context>
      </body>
</ncl>
```

Figura 3 (continuação). Exemplo de documento NCL.

2.5.2.2. Lua

Lua é uma linguagem de propósito geral voltada para programação procedural com recursos para descrição de dados. Uma de suas principais características diz respeito à inexistência de um programa principal para sua execução, devendo sempre ser incorporada a um cliente (IERUSALIMSCHY *et al.*, 1996).

Os *scripts* Lua podem ser adotados no subsistema Ginga-NCL a fim de implementar objetos procedurais em documentos NCL. Mais especificamente chamada de NCLua, por conter alguns módulos existentes na linguagem Lua, mas também por contar com módulos próprios, essa linguagem dispõe de funções com comandos de edição, ou seja, possibilita a manipulação dos elementos definidos no documento NCL em tempo de execução.

Combinando uma sintaxe procedural com declarativa, a linguagem Lua tem a simplicidade como uma de suas características mais marcantes. Com uma pequena quantidade de comandos primitivos, sua implementação acaba sendo eficiente e leve, ao mesmo tempo. O alto grau de portabilidade oferecido, aliado a sua licença livre, contribuíram para sua adoção no *middleware* Ginga.

Conforme destacam SOARES & BARBOSA (2009), a fim de se adaptar ao cenário de TV Digital, a linguagem Lua foi estendida com novas funcionalidades, adicionando os seguintes módulos à biblioteca-padrão da linguagem: *event*, *canvas*, *settings* e *persistent*. Por outro lado, algumas funções apresentam dependência sobre a plataforma de execução e foram removidas do padrão, como: função *loadlib* do módulo *package*; módulo *io*; as funções *clock*, *execute*, *exit*, *getenv*, *remove*, *rename*, *tmpname* e *setlocale* no módulo *os*.

Todas as funções podem receber como parâmetro um valor com referência de tempo, indicando em que momento tal comando deve ser executado. Outra forma de controle existente é fornecida por funções com comandos de *start*, *stop*, *pause* ou *resume*, gerenciando a execução de objetos de mídia em documentos NCL. Por último, trechos de código NCLua podem se registrar como ouvintes de transição de estados,

permitindo a execução de uma função na ocorrência de um evento NCL (SOARES *et al.*, 2007).

2.5.3. Ginga-J

Ginga-J é o subsistema responsável por processar objetos Xlet, implementados em Java. Com isso, ele se diferencia do subsistema Ginga-NCL, que trata de documentos NCL declarativos. O ambiente procedural faz uso de uma *Java Virtual Machine* (JVM) para executar as aplicações.

Além disso, o ambiente Ginga-J incorpora algumas inovações, porém consegue manter compatibilidade com a maioria dos *middlewares* de TV Digital, aderindo a um padrão amplamente aceito, o GEM (*Globally Executable MHP*) (SOUZA FILHO *et al.*, 2007).

Conforme SOUZA FILHO *et al.* (2007) colocam, o sistema brasileiro de TV Digital foi inaugurado considerando a convergência entre mídias, com transmissões tanto em alta definição como em definição padrão, para televisões e dispositivos móveis. A especificação do Ginga-J também prevê tal convergência, incluindo suporte a tecnologias de comunicação como Bluetooth, Wi-Fi, Infravermelho etc. Com isso, diferentes dispositivos podem ser usados para interagir com uma mesma aplicação ao mesmo tempo.

Após reconhecer as principais características de aplicações interativas de TV Digital, é importante analisar técnicas voltadas para o seu desenvolvimento. Alternativas que viabilizem reutilização se colocam como uma escolha de grande valor.

3. Reutilização e Desenvolvimento de *Graphical User Interfaces*

Considerada uma prática de grande importância na Engenharia de *Software*, a reutilização é passível de aplicação em diversos domínios, como a das aplicações interativas para TV Digital.

A reutilização é caracterizada pela utilização de produtos de *software* em um contexto diferente do contexto original (FREEMAN, 1980 *apud* MILER JÚNIOR, 2000). Ela visa à possibilidade de reaproveitar componentes como código, especificações, projeto, plano de testes, dentre outros.

Este capítulo aborda algumas técnicas comuns em reúso de *software*, de modo a contextualizar a proposta principal desse estudo exposta no Capítulo 5: uma *Application Programming Interface* (API) para o desenvolvimento de interfaces de usuário no ambiente de aplicações para TV Digital. Dessa forma, a Seção 3.1 aborda o papel de APIs em processos de implementação de projetos de *software*. A Seção 3.2 apresenta uma discussão relacionada a vantagens e desvantagens no uso de *frameworks*. Por último, a Seção 3.3 analisa o uso de tais alternativas voltadas para o desenvolvimento de elementos de interface gráfica, como aplicações para criação de *Graphical User Interfaces* (GUIs).

3.1. APIs

A adoção de bibliotecas de código no processo de desenvolvimento de *software* é uma atividade comum e deve ser cada vez mais incentivada, dados os benefícios da reutilização neste contexto. Grandes sistemas são baseados em coleções reutilizáveis de

funcionalidades implementadas na forma de bibliotecas (KAWRYKOW & ROBILLARD, 2009).

No entanto, tais funcionalidades necessitam de uma interface que forneça o serviço esperado de tudo o que foi implementado na coleção em questão. A fim de exercer essa função, as APIs permitem aos desenvolvedores acessar bibliotecas e aplicar os serviços providos pelas mesmas. Dentre suas principais características, as APIs fornecem abstrações em alto nível, suportam reutilização de código (sua essência), e colaboram de forma a simplificar a fase de programação, uniformizando algumas tarefas (ROBILLARD, 2009).

Apesar de ser uma alternativa bem comum em processos de desenvolvimento, diversas questões são colocadas durante o uso de APIs em projetos, de forma a reduzir as dificuldades em sua utilização e otimizar a codificação. As questões variam desde falta de documentação a uma API pouco intuitiva.

Em alguns casos, bibliotecas possuem funcionalidades de grande valor e sua utilização poderia ser de grande valia para os desenvolvedores que a adotam, porém a documentação é insuficiente e em muitas vezes faltam exemplos de código-fonte (ALNUSAIR *et al.*, 2010). No que diz respeito a essa última questão, com tal tipo de adição à documentação, os programadores podem inserir o código do exemplo diretamente no projeto e apenas adaptar às suas necessidades.

Em bibliotecas de grande porte, até mesmo programadores experientes costumam não conhecer todas as funcionalidades fornecidas através das APIs. Esse ponto ainda se agrava quando a API não atende aos requisitos específicos do desenvolvedor. Como consequências, STYLOS *et al.* (2008) apontam que os desenvolvedores podem passar um longo período tentando fazer outras APIs

funcionarem em conjunto com a atual e acabar escrevendo código a partir do zero em vez de usar uma API complexa.

Outra questão que é comum em desenvolvimento aliado ao uso de bibliotecas é a gerência de configuração, mais especificamente o que diz respeito à evolução das APIs. Em certas ocasiões, as APIs são atualizadas, porém os projetos que se baseiam nelas não. Isso não representa problemas de compatibilidade, obrigatoriamente, pois às vezes as APIs são desenvolvidas observando-se a sua compatibilidade com versões anteriores. Porém, a qualidade do código fica em risco, pois muitas funcionalidades acabam não sendo utilizadas e observam-se a implementação, por parte do consumidor da API, de serviços já fornecidos pela biblioteca. Isso pode levar a problemas como baixa modularidade, desempenho abaixo do ótimo e até obsolescência, arriscando a manutenibilidade do sistema a longo prazo (KAWRYKOW & ROBILLARD, 2009).

De forma resumida, uma diretriz que deve ser adotada no desenvolvimento de uma API é a de buscar que o cliente não faça nada que a biblioteca já faça. O cliente deve usar a API da maneira mais eficiente possível.

3.2. *Frameworks*

Além de APIs, abordadas na Seção 3.1, *frameworks* também representam uma forma de reúso de código. Neste caso, *frameworks* orientados a objetos promovem reutilização em larga escala.

Diferentemente de bibliotecas de classes, *frameworks* acrescentam os relacionamentos entre as classes, fornecendo para uso o projeto abstrato referente ao

domínio de aplicação, como uma infraestrutura de soluções para um conjunto de problemas (MALDONADO *et al.*, 2002).

Frameworks, além de otimizarem a produtividade e a manutenção de projetos, também representam uma ótima alternativa para reutilização de *software*. Eles permitem reusar análise, descrevendo os objetos importantes e dividindo um problema maior em casos menores. A fase de projeto também é reutilizada ao contar com algoritmos abstratos, interfaces e restrições. Por fim, *frameworks* também permitem reusar código, pois componentes podem ser implementados, herdando das superclasses abstratas fornecidas (JOHNSON, 1991 *apud* MALDONADO *et al.*, 2002).

Uma característica importante de *frameworks* que os diferem de bibliotecas de código diz respeito ao controle interno. Eles funcionam como estruturas extensíveis, pois os métodos definidos pelo usuário são chamados de dentro do próprio *framework*, controlando o fluxo de execução.

A forma como os métodos são adicionados às subclasses das classes do *framework* também o diferencia quanto ao seu o tipo. O chamado *framework* caixa branca é caracterizado pelo usuário criando subclasses das classes abstratas existentes no *framework*, promovendo o reúso por herança. No tipo caixa preta, a reutilização ocorre por composição, cabendo ao usuário combinar classes concretas do próprio *framework* para gerar a aplicação.

Em termos de desenvolvimento, a tarefa de projetar um *framework* do tipo caixa branca é mais simples, não havendo necessidade de prever as alternativas de implementação possíveis, diferentemente do *framework* caixa preta. Já em termos de uso, o tipo caixa branca é mais complexo, sendo necessário realizar as implementações

completas, ao contrário do *framework* caixa preta, sendo preciso apenas escolher a implementação desejada (MALDONADO *et al.*, 2002).

Assim como ocorre com APIs, o uso de *frameworks* esbarra em alguns obstáculos. A reutilização com *frameworks* pode ser complicada para desenvolvedores que estão tendo seu primeiro contato com um *framework* específico, devido à grande quantidade de informações em termos de comportamento e estrutura, o que pode exigir muito esforço por parte dos desenvolvedores para compreenderem todos os conceitos envolvidos. Além disso, uma documentação inadequada, ou mesmo sua inexistência, dificulta ainda mais o reúso, limitando a adoção de *frameworks* (KIRK *et al.*, 2002). A abstração dos componentes reutilizados no uso de um *framework* costuma ser maior do que o que ocorre com APIs. Por serem mais específicos a determinado domínio de aplicação, a adoção de *frameworks* acaba ocorrendo em menor nível do que se comparado a APIs, que são mais genéricas.

3.3. Elementos de Interface Gráfica

As alternativas visando o reúso de *software* são aplicáveis a diferentes áreas, desde aplicações tradicionais em ambientes *desktop* até aplicações para TV Digital. No que diz respeito às aplicações interativas de TV, um campo que merece destaque é o de elementos de interface gráfica.

Em um ambiente como o da TV Digital, cuja interação se dá basicamente pelo controle remoto, não tendo à disposição um teclado ou um *mouse*, o projeto da interface gráfica deve levar diversos aspectos em consideração, a fim de produzir uma aplicação intuitiva. Nesse sentido, o desenvolvimento de interfaces para o usuário deve ter como

preocupação quais os dispositivos em que tais interfaces serão visualizadas, os diferentes grupos de usuários, e os variados ambientes de uso (WANG *et al.*, 2006).

3.3.1. Classificação de Elementos de Interface

De forma a melhor compreender como ocorre a interação entre usuário e um elemento de interface gráfica, aqui tratado como *widget*, é importante consultar um modelo de interação entre humanos e ambientes virtuais. WANG *et al.* (2006) propõem um modelo no qual usuários seguem um processo de interação com um computador. Ao iniciar o processo, o usuário tem um objetivo em mente e avança para um estado seguinte, em que há uma escolha a ser feita, com duas situações possíveis. Uma delas é oportunista e incerta, havendo necessidade de explorar mais. A outra situação é certa e intencional, com escolhas óbvias e sem dúvidas quanto a qual ação tomar. Quando o usuário toma uma ação, o computador irá processar e enviar o *feedback*. Nesse modelo, existem três tipos de *feedback*: progresso, controle de sistema e saída. O progresso exibe como a execução da ação está. O controle de sistema indica ao usuário que o mesmo deve confirmar escolhas entre ações como “OK” ou “Cancelar”.

O modelo descrito é utilizado no desenvolvimento de uma classificação dos elementos. *Tags* XML são utilizadas para identificar as categorias. Quatro cenários base servem de abstração para os processos de interação humano-computador. A hierarquia de classificação proposta por (WANG *et al.*, 2006) é exibida a seguir:

- `<observar>`

  Usuários observam o que o computador exibe para eles.

  - `<agrupamento-info>`

    Atua como contêiner para *widgets* ou informação.

- <apresentação-info>

  Exibe informação de forma estruturada para os usuários.

- <guia-usuário>

  Direciona usuários para explorar o que desejarem.

- <tomada de decisão>

  Usuários informam aos computadores o que pensam e inserem isso no computador.

- <seleção>

  Ajuda usuários a selecionarem objetos de forma que o computador saiba sua intenção.

- <edição>

  Possibilita aos usuários adicionarem ou removerem objetos dos computadores.

- <ação>

- Usuários inserem um comando e o computador o executa.

- <feedback>

  Um tipo de saída do sistema.

- <mensagem>

  Saída em formato de informação.

- <indicador>

  Saída em formato de indicação como sucesso, falha ou estado intermediário para uma execução de comando.

- <consulta>

  Saída em formato de resultado de consulta.

Com essa hierarquia, alguns *widgets* mais comuns em interfaces gráficas podem ser classificados. A Tabela 1 apresenta uma classificação para alguns elementos de interface gráfica.

Classificação	***Widgets***
<ação> ... </ação>	Botão Caixa de seleção (*check box*)
<tomada de decisão><edição> ... </edição></tomada de decisão>	Caixa de texto
<tomada de decisão><seleção> ... </seleção></tomada de decisão>	Botão de opção (*radio button*) Grupo de caixas de seleção (*check boxes*)
<*feedback*> ... </*feedback* >	Barra de progresso
<observação><agrupamento-info> ... </agrupamento-info></observação>	Barra de menu Barra de status Barra de ferramentas Formulário
<observação><guia-info> ... </guia-info></observação>	Controle de rótulo
<observação><apresentação-info>...</apresentação-info></observação>	Imagem
<apresentação-info> + <ação>	Controle de diálogo
<apresentação-info> + <seleção>	Caixa de listagem
<seleção> + <edição>	Caixa de combinação (*combo box*)

Tabela 1. Classificação de *widgets* (adaptada de (WANG *et al.*, 2006)).

Tal classificação permite estruturar um processo de escolha de *widgets* para uma determinada aplicação. Com as características de cada elemento, o desenvolvedor da interface gráfica para usuário de uma aplicação deve decidir qual deles atende melhor os requisitos estabelecidos.

3.3.2. GUIs

Ainda na área de interação humano-computador, as chamadas *Graphical User Interfaces* (GUIs) representam um modelo de interface que podem adotar elementos gráficos como janelas, menus, ícones, dentre outros, para manipulação pelo usuário. Tal manipulação pode ser realizada através de diferentes formas, como por *mouse* e teclado em um computador, pelo teclado numérico, diretamente na tela sensível ao toque em um *smartphone*, ou pelo controle remoto em um aparelho de TV Digital. Os meios de manipulação e interação podem variar dependendo do ambiente em questão.

As GUIs representam uma forma de interface totalmente oposta às interfaces por linha de comando, que acompanham alguns sistemas operacionais. Tais interfaces são acessíveis somente por meio do teclado.

Uma das maiores motivações em se adotar uma GUI para um sistema ou aplicação diz respeito a tornar seu uso mais intuitivo, de forma que o usuário possa interagir e realizar tarefas de modo simples, sem ter que decorar vários comandos preestabelecidos. O aspecto de tornar o uso mais intuito é também possível pelos eventos de *feedback*, comuns em interfaces gráficas. Essas características colaboram para melhorar a usabilidade de aplicações.

É importante, no entanto, ressaltar que muitas das características comuns à GUIs, atualmente, só foram incorporadas ao longo do tempo. Seu histórico, descrito brevemente na Seção 3.3.2.1, revela aspectos interessantes de sua origem.

3.3.2.1. Breve Histórico

Os conceitos iniciais de GUIs foram introduzidos por Vannevar Bush, enquanto cientista no *Massachusetts Institute of Technology* (MIT) durante a Segunda Guerra Mundial. Em um artigo de 1945, intitulado "*As We May Think*", ele propunha uma ferramenta para gerenciamento de informação denominada *Memex*, que possibilitaria armazenar dados em um microfilme de forma acessível, com hiperlinks e programável (*THE LINUX INFORMATION PROJECT*, 2004).

Apesar dos conceitos remeterem à década de 1940, a primeira GUI foi desenvolvida por Ivan Sutherland para seu tema de Ph.D. em 1963. A GUI fazia parte de um *software* chamado Sketchpad, que permitia a manipulação de objetos gráficos diretamente na tela, utilizando uma caneta própria. Sua proposta incluía conceitos como aumentar e diminuir o zoom na tela, além de desenhar linhas precisas e com curvas.

Em termos comerciais, a primeira GUI utilizável foi implementada pela empresa Xerox, em 1974, na cidade de Palo Alto, Califórnia. A GUI foi incorporada a duas versões de computadores da empresa, porém estas não fizeram sucesso, provavelmemnte devido aos altos preços dos equipamentos.

Posteriormente, Steve Jobs, um dos fundadores da Apple Computer, decidiu adotar as propostas de GUI nos computadores da própria empresa, estendendo o trabalho começado pela Xerox. O Apple Macintosh foi o primeiro computador considerado um sucesso de vendas e que adotava uma GUI. Elementos como janelas, barras, menus e ícones já estavam presentes no sistema.

Em 1985, foi lançado o Windows 1.0, primeiro sistema operacional da Microsoft com uma GUI incorporada. Porém, somente em 1995, com o lançamento do Windows

95, a empresa conseguiu oferecer uma GUI que veio a promover um relativo sucesso comercial.

3.3.2.2. Desenvolvendo GUIs

Desenvolver GUIs engloba algumas questões, como a definição do leiaute de cada componente, suas dimensões máximas e mínimas, seus eventos, a integração da interface com o resto do projeto, bem como a sua portabilidade para diferentes sistemas (SCODITTI & STUERZLINGER, 2009).

Ao se escolher o conjunto de *widgets* que farão parte da GUI, duas características principais de cada elemento devem ser planejadas: a aparência e a sensibilidade (PABLO & PETRI, 2002). A aparência de um elemento gráfico é a sua apresentação visual, i.e., se será estático, dinâmico, com animações etc. A sensibilidade denota o comportamento do *widget* à interação do usuário da aplicação, i.e., qual será a reação a uma ação do usuário.

Outros aspectos, como alinhamento dos *widgets*, comportamento quando a janela que os contém tem seu tamanho alterado e efeitos gráficos esperados também precisam ser planejados. Nesse ponto, durante a fase de desenvolvimento, pode haver dois pontos de vista que precisam ser analisados: o ponto de vista do *designer* gráfico, que terá o foco voltado para uma aparência adequada da interface e seus elementos, e o ponto de vista do programador, que estará voltado para o funcionamento correto do sistema. Os dois pontos de vista devem ser considerados, e é importante que sejam integrados, de modo que não se priorize um, em detrimento do outro, o que pode afetar a usabilidade da aplicação.

Existem algumas ferramentas disponíveis no mercado voltadas para a construção de interfaces gráficas. Um exemplo é a API Swing (JAVA SE DESKTOP, 2012), que possibilita adicionar e customizar *widgets* em aplicações desenvolvidas com a linguagem Java. Outro exemplo é o *framework* Struts (APACHE STRUTS, 2012), que usa restrições para definir a posição de componentes de forma independente entre eles. Além desses, também é possível citar a ferramenta Interface Builder integrada à IDE Xcode (XCODE, 2012), voltada para a construção de interfaces, desenvolvendo a GUI em aplicações para MAC OS X. As APIs GTK+ (GTK, 2012) e Motif (MOTIFZONE, 2012) permitem desenvolver GUIs em ambientes Linux, enquanto a Common Controls Library (COMMON CONTROLS, 2012) provê recursos para utilização de elementos de interface em sistemas operacionais Windows.

Uma alternativa de desenvolvimento de GUIs para aplicações de TV Digital é o *framework* ftv (PABLO & PETRI, 2002). Ele segue a especificação HAVi, incluída no modelo DVB-MHP, adotado como padrão de TV Digital em alguns países, sobretudo na Europa. Tal *framework* foi desenvolvido em Java, baseando-se no pacote java.awt, destinado à criação e manipulação de elementos de interfaces gráficas.

Ferramentas desse formato para o ambiente Ginga-NCL ainda representam um volume pequeno de ofertas. Um trabalho em desenvolvimento que se assemelha à proposta desse estudo é o projeto SAGGA (*Support to Automatic Generation of Ginga Applications*) (LABORATÓRIO TELEMÍDIA – PUC-RIO, 2012), que visa a criação de uma infraestrutura para geração automática de aplicações NCL 3.0, mais especificamente o subprojeto SAGGA4, que propõe a criação de *widgets* para serem incorporadas às aplicações NCL geradas pelos outros subprojetos, SAGGA1, SAGGA2 e SAGGA3. Diferentemente desses projetos, nos quais os elementos de interface seriam aplicações NCL isoladas a serem incorporadas através do subprojeto SAGGA2, no

presente trabalho os *widgets* são inseridos por meio de objetos NCLua diretamente no documento NCL da aplicação principal.

Nesse contexto, a proposta deste trabalho, que é voltada para o padrão brasileiro de TV Digital, é apresentada no Capítulo 5, visando fornecer uma API para a criação de GUIs em aplicações interativas.

4. *Design Pattern* para TV Digital Interativa

Neste capítulo, é feita uma revisão bibliográfica dos chamados *design patterns* de interação, aqui abordados como padrões de projeto de interface. Tais padrões visuais são usados no desenvolvimento de aplicações interativas para TV Digital e foram considerados na proposta de uma API para desenvolvimento de GUIs, a ser apresentada no Capítulo 5.

Uma visão geral de padrões, além de definições relacionadas, é dada na Seção 4.1. Os conceitos discutidos nessa revisão são baseados, principalmente, no trabalho feito por KUNERT (2009), apresentando na Seção 4.2 os métodos necessários para o desenvolvimento de padrões para aplicações interativas e propondo alguns desses modelos na Seção 4.3.

4.1. *Design Patterns*

Antes de focar na área de aplicações para TV Digital, é importante conhecer a origem do conceito de *design patterns*. Essa ideia vem da arquitetura, com as primeiras sugestões feitas por Alexander (1977 *apud* BORCHERS, 2000). Neste contexto, padrões buscavam transmitir a essência de soluções de sucesso para problemas de *design* existentes em projetos de arquitetura urbana.

Em sua concepção, os *design patterns* se baseavam nas atividades desempenhadas no cenário do projeto para começar a esboçar a solução. Nesse ponto de vista, o arquiteto seria considerado o *designer* e os habitantes daquele cenário seriam os usuários, que, ao final, irão interagir de alguma forma com o artefato desenvolvido pelo *designer*.

Aproveitado por outras áreas, como a engenharia de *software* (GAMMA *et al.*, 1998) e interação humano-computador, os *design patterns* apresentam conceitos que remetem à reutilização, fornecendo as melhores práticas para um determinado conjunto de problemas. Uma outra forma de caracterizar um padrão de projeto é por meio de um par "problema/solução", pois este propõe uma solução para um problema comum em desenvolvimento de projetos (BUSCHMANN, 1996 *apud* MALDONADO *et al.*, 2002).

Um *design pattern* é composto por um *template*, cujos dados a serem preenchidos variam de acordo com o padrão adotado, considerado mais apropriado para documentar o *pattern* em questão. Alguns dos principais itens desses padrões são destacados a seguir (APPLETON, 1997 *apud* MALDONADO *et al.*, 2002):

- **Nome (*Name*)**: item obrigatório em um *pattern*, com uma palavra ou frase referindo-se ao modelo e conhecimento descritos. Subseções "Pseudônimos" ("*Aliases*") ou "Também conhecido como" ("*Also known as*") devem ser adicionadas caso o padrão tenha mais de um nome reconhecido.
- **Problema (*Problem*)**: informa os objetivos do padrão, caracterizando o problema a ser tratado.
- **Contexto (*Context*)**: descreve o estado inicial do cenário, em que tanto o problema como a solução ocorrem.
- **Forças (*Forces*)**: destaca as limitações e impactos relacionados ao problema, funcionando como uma justificativa para a criação de um *pattern*.
- **Solução (*Solution*)**: descreve a solução proposta para o problema tratado, utilizando regras e relacionamentos.
- **Exemplos (*Examples*)**: apresenta aplicações do *design pattern*.

- **Contexto Resultante (*Resulting context*)**: descreve o estado final do cenário, com os resultados da aplicação do padrão, como consequências e efeitos colaterais.
- **Lógica (*Rationale*)**: explica o passo-a-passo da implementação do *pattern*, demonstrando que sua aplicação poderá ser bem-sucedida.
- **Padrões Relacionados (*Related Patterns*)**: apresenta os padrões relacionados ao atual, onde os relacionamentos podem denotar padrões predecessores, sucessores, alternativos e codependentes.
- **Usos Conhecidos (*Known Uses*)**: apresenta casos reais de aplicação do *design pattern*, funcionando como uma validação para o mesmo.

Existem outras metodologias que propõem diferentes *templates* para a descrição de padrões na literatura. Uma delas, voltada para *design* de projetos de *software* orientado a objetos, é encontrada em (GAMMA *et al.*, 1998).

É importante destacar que vários padrões são descritos visando soluções para um mesmo domínio. A partir desse ponto, muitos padrões são identificados em conjunto com outros, de forma a se apoiar nesses para construir uma coleção estruturada de *patterns* e viabilizar soluções a diversos problemas de um domínio específico. Tal coleção é conhecida como linguagem de padrões. Um exemplo de linguagem é apresentado na Seção 4.3.1.

A utilização de *design patterns* na área de *software* permite uma classificação de tais padrões de acordo com o nível de abstração em destaque. MALDONADO *et al.* (2002) apresentam algumas dessas categorias de *patterns*, como:

- ***Pattern* de processo:** apresentam soluções para questões comuns em processos relacionados à engenharia de *software*.

- ***Pattern* de *pattern*:** definem padrões para o formato em que *patterns* devem ser escritos.
- ***Pattern* de projeto:** expressam soluções relacionadas a projetos de *software.*
- ***Pattern* de interface:** sendo considerado um caso particular de *pattern* de projeto, apresentam soluções para problemas encontrados no desenvolvimento de aplicações hipermídia (RUBART, 2008) e interfaces de sistemas. Nesse contexto, também se inserem os chamados padrões de interação.

No que se refere à área de interação humano-computador, foco desse estudo e, de acordo com a classificação apresentada, inserida na categoria de padrões de interface, os *patterns* revelam aspectos semelhantes entre tal área e a arquitetura convencional, como as atividades de *design* e a importância de se considerar a experiência do usuário como um dos fatores principais.

4.2. Desenvolvendo *Design Patterns*

A fim de se seguir um processo de descrição de padrões de projeto de interface, alguns passos devem ser executados. KUNERT (2009) propõe um *framework* para *design patterns*, atuando como referência para o uso de tais modelos. Cinco itens são listados como passos a serem executadas: definição de um critério de qualidade, identificação de problemas de *design*, discussão de soluções alternativas, explicação das soluções e avaliação dos padrões documentados.

O critério de qualidade escolhido segue o padrão comum adotado tanto por pesquisadores como por usuários, a usabilidade.

Identificar os problemas de interação referentes ao projeto de interface é o próximo passo para o uso de padrões. Essa etapa deve ser realizada com cuidado, pois pode gerar dificuldades no futuro. Como, em geral, os padrões possuem um alto nível de abstração e são descritos de forma textual, reconhecer as situações onde esses podem ser reutilizados costuma ser uma tarefa complexa (NANARD *et al.*, 1998).

Nesse contexto, KUNERT (2009) destaca a importância de se considerar a perspectiva de usuários e projetistas na identificação de problemas. Na parte de usuários, deve-se proceder com uma análise em termos de contexto de uso e requisitos e tarefas de usuários. O ponto de vista de projetistas indica uma análise voltada para guias de *design* e requisitos por parte dos mesmos.

A discussão de alternativas se revela outra importante atividade, assinalando vantagens e desvantagens para cada possível solução. Essa análise é vital, pois o projeto da interação é um processo que busca convergir aspectos como requisitos do usuário e contexto de uso. Uma decisão precisa balancear esses e outros elementos pertencentes ao problema em questão.

A melhor forma de se justificar as soluções escolhidas é por meio de testes de usabilidade. A evidência empírica dos resultados colabora para embasar a proposta de solução. É igualmente importante incorporar os resultados dos testes nos padrões, validando as escolhas.

Por fim, o *framework* proposto por KUNERT (2009) prevê a fase de avaliação dos padrões de projeto de interface. Tal avaliação deve ser iterativa, modo normalmente sugerido em processos de desenvolvimento com foco no usuário. Os padrões devem ser avaliados no que se refere aos requisitos do problema, à validade da solução e à relevância do problema apresentado, dentre outros aspectos.

Com o *framework* estabelecido, deve-se proceder para o processo de construção dos padrões propriamente dito.

4.2.1. Métodos de desenvolvimento

Ao propor alguns padrões de projeto de interface, KUNERT (2009) explica detalhadamente todos os métodos utilizados para a construção de tais padrões. Antes de analisar os modelos propostos, deve-se referenciar de forma resumida os métodos citados. Tal passo-a-passo poderá colaborar para uma melhor compreensão dos padrões de projeto de interface a serem utilizados para aplicações interativas de TV Digital.

O primeiro método de todo o processo passa por identificar os requisitos estabelecidos por *designers* com relação à orientação de *design* para forma e conteúdo de aplicações interativas. Nesse ponto, os resultados obtidos por KUNERT (2009) revelam a importância de se usar aspectos específicos no que se refere à orientação de *design*, destacando a insuficiência de guias genéricos na tarefa de se desenvolver interfaces intuitivas em aplicações de TV Digital.

Em termos de conteúdo, os problemas identificados foram classificados de acordo com a seguinte divisão: concepção, arquitetura de informação, leiaute da página, navegação, teclas do controle remoto, funções básicas, apresentação de conteúdo, *design* de texto, entrada de texto, ajuda e busca. Já no que diz respeito aos requisitos de forma, alguns são considerados de grande importância, como: foco em usabilidade, documentação de método de desenvolvimento de guias para orientação em projetos, exemplos de implementação, suporte para encontrar orientação relevante e argumentos para decisão de *design* (KUNERT, 2009).

O próximo método a ser executado está relacionado à criação de uma hierarquia entre os problemas identificados. Tal organização servirá como base para a criação de uma linguagem de *patterns* (ALEXANDER, 1977 *apud* KUNERT, 2009; VAN WELIE & VAN DER VEER, 2003 *apud* KUNERT, 2009). A hierarquia é disposta em diferentes níveis, sendo nove problemas identificados para o nível mais alto da hierarquia, identificado como nível 1. Tais problemas são: leiaute de página, navegação, teclas de controle remoto, funções básicas, apresentação de conteúdo, participação do usuário, entrada de texto, ajuda, acessibilidade e personalização. A fim de possibilitar uma visão mais completa dos problemas e suas classificações, a hierarquia proposta por KUNERT (2009) é exposta nas Tabelas 2 a 10.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Overlay	Posição do *overlay* Tamanho do *overlay* Transparência
Tela inteira com vídeo	Posição do vídeo Tamanho do vídeo Áudio
Tela inteira sem vídeo	Áudio

Tabela 2. Problemas de *design* referentes a “Leiaute de página”.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Menu	Apresentação do menu Total de itens do menu Texto dos itens do menu Posição Teclas de controle remoto Destacamento de itens selecionados no menu Transparência Diferentes tipos de menu Pré-visualização para item de conteúdo

	Apresentação de múltiplos níveis de menu
Vídeo multi-tela	Leiaute Número de transmissões de vídeo Tamanho das transmissões de vídeo Cabeçalhos Teclas de controle remoto Destaque de transmissão de vídeo selecionada Áudio
Índice	Leiaute de página Estrutura Teclas de controle remoto Destaque de item selecionado
Números de páginas	Números Teclas de controle remoto Indicador em tela
Abas	Apresentação de abas Texto de rótulos em abas Teclas de controle remoto

Tabela 3. Problemas de *design* referentes a “Navegação”.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Teclas de setas	Indicadores de navegação em tela
Tecla OK	Indicadores de navegação em tela
Teclas de cor	Indicadores de navegação em tela
Teclas de números	Indicadores de navegação em tela
Teclas especiais	Indicadores de navegação em tela

Tabela 4. Problemas de *design* referentes a “Teclas de controle remoto”.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Chamada inicial para ação	Método de acesso Criação de consciência Indicador de navegação em tela Posição do indicador de navegação

	Duração do indicador em tela
Inicializações	Disponibilidade Total de opções Tecla de controle remoto
Carregamento de indicação	Leiaute de página Posição do indicador de status
Saída	Disponibilidade Tecla de controle remoto Indicador de navegação em tela Posição do indicador em tela Cancelar opção
Ocultamento de aplicação	Disponibilidade Tecla de controle remoto Posição do indicador em tela Cancelar opção
Ir um nível acima	Tipo de função Disponibilidade Tecla de controle remoto

Tabela 5. Problemas de *design* referentes a “Funções básicas”.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Design de texto	Tipo de fonte Tamanho da fonte Cor Espaçamento entre linhas Comprimento de texto Estrutura de texto
Caixa de conteúdo	Tipos de mídia apresentados Leiaute de página Transparência Áudio
Paginação	Teclas de controle remoto Posição de indicadores de navegação em tela

	Posição de indicador
Rolagem	Tipo de apresentação de conteúdo Direção de rolagem Teclas de controle remoto Posição de indicadores de navegação em tela Posição de indicador
Alternância entre itens de conteúdo	Teclas de controle remoto Posição de indicadores de navegação em tela Posição de indicador
Sincronização de conteúdo	Método de acesso a conteúdo Indicador em tela Duração de indicador em tela Indicação de tempo restante

Tabela 6. Problemas de *design* referentes a “Apresentação de conteúdo”.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Votação e perguntas de múltipla escolha	Número de respostas possíveis Total de opções de respostas Apresentação de opções de respostas Teclas de controle remoto Destaque de resposta selecionada Cancelar opção Indicação de resposta correta
Alocação de itens	Total de itens por conjunto Apresentação de itens Teclas de controle remoto Destaque de item selecionado Cancelar opção Indicação de alocações corretas
Completar texto	Apresentação de opções corretas Total de opções de respostas Total de lacunas por página Teclas de controle remoto

	Destaque de lacuna selecionada Destaque de resposta selecionada Cancelar opção Indicação de respostas corretas
Aprovação para conectividade	Opções fornecidas Teclas de controle remoto Destaque de opção selecionada

Tabela 7. Problemas de *design* referentes a “Participação do usuário”.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Teclado QWERTY ou alfabético em tela	Tipo de teclado em tela Apresentação de teclado em tela Entrada de caractere Teclas de controle remoto Destaque de caractere selecionado Cursor Correção de caracteres
Teclado de telefone móvel	Teclado com/sem teclado em tela Apresentação de teclado em tela Seleção de caractere Confirmação de caractere Teclas de controle remoto Indicação de caractere selecionado Cursor Correção de caracteres

Tabela 8. Problemas de *design* referentes a “Entrada de texto”.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Instrução em tela	Texto Posição
Seção de ajuda	Ajuda ativa/passiva Acessando e saindo da seção de ajuda Posição do indicador de navegação em tela

	Leiaute de página Conteúdo da seção de ajuda Navegação pela seção de ajuda

Tabela 9. Problemas de *design* referentes a “Ajuda”.

Problema de *design* – Nível 2	Problema de *design* – Nível 3
Acessibilidade	Problemas de acessibilidade Opções de apresentação Teclas de controle remoto para alterar configurações
Personalização	Personalização passiva/ativa Problemas de personalização Apresentação de opções Teclas de controle remoto para alterar configurações

Tabela 10. Problemas de *design* referentes a “Acessibilidade e personalização”.

O método seguinte procura implementar soluções para os problemas identificados relacionados à interação em aplicações para TV Digital por meio de protótipos, que serão utilizados na realização de testes de usabilidade.

Conforme mencionado, o próximo método visa, por meio de testes de usabilidade com a participação direta de usuários, avaliar soluções de *design*, verificando prós e contras de cada proposta. Durante os testes, as métricas relacionadas à usabilidade escolhidas foram: eficácia, medida pela porcentagem de participantes que completavam uma tarefa, eficiência, representada por dados subjetivos de carga mental de trabalho e satisfação, também de caráter subjetivo, representando a satisfação de um usuário ao utilizar uma aplicação.

O próximo passo no processo de desenvolvimento de padrões de projeto de interface busca a criação da linguagem de *patterns*. Apesar de se basear na hierarquia de problemas previamente estabelecida, nem todo problema se reflete num padrão específico. Na proposta de KUNERT (2009), os problemas identificados nos níveis

mais baixos da hierarquia teriam sua solução mapeada para os problemas do nível 2. Além disso, alguns itens do nível 1 não promoveram a criação de um padrão de projeto específico, pois em certos casos os itens eram apenas classificações de problemas e não problemas em si.

Templates também foram desenvolvidos no estudo de (KUNERT, 2009), a fim de facilitar a compreensão e fornecer uma estrutura única para soluções de certos grupos de problemas. O *template* adotado tem os seguintes elementos: nome, exemplos através de capturas de tela, contexto de uso, problema, com as soluções alternativas para o problema, solução, com a indicação de uso de cada alternativa, evidência, como resultados de testes de usabilidade e referências para a literatura e padrões relacionados. A lista com a linguagem de *design patterns* proposta é apresentada na Seção 4.3.1.

Por último, após todos os métodos utilizados, a tarefa final visa avaliar a forma e o conteúdo da linguagem desenvolvida, sendo um método para otimizar os padrões criados. A metodologia empregada incluía entrevista qualitativa semiestruturada com especialistas em *design* de aplicações para TV Digital interativa. Alguns critérios foram utilizados para avaliar tanto o conteúdo como a forma dos padrões, cabendo aos entrevistados avaliar itens como a clareza dos nomes dos padrões de projeto, relevância, completude, utilidade do processo de desenvolvimento, entre outros.

Com todos os métodos descritos neste trabalho, percebe-se que o processo de desenvolvimento de uma linguagem de *design patterns* é iterativo, com diversas fases voltadas para o aperfeiçoamento das soluções, realizando avaliações e corrigindo os modelos propostos em fases anteriores. Ao cumprirem-se todos esses passos, assegura-se que cada padrão poderá compartilhar um vocabulário e entendimento sobre seu

domínio de problemas, tendo o seu uso como uma forma de reutilizar experiências (RUBART, 2008).

4.3. Coleção de *Design Patterns*

A partir do processo de desenvolvimento de padrões de projeto de interface abordado na Seção 4.2, com a criação de uma hierarquia, KUNERT (2009) propõe posteriormente uma abordagem de padrões de projeto de interface para aplicações de TV Digital. Uma linguagem de *patterns* é apresentada na Seção 4.3.1 e uma descrição para cada padrão é explicitada na Seção 4.3.2.

4.3.1. Linguagem de *Patterns*

Uma forma de apresentar os padrões sugeridos é por meio de uma linguagem, com uma classificação em grupos. A lista com os padrões (KUNERT, 2009), divididos em dez grupos, pode ser vista a seguir:

- Grupo A: Leiaute de página
    - A1 - Escolhendo o leiaute de página correto
    - A2 – *Overlay*
    - A3 – Tela inteira com vídeo
    - A4 – Tela inteira sem vídeo
- Grupo B: Navegação
    - B1 – Múltiplas formas para navegar
    - B2 – Menu
    - B3 – Vídeo multi-tela

    - B4 – Índice
    - B5 – Números de página
    - B6 – Abas
- Grupo C: Teclas de controle remoto
    - C1 – Escolhendo as teclas corretas
    - C2 – Teclas de setas
    - C3 – Tecla OK
    - C4 – Teclas de cor
    - C5 – Teclas de números
    - C6 – Teclas especiais
- Grupo D: Funções básicas
    - D1 – Chamada inicial para ação
    - D2 – Iniciando
    - D3 – Carregando indicação
    - D4 – Saindo
    - D5 – Ocultando aplicação
    - D6 – Ir um nível acima
- Grupo E: Apresentação de conteúdo
    - E1 – *Design* de texto
    - E2 – Caixa de conteúdo
    - E3 – Paginação
    - E4 – Rolagem
    - E5 – Alternando entre itens de conteúdo
    - E6 – Conteúdo sincronizado
- Grupo F: Participação do usuário

    - F1 – Múltiplas formas de participação do usuário
    - F2 – Votação e perguntas de múltipla escolha
    - F3 – Alocação de itens
    - F4 – Completar texto
    - F5 – Aprovação para conectividade
- Grupo G: Entrada de texto
    - G1 – Múltiplas formas de entrada de texto
    - G2 – Teclado QWERTY ou alfabético em tela
    - G3 – Teclado de telefone móvel
- Grupo H: Ajuda
    - H1 – Instrução em tela
    - H2 – Seção de ajuda
- Grupo I: Acessibilidade e personalização
    - I1 - Acessibilidade
    - I2 – Personalização
- Grupo J: Grupos de usuários específicos
    - J1 - Crianças

4.3.2. Padrões de Projeto de Interface

Nessa seção, é feita uma breve descrição do item problema do *template* de cada um dos padrões de projeto de interface propostos no trabalho realizado por KUNERT (2009).

4.3.2.1. Grupo A: Leiaute de página

- A1 – Escolhendo o leiaute de página correto

  Existem vários leiautes de página possíveis para aplicações interativas de TV Digital. O grande diferencial entre eles diz respeito a quanto da aplicação cobre a transmissão de vídeo na tela da TV. Os possíveis leiautes considerados aqui são tratados nos padrões A2, A3 e A4.

- A2 – *Overlay*

  Aplicações em *overlay* cobrem uma pequena parte da tela, permitindo uma visualização considerável do vídeo em segundo plano. Com isso, uma pessoa pode utilizar a aplicação e as demais pessoas de um grupo podem continuar acompanhando a programação da TV. É um padrão indicado para apresentar pequenos trechos de texto ou conteúdo específico, relacionado ao programa em exibição.

- A3 – Tela inteira com vídeo

  Aplicações nesse formato normalmente incluem uma região de vídeo cuja dimensão é de cerca de ¼ da dimensão da tela, disposta em um dos cantos da tela da TV. O resto da tela é ocupado pela aplicação, por isso diz-se que esse formato é adequado para aplicações que serão utilizadas por um grupo, apesar que ainda é possível, para quem desejar, continuar acompanhando o programa em exibição, levando em consideração que elementos de dimensão reduzida no vídeo, terão sua visualização dificultada pelo tamanho da região destinada ao vídeo da programação.

- A4 – Tela inteira sem vídeo

Ocupando toda a área disponível para exibição na tela da TV, aplicações interativas nesse formato costumam ser indicadas para serviços permanentes, sem relação com nenhuma programação, já que o vídeo não é visualizado.

4.3.2.2. Grupo B: Navegação

- B1 – Múltiplas formas para navegar

  Modelo para escolher o tipo de elementos de navegação que serão empregados na interface de usuário, a fim de prover acesso a outras funções e componentes de conteúdo. Os possíveis elementos considerados aqui são tratados nos padrões B2, B3, B4, B5 e B6.

- B2 – Menu

  Esse componente permite uma navegação de forma hierárquica a itens de conteúdos e funções. Às vezes, itens em um menu levam a um submenu. Menus podem ser apresentados de forma estática, sem necessitar de ações do usuário para visualizar todos os itens, e dinâmica, como menus *pull-down*. Um aspecto que deve ser considerado ao se disponibilizar um menu diz respeito ao número de itens que serão colocados no mesmo. Uma forma eficaz de se tratar isso está relacionada à memória de curta duração. Em média, um ser-humano pode registrar de cinco a sete itens de informação (KUNERT, 2009).

- B3 – Vídeo multi-tela

  Essa opção dispõe de diversos vídeos sendo exibidos ao mesmo tempo. Com isso, o usuário pode ter uma visão geral de todos os conteúdos disponíveis e selecionar o seu favorito, de forma que este ocupe a tela inteira ou apenas uma região de dimensões maiores.

- B4 – Índice

Essa forma de navegação oferece um índice que permite acesso a uma lista de itens de conteúdo ou funções organizadas em ordem alfabética.

- B5 – Números de página

  Essa opção disponibiliza números com acesso direto a certas páginas de conteúdo, funcionando como um atalho para os usuários.

- B6 – Abas

  Essa forma de navegação permite acesso a funções e itens de conteúdo da mesma forma que menus, porém são familiares a usuários de computadores. As abas podem ser organizadas tanto de forma horizontal como vertical.

4.3.2.3. Grupo C: Teclas de controle remoto

- C1 – Escolhendo as teclas corretas

  Existem vários *design* possíveis para controles remotos. No entanto, algumas teclas são padronizadas, apesar de apresentarem diferenças em termos de formas e dimensão, mas estão presentes em todos os modelos de controle, como as teclas tratadas nos padrões C2, C3, C4 e C5. Alguns controles remotos, por outro lado, possuem teclas diferentes, como as cobertas pelos padrões C6 e D6.

- C2 – Teclas de setas

  Esse modelo trata da análise de vantagens e desvantagens de se usar teclas de setas para navegação. Formando um conjunto clássico de teclas, facilmente encontrado no controle remoto, tais teclas são normalmente utilizadas em conjunto com a tecla OK, para selecionar um item ou função na tela.

- C3 – Tecla OK

  Essa opção trata da tecla OK, normalmente utilizada para confirmar uma seleção feita. Na maioria dos controles remotos, essa tecla fica disposta no meio das

quatro teclas de setas. No entanto, seu rótulo não é sempre o mesmo, podendo apresentar outros nomes além de OK.

- C4 – Teclas de cores

  Esse conjunto de teclas é padronizado, com a ordem das teclas sendo sempre a mesma. Sua disposição no controle remoto, normalmente, é na forma horizontal. Seu uso costuma estar associado à escolha de algum item em um menu na tela da TV.

- C5 – Teclas de números

  Esse padrão trata do conjunto de teclas de números, variando de 0 a 9. Na maioria dos controles remotos, as teclas ficam dispostas em conjuntos de três teclas por linha, totalizando três linhas, com a tecla 0 sendo usualmente colocada centralizada abaixo da linha mais inferior. Pode-se destacar certa dificuldade em se encontrar a tecla correta somente pelo toque, sem olhar para o controle, diferentemente de outras teclas.

- C6 – Teclas especiais

  Esse modelo cobre as teclas especiais, cujas funções diferem das teclas padronizadas tratadas em outros padrões. Exemplos de funções para essas teclas são teclas com rótulos como “i” ou “*Interactive*”, para iniciar aplicações interativas, além de outras possibilidades.

4.3.2.4. Grupo D: Funções básicas

- D1 - Chamada inicial para ação

  Após escolher o leiaute mais adequado, os modos de navegação e as teclas corretas no controle remoto, esse padrão trata do modo como o espectador vai iniciar a sua interação com algum aplicativo na TV. Essa situação levanta

algumas questões, tais como se dará o método de acesso (automaticamente, com a aplicação iniciando diretamente na tela ou se o usuário poderá controlar a apresentação de tal aplicação), além da forma como haverá uma indicação na tela sobre a disponibilidade de se executar uma aplicação.

- D2 – Iniciando

  Esse modelo trata do problema de se iniciar uma aplicação. Essa tarefa deve ser fácil e intuitiva, uma vez que qualquer indício de dificuldade pode fazer com que o usuário desista de prosseguir.

- D3 – Carregando indicação

  A questão abordada aqui diz respeito ao processo de carregamento de uma aplicação ou de uma página da aplicação, pois muitas vezes isso pode demorar consideravelmente por limitações de desempenho do *set-top box* ou algum outro fator. Nesse ponto, uma indicação ao usuário na forma de *feedback* sobre o carregamento em progresso é importante.

- D4 - Saindo

  Esse padrão de projeto busca discutir sobre alguns aspectos relevantes do processo de saída de uma aplicação, como a disponibilidade constante dessa opção, a facilidade de se usar apenas uma tecla do controle remoto para isso, sugestões de localização na tela do indicador de saída e a possibilidade de se cancelar esse processo (por meio de uma confirmação, por exemplo).

- D5 – Ocultando aplicação

  Uma possibilidade que o usuário sempre deveria ter seria a de ocultar uma aplicação, retornando à visualização da programação da TV, e não sair da aplicação em questão. Com isso, o usuário não precisaria reiniciar tal aplicação, o que tomaria mais tempo. Com uma aplicação oculta, existe a possibilidade de

se indicar isso por meio de um menu reduzido, com a indicação de teclas a serem utilizadas para que a aplicação volte ao primeiro plano.

- D6 – Ir um nível acima

  Uma função importante para uma navegação eficiente é a de ir um nível acima. Essa função difere da função “Voltar”, pois esta está relacionada a retornar à página em termos de histórico de ações do usuário, enquanto que ir um nível acima faz referência à hierarquia da estrutura de página da aplicação. Isso é importante caso o usuário se perca durante a navegação em uma aplicação interativa.

4.3.2.5. Grupo E: Apresentação de conteúdo

- E1 – *Design* de texto

  Esse modelo trata da importância de se escolher uma fonte adequada para a leitura de texto em uma TV, pois muitas vezes, com a resolução baixa de telas, além de outros aspectos, algumas fontes são mais difíceis de serem lidas que outras. Outras características de um texto, como tamanho da fonte, cor, espaçamento entre linhas e até mesmo o tamanho do texto, são aspectos igualmente importantes ao se planejar uma aplicação de TV que faça uso de elementos de texto.

- E2 – Caixa de conteúdo

  Uma caixa de conteúdo representa um espaço na tela onde itens de conteúdo são apresentados. Alguns itens podem ser mais bem utilizados em conjunto, a fim de servirem como forma de descrever mais adequadamente um conteúdo, como texto com imagens ou vídeos juntos. Outros aspectos, como a transparência da caixa, devem ser ajustados de acordo com o elemento sendo apresentado, de

forma a permitir um contraste adequado para a visualização. Também é sugerido que o áudio da programação de TV seja audível durante a aplicação para que o usuário possa acompanhar tal programação de alguma forma.

- E3 – Paginação

  Esse é um modo de se dividir itens de conteúdo em diversas páginas, como várias imagens ou textos longos, a fim de permitir uma visualização adequada dos elementos, o que não seria possível em apenas uma página. Com isso, outros elementos, como indicadores de navegação entre as páginas e as teclas do controle remoto que serão utilizadas para isso, devem ser considerados conjuntamente.

- E4 – Rolagem

  Uma forma de se possibilitar a visualização adequada de itens que não cabem em apenas uma página é por meio da rolagem. Diferente da paginação (E3), cuja navegação faz com que se carregue uma nova página por completo, a rolagem vai mostrando uma nova linha por vez, no caso de um texto longo. Esse é um recurso familiar aos usuários acostumados com documentos e páginas *web* em computadores. Alguns aspectos como indicação e até mesmo a direção da navegação devem ser observados.

- E5 – Alternando entre itens de conteúdo

  Esse modelo pode ser ineficientemente implementado usando duas ações: indo um nível acima (D6) e então escolher o próximo item. Para otimizar essa tarefa, se entre todos os itens de um mesmo nível houvesse um *link*, uma navegação mais rápida seria possível. Com isso, teclas do controle remoto poderiam ser utilizadas, tais como as teclas de setas para ir para o item anterior ou em

sequência, ou até mesmo as teclas de números, permitindo ao usuário ir diretamente para um item específico.

- E6 – Conteúdo sincronizado

  Esse padrão trata dos itens de conteúdo ou funções que ficam disponíveis somente durante certo período de tempo, sendo estritamente relacionado a um programa de TV, por exemplo. Essa possibilidade permite uma integração maior da aplicação com a programação da TV. Alguns aspectos têm que ser considerados, como a forma de se iniciar a aplicação, a indicação da disponibilidade do conteúdo, com até mesmo um indicador de tempo restante.

4.3.2.6. Grupo F: Participação do usuário

- F1 – Múltiplas formas de participação do usuário

  A seleção de uma forma de participação do usuário deve considerar o tipo de aplicação, pois várias formas são possíveis, cada uma com seus prós e contras. As formas disponíveis nessa linguagem de *design patterns* são tratados nos padrões F2, F3, F4 e F5.

- F2 – Votação e questões de múltipla-escolha

  Esse padrão de projeto de interface destaca uma das formas de se ter participação do usuário numa aplicação interativa. Nesse caso, para uma questão, as respostas são baseadas no padrão de múltipla-escolha, cabendo ao usuário selecionar a(s) sua(s) resposta(s). Esse formato é adequado para diversos tipos de aplicação, como pesquisa de opinião, votação, testes e até em uma competição com *quiz*, com a participação de diversos usuários. Alguns aspectos devem ser observados, como o número permitido de respostas, a forma de apresentação das respostas (texto, figuras etc.), a possibilidade de se destacar ou

não uma resposta selecionada, se é permitido ou não cancelar uma resposta selecionada e até indicar a resposta correta posteriormente.

- F3 – Alocação de itens

  Essa forma de participação de usuário é uma analogia ao exercício de se relacionar duas colunas. No caso de uma aplicação interativa, existem dois conjuntos. Para um dos conjuntos, cada item possui uma tecla associada, como uma tecla de número ou cor. Com isso, deve-se selecionar um dos itens do segundo conjunto, por exemplo, por meio das teclas de setas. Ao se selecionar um dos itens do segundo conjunto, o usuário deve pressionar uma das outras teclas, selecionando o item correspondente do primeiro conjunto. Nesse cenário, aspectos como o número de itens por conjunto, a forma de apresentação dos itens, as teclas do controle remoto destinadas a cada item e a forma de se destacar os itens selecionados, e até mesmo como se destacar uma associação correta entre itens devem ser considerados.

- F4 – Completar texto

  Essa forma de participação também é um exemplo de exercício de aprendizado, onde o usuário deve completar um texto onde algumas palavras estão faltando. Como a digitação de caracteres pelo controle remoto é complicada, deve ser evitada, mostrando-se algumas opções de resposta para cada lacuna no texto. Com isso, algumas características devem ser planejadas, como a forma de apresentação das opções de resposta, o número dessas opções, a seleção de uma lacuna, a possibilidade de se cancelar uma opção selecionada, entre outras.

- F5 – Aprovação para conectividade

  Esse padrão trata da questão de se autorizar conectividade em termos de canal de retorno, quando for necessário seu uso por uma aplicação. Muitas vezes a

aplicação só irá exigir tal conexão, se ainda não estabelecida, quando chegar a um ponto necessário. Tal forma de aprovação pode ser implementada através de uma caixa de diálogo ou em uma página, avisando ao usuário sobre a conexão, alertando para possíveis cobranças financeiras e provendo as opções para aceitar ou não.

4.3.2.7. Grupo G: Entrada de texto

- G1 – Múltiplas forma de entrada de texto

  Esse modelo irá tratar da escolha da forma mais adequada de entrada de texto a ser usada em uma aplicação de TV Digital interativa. São propostas três formas de entrada de texto. Uma delas diz respeito a um teclado QWERTY físico, incluído em conjunto com alguns modelos de TV, porém como não se trata de uma oferta comum, outras formas são necessárias. As demais formas são tratadas nos padrões G2 e G3.

- G2 – Teclado QWERTY ou alfabético em tela

  Um teclado na tela exige dois elementos na interface: o teclado a ser exibido na tela propriamente dito e uma caixa de texto ou campo de formulário, onde o texto a ser digitado será exibido. Aspectos como o formato do teclado, QWERTY, familiar para usuários de computadores, ou em ordem alfabética, familiar a todos os tipos de usuários, mas não muito comum para digitação de texto, precisam ser levados em consideração. Outras questões também devem ser planejadas com cuidado, como a apresentação do teclado que deve ser em uma única página, exigindo bastante espaço da tela, a forma de seleção de um caractere, a indicação gráfica de que um caractere foi selecionado, a posição

atual do cursor no texto e a forma de se corrigir caracteres selecionados por engano.

- G3 – Teclado de telefone móvel

  Uma forma alternativa de entrada de texto diz respeito a se usar o modelo de entrada de texto de teclados físicos de telefones móveis, onde as letras são impressas nas teclas de números. Esse modelo pode ser familiar aos usuários desses tipos de telefones, e é adaptável ao teclado numérico do controle remoto de uma TV. Deve-se escolher também entre exibir ou não na tela um teclado nesse formato, de modo a guiar o usuário na sua seleção de caracteres.

4.3.2.8. Grupo H: Ajuda

- H1 – Instrução em tela

  Essa forma de ajuda é voltada para a exibição de sentenças curtas em tela. Costumam ser utilizadas em todas as páginas de uma aplicação de forma automática, indicando as opções mais adequadas a serem selecionadas. As instruções se diferem dos indicadores de navegação, pois estes últimos são representados por ícones acompanhados por apenas uma palavra, enquanto que as instruções são frases completas referindo-se a uma tarefa específica. Aspectos como a forma de escrita (que deve ser simples e precisa), e a posição do texto devem ser consideradas.

- H2 – Seção de ajuda

  Esse elemento trata de um componente de aplicação dedicado a prover ajuda em termos de conteúdo, com instruções acerca da navegação, estrutura e funções da aplicação como um todo. Algumas características desse componente devem ser pensadas, como a forma de se iniciar a ajuda, se esta deve ser passiva,

aguardando a ação do usuário para iniciar, ou ativa, rastreando as ações do usuário para saber quando deve iniciar ou não. O leiaute da página dessa seção, o conteúdo e a navegação nesse componente também devem ser cuidadosamente planejados.

4.3.2.9. Grupo I: Acessibilidade e personalização

- I1 – Acessibilidade

  Em termos de acessibilidade, diversos itens devem ser considerados, como: tamanho da fonte, legendas, intérprete de linguagem de sinais, idioma utilizado, *feedback* em formato de áudio, transparência do plano de fundo, uso de cores, entre outros. Muitas vezes é interessante permitir ao usuário configurar a acessibilidade, alterando alguns dos itens listados anteriormente.

- I2 – Personalização

  A personalização dá maior flexibilidade à aplicação, permitindo ao usuário moldar a aplicação de acordo com suas preferências. Existe também a possibilidade de implementar uma personalização passiva, requisitando que o usuário faça suas escolhas, ou ativa, observando a forma de uso da aplicação para alterar algumas configurações.

4.3.2.10. Grupo J: Grupos de usuários específicos

- J1 – Crianças

  Esse padrão de projeto trata dos problemas em torno do desenvolvimento de aplicações voltadas para crianças. Com isso, deve-se pensar em vários aspectos, como dificuldade em se usar um controle remoto tradicional ou até limitação de leitura para compreender as instruções em tela. É indicado o uso de teclas de

números, setas e cores para se navegar no menu (B2) e opção de vídeo multi-tela (B3). Em componentes de questões de múltipla-escolha (F2), é desejável o uso de elementos gráficos, como animações e figuras nas respostas.

Com os padrões de projeto apresentados, diversos componentes comuns em aplicações interativas de TV Digital passam a ter um guia de desenvolvimento voltado para a construção de suas interfaces. Esse tipo de orientação é de grande importância, pois leva em consideração aspectos específicos de TV Digital interativa, que diferem em vários níveis de outros cenários.

Os padrões de projeto de interface apresentados servem como apoio para o desenvolvimento de novas aplicações interativas em TV Digital e como estímulo para o estudo e utilização de técnicas de reutilização nessa área. Nesse contexto, uma biblioteca de código implementada para o padrão brasileiro de TV Digital é apresentada no Capítulo 5, de forma a apoiar o desenvolvimento de soluções orientadas aos padrões de projeto de interface interativa. A evolução dessa biblioteca para um *pattern* é uma opção a ser considerada e também será discutida, respeitando os critérios apresentados neste capítulo.

5. NCLForms - Apresentação e Contexto de Uso

Utilizando os conceitos de *design patterns*, bibliotecas de código e elementos de interface gráfica, juntamente com as características próprias das aplicações interativas para TV Digital, mais propriamente dentro do modelo brasileiro, foi desenvolvida nesse trabalho uma *Application Programming Interface* (API) para o desenvolvimento de *Graphical User Interfaces* (GUIs), cujo nome é NCLForms. Uma apresentação dessa API, bem como dos elementos que a compõem, é realizada na Seção 5.1. A Seção 5.2 fornece um guia de uso da API, com a apresentação de um exemplo.

5.1. A API

A implementação da API NCLForms visa fornecer ao desenvolvedor de aplicações para o *middleware* Ginga, em sua versão para a linguagem NCL, um conjunto de *widgets* para a adição de formulários em aplicações interativas.

Apesar de ser possível inserir formulários em aplicações NCL por meio de documentos HTML, não é possível uma customização tão profunda de tais elementos, a ponto de escolher aspectos relacionados à aparência dos elementos e, até mesmo, uma integração mais eficiente dos mesmos com os demais componentes da aplicação. Como a API foi desenvolvida em NCLua, sua implementação em documentos NCL ocorre de maneira bem simplificada, cabendo ao desenvolvedor cuidar apenas da parte de tratamento dos dados relacionados ao formulário, pois a parte de interface já se encontra estruturada pela API.

A Figura 4 exibe um protótipo de aplicação, apresentando um formulário interativo, fazendo uso da API para a construção de uma GUI.

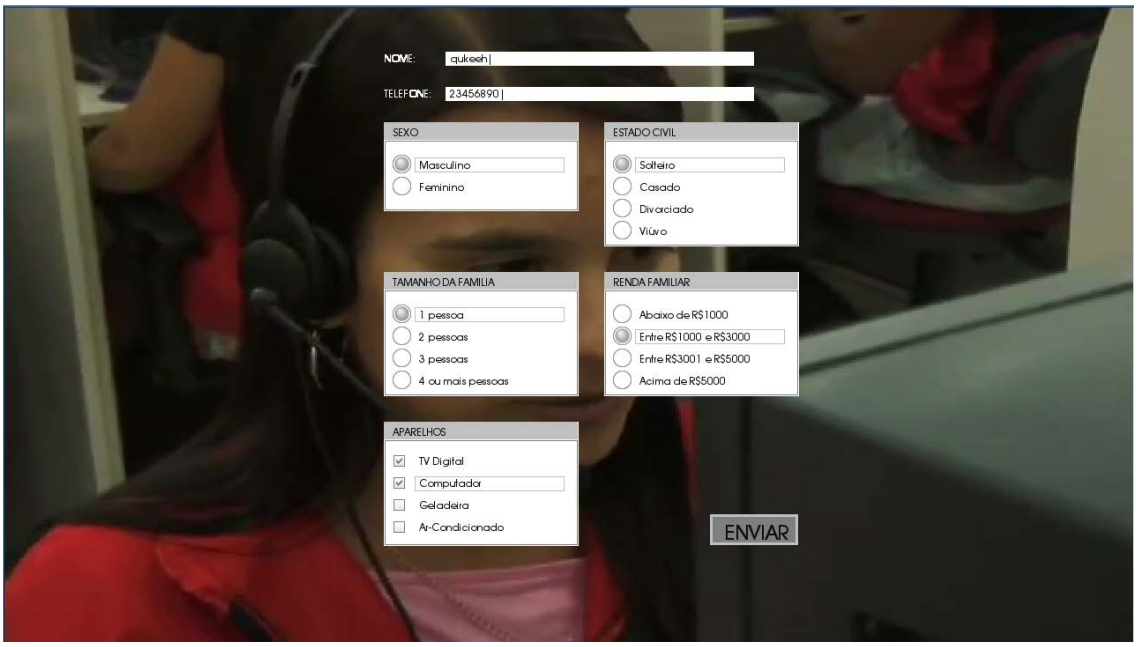

Figura 4. Aplicação com formulário implementado através da API NCLForms.

Durante o desenvolvimento da API foram utilizados como ferramentas a máquina virtual Ginga-NCL Virtual Set-top Box v.0.12.3 (PORTAL DO SOFTWARE PÚBLICO BRASILEIRO, 2012), para emular o *middleware* Ginga-NCL, o *software* de virtualização VMware Player v.3.1.4 (VMWARE PLAYER, 2012), para executar a máquina virtual, e o ambiente de desenvolvimento Eclipse (THE ECLIPSE FOUNDATION, 2012), com *plug-in* NCL Eclipse (NCL ECLIPSE, 2012) para validação de arquivos NCL e *plug-in* akdebugger (SOURCEFORGE, 2012) para edição de arquivos Lua.

As questões de reutilização e flexibilidade de customização dos elementos gráficos foram consideradas em todos os pontos do projeto. Uma análise mais detalhada desses aspectos é feita nas Seções 5.1.1, com a apresentação de elementos próprios da linguagem NCL e da API referente à possibilidade de reúso e 5.1.2, com a apresentação dos *widgets* e a descrição de seus parâmetros de uso.

5.1.1. Reutilização

Apesar de ser uma prática mais comum em linguagens imperativas, o reúso também é aplicado no paradigma declarativo (SOARES & SOARES NETO, 2009). Em geral, linguagens declarativas são projetadas voltadas para domínios específicos. Com isso, o reúso nesse modelo deve ser direcionado para o mesmo foco da linguagem.

Uma técnica recomendada para apoiar a reutilização em aplicações interativas de TV Digital é a adoção de padrões de interface. Com esses padrões, é possível reutilizar o posicionamento de elementos na cena, além do comportamento de objetos de mídia (SOARES & SOARES NETO, 2009). Com a linguagem NCL, isso é não apenas possível, como recomendado. Com a possibilidade de reutilizar um elemento `<region>` para o mesmo posicionamento de diferentes elementos, além de um elemento `<descriptor>`, designando um mesmo comportamento para diferentes objetos de mídia.

Outra prática amplamente aplicada em reúso diz respeito às bibliotecas de código, aspecto que está diretamente relacionado à utilização da API NCLForms. Sua implementação considerou que a mesma pudesse ser facilmente incorporada a um documento NCL, criando novas aplicações para TV Digital a partir da utilização de *widgets*. Uma descrição desses elementos é realizada a seguir.

5.1.2. *Widgets*

Em uma fase inicial do desenvolvimento da API, optou-se por criar cada elemento de forma individual, independente de um elemento formulário. Dessa forma, o usuário da API teria que importar cada elemento individualmente. No entanto, de forma a simplificar a importação e facilitar a implementação de uma navegabilidade mais

eficiente através dos elementos do formulário, alterou-se o projeto. Incorporando apenas um objeto NCLua ao código NCL, que seria um elemento formulário, o usuário da API pode decidir por customizar tal formulário com os *widgets* existentes no mesmo da forma como desejar.

Visando o desenvolvimento de um único elemento NCLForms, todos os *widgets* existentes nesse formulário foram implementados de forma uniforme e integrada. Cada *widget* conta com um conjunto de parâmetros que permite a sua customização. No entanto, os parâmetros escolhidos representam uma proposta inicial, e uma alteração dos mesmos, com a exclusão de alguns e a inclusão de outros, também é uma alternativa viável e pode ser adotada em uma revisão do projeto.

Alguns elementos possuem parâmetros comuns. De forma a ilustrar essa relação entre os elementos propostos, a Figura 5 apresenta um modelo conceitual na forma de um diagrama de classes de todos os *widgets*.

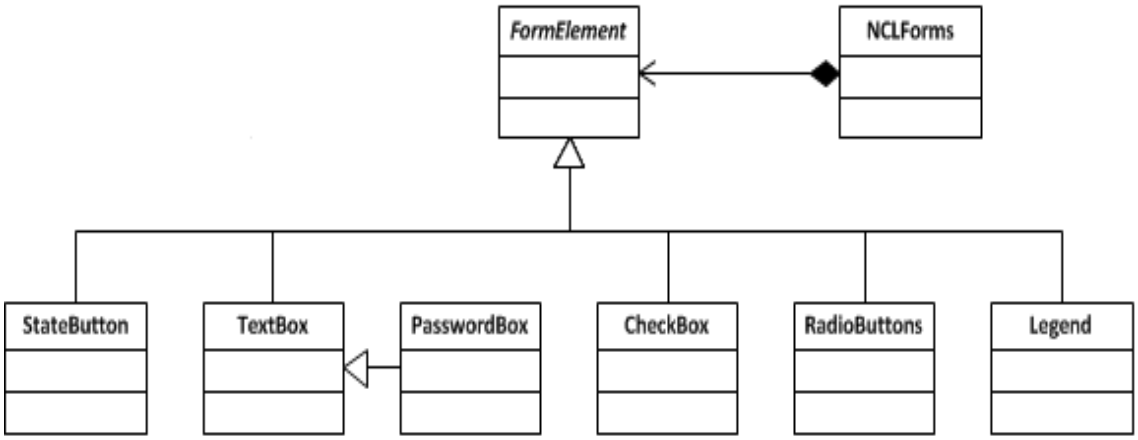

Figura 5. Modelo conceitual com todos os elementos propostos na API.

Com relação à interação do usuário com a aplicação, todos os *widgets*, com exceção da legenda, contam com os seguintes eventos em comum:

- **Ir para o próximo elemento**

Quando o usuário pressiona a tecla de seta para direita do controle remoto, o foco atual da aplicação passa para o próximo *widget* na ordem de foco definida pelo desenvolvedor da aplicação e informada na passagem de parâmetros para a API.

- **Ir para o elemento anterior**

  Quando o usuário pressiona a tecla de seta para esquerda do controle remoto, o foco atual da aplicação passa para o *widget* anterior na ordem de foco definida pelo desenvolvedor da aplicação e informada na passagem de parâmetros para a API.

A seguir, cada um dos *widgets* é apresentado, com sua descrição e seus respectivos parâmetros. Na Seção 5.2, são apresentados o elemento formulário e a forma como os parâmetros de todos os *widgets* internos ao mesmo são organizados, além dos eventos pertinentes a cada elemento, demonstrando a utilização da API.

5.1.2.1. Botão de estados (*StateButton*)

Apesar de ser possível implementar um botão de estados de forma simples, compondo imagens e descritores, foi decidido incluir um elemento *statebutton* na API, a fim de poder manter uma identidade visual única, afinal o botão também é construído com as primitivas de desenho do módulo *canvas* de NCLua, assim como os demais *widgets*. Além disso, com um elemento *statebutton* integrado ao formulário, a navegação já estaria estruturada e funções comuns a formulários, como um botão do tipo *submit*, para enviar os dados preenchidos, seriam facilmente incorporadas. A Figura 6 ilustra a representação gráfica dos estados (pressionado ou não) do elemento *statebutton*.

Figura 6. Representação gráfica dos estados do *widget statebutton*.

Alguns parâmetros desse elemento são comuns a outros *widgets*, como aspectos relacionados à dimensão ou cores. Tais parâmetros são listados a seguir:

- **Ordem de foco**

  Corresponde ao índice que informa à aplicação em que posição esse elemento se encontra na sequência de elementos do formulário. Parâmetro utilizado para fins de navegação e identificação do elemento com outro *widget*, a legenda.

- **Rótulo**

  Corresponde ao texto que será exibido sobre o botão, identificando sua função no contexto da aplicação.

- **X**

  Corresponde à coordenada x do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Y**

  Corresponde à coordenada y do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Largura**

  Corresponde à largura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget* botão.

- **Altura**

  Corresponde à altura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget* botão.

- **Cor de fundo**

  Corresponde à cor de fundo do elemento em seu estado inativo ("*Off*").

- **Tamanho da fonte**

  Corresponde ao tamanho da fonte do rótulo do elemento.

- **Cor da fonte**

  Corresponde à cor da fonte do rótulo do elemento.

  A seguir, é descrito o único evento relacionado a esse *widget*:

- **Pressionar botão**

  Quando o usuário pressiona a tecla OK do controle remoto, o estado do botão se altera, com transição de "*On*" para "*Off*" ou vice-versa, dependendo do estado atual. Cada estado do botão apresenta uma cor de fundo diferente.

5.1.2.2. Caixa de texto (*TextBox*)

Um dos elementos mais comuns em um formulário é a caixa de texto, que pode ser preenchida pelo usuário com caracteres alfanuméricos. O *widget textbox* representa esse elemento, permitindo ao desenvolvedor personalizar algumas de suas características, como as dimensões da caixa de texto, bem como o limite de caracteres que podem ser utilizados em seu preenchimento. Em termos de navegação, a entrada de texto se dá através do teclado numérico do controle remoto, mas o usuário da API pode alterar esse componente, estendendo a API para a exibição de um teclado virtual na tela. A Figura 7 apresenta a interface gráfica do *widget textbox*.

Figura 7. Representação gráfica do elemento *textbox*.

Alguns parâmetros desse *widget* são comuns a outros elementos, como aspectos relacionados à posição ou à dimensão. Os parâmetros são apresentados a seguir:

- **Ordem de foco**

  Corresponde ao índice que informa à aplicação em que posição esse elemento se encontra na sequência de elementos do formulário. Parâmetro utilizado para fins de navegação e identificação do elemento com outro *widget*, a legenda.

- **Rótulo**

  Corresponde ao texto que será exibido ao lado da caixa de texto, identificando qual dado deve ser preenchido.

- **X**

  Corresponde à coordenada x do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Y**

  Corresponde à coordenada y do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **dX**

  Corresponde à distância horizontal entre o rótulo e a caixa de texto.

- **dY**

  Corresponde à distância vertical entre o rótulo e a caixa de texto.

- **Largura**

  Corresponde à largura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget* caixa de texto.

- **Altura**

  Corresponde à altura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget* caixa de texto.

- **Cor de fundo**

  Corresponde à cor de fundo da caixa de texto.

- **Cor da borda**

  Corresponde à cor da borda da caixa de texto.

- **Tamanho da fonte**

  Corresponde ao tamanho da fonte dos caracteres a serem digitados no elemento.

- **Cor da fonte**

  Corresponde à cor da fonte dos caracteres a serem digitados no elemento.

- **Comprimento máximo**

  Corresponde ao número máximo de caracteres que podem ser digitados no *widget*.

  A seguir, são descritos os eventos relacionados a essa primitiva gráfica:

- **Digitar caractere**

  Quando o usuário pressiona uma das teclas numéricas do controle remoto, o caractere correspondente é exibido na caixa de texto.

- **Apagar caractere**

  Quando o usuário pressiona a tecla de cor vermelha do controle remoto, o caractere correspondente à posição atual do cursor é apagado na caixa de texto.

- **Ativar/Desativar caixa alta**

  Quando o usuário pressiona a tecla de seta para cima do controle remoto, é ativada a caixa alta para digitação de caracteres se ela já não estiver ativada, ou é desativada, caso ela esteja ativada.

- **Confirmar dados**

Quando o usuário pressiona a tecla OK do controle remoto, os caracteres digitados são guardados e o cursor da caixa de texto deixa de ser exibido.

5.1.2.3. Caixa de senha (*PasswordBox*)

Outro elemento comum em um formulário é a caixa de senha, onde o usuário pode preenchê-la, sem que os caracteres digitados sejam exibidos. Normalmente, são *widgets* aplicados em formulários de autenticação de uma conta. Assim como o elemento caixa de texto, é possível personalizar algumas de suas características, como as dimensões e o limite de caracteres que podem ser utilizados em seu preenchimento. A Figura 8 apresenta a interface gráfica do *widget passwordbox*.

Figura 8. Representação gráfica do elemento *passwordbox*.

Os parâmetros desse *widget* são os mesmos do elemento caixa de texto. Os parâmetros são apresentados a seguir:

- **Ordem de foco**

  Corresponde ao índice que informa à aplicação em que posição esse elemento se encontra na sequência de elementos do formulário. Parâmetro utilizado para fins de navegação e identificação do elemento com outro *widget*, a legenda.

- **Rótulo**

  Corresponde ao texto que será exibido ao lado da caixa de senha, identificando qual dado deve ser preenchido.

- **X**

Corresponde à coordenada x do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Y**

  Corresponde à coordenada y do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **dX**

  Corresponde à distância horizontal entre o rótulo e a caixa de senha.

- **dY**

  Corresponde à distância vertical entre o rótulo e a caixa de senha.

- **Largura**

  Corresponde à largura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget* caixa de senha.

- **Altura**

  Corresponde à altura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget* caixa de senha.

- **Cor de fundo**

  Corresponde à cor de fundo da caixa de senha.

- **Cor da borda**

  Corresponde à cor da borda da caixa de senha.

- **Tamanho da fonte**

  Corresponde ao tamanho da fonte dos caracteres a serem digitados no elemento.

- **Cor da fonte**

  Corresponde à cor da fonte dos caracteres a serem digitados no elemento.

- **Comprimento máximo**

Corresponde ao número máximo de caracteres que podem ser digitados no *widget*.

A seguir, são descritos os eventos relacionados a esse elemento:

- **Digitar caractere**

  Quando o usuário pressiona uma das teclas numéricas do controle remoto, o caractere "*" é exibido na caixa de senha, de forma a não revelar o conteúdo digitado.

- **Apagar caractere**

  Quando o usuário pressiona a tecla de cor vermelha do controle remoto, o caractere correspondente à posição atual do cursor é apagado na caixa de senha.

- **Ativar/Desativar caixa alta**

  Quando o usuário pressiona a tecla de seta para cima do controle remoto, é ativada a caixa alta para digitação de caracteres se ela já não estiver ativada, ou é desativada, caso ela esteja ativada.

- **Confirmar dados**

  Quando o usuário pressiona a tecla OK do controle remoto, os caracteres digitados são guardados e o cursor da caixa de senha deixa de ser exibido.

5.1.2.4. Caixa de seleção (*CheckBox*)

Um elemento que foi adicionado à API é a caixa de seleção, também conhecida por *checkbox*, onde é possível selecionar um ou mais itens de uma lista de opções. Tal *widget* foi implementado utilizando primitivas de desenho do módulo *canvas* de NCLua, além de imagens para compor a caixa de seleção propriamente dita. A Figura 9 apresenta a interface gráfica do elemento *checkbox*.

Figura 9. Representação gráfica do *widget checkbox*.

Os parâmetros são apresentados a seguir:

- **Ordem de foco**

  Corresponde ao índice que informa à aplicação em que posição esse elemento se encontra na sequência de elementos do formulário. Parâmetro utilizado para fins de navegação e identificação do elemento com outro *widget*, a legenda.

- **Número de opções**

  Corresponde ao número de opções que serão exibidas no *widget*, cada uma correspondente a uma caixa de seleção.

- **Rótulo**

  Corresponde ao texto que será exibido acima do *widget*, identificando o propósito do elemento no formulário.

- **X**

  Corresponde à coordenada x do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Y**

  Corresponde à coordenada y do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Largura**

Corresponde à largura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget*.

- **Altura**

  Corresponde à altura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget*.

- **Cor de fundo**

  Corresponde à cor de fundo do elemento.

- **Tamanho da fonte**

  Corresponde ao tamanho da fonte do rótulo de cada opção.

- **Cor da fonte**

  Corresponde à cor da fonte do rótulo de cada opção no elemento.

- **Rótulos das opções**

  Corresponde aos nomes das opções para cada caixa de seleção. Tal parâmetro deve ser informado com todos os rótulos de uma vez, separados por ponto e vírgula.

A seguir, são descritos os eventos relacionados a esse *widget*:

- **Selecionar opção**

  Quando o usuário pressiona a tecla OK do controle remoto sobre uma das opções listadas, se a caixa de seleção da opção com foco atual não estiver assinalada, ela é selecionada, marcando sua escolha.

- **Anular a seleção de opção**

  Quando o usuário pressiona a tecla OK do controle remoto sobre uma das opções listadas, se a caixa de seleção da opção com foco atual estiver assinalada, ela deixa de ser selecionada, marcando a desistência em sua escolha.

- **Ir para a opção abaixo**

  Quando o usuário pressiona a tecla de seta para baixo do controle remoto, o quadro de item atual é deslocado para a opção abaixo, marcando o novo item com foco atual.

- **Ir para a opção acima**

  Quando o usuário pressiona a tecla de seta para cima do controle remoto, o quadro de item atual é deslocado para a opção acima, marcando o novo item com foco atual.

5.1.2.5. Botão de opção (*Radio buttons*)

Um elemento cuja interface é semelhante às caixas de seleção, porém cujo comportamento é diferente, sendo possível selecionar apenas um item das opções listadas. Tal *widget* foi implementado utilizando primitivas de desenho do módulo *canvas* de NCLua, além de imagens para compor os botões de opção. A Figura 10 apresenta a interface gráfica do elemento *radio buttons*.

Figura 10. Representação gráfica do *widget radio buttons*.

Os parâmetros são quase os mesmos do *widget checkbox*. Tais parâmetros são apresentados a seguir:

- **Ordem de foco**

Corresponde ao índice que informa à aplicação em que posição esse elemento se encontra na sequência de elementos do formulário. Parâmetro utilizado para fins de navegação e identificação do elemento com outro *widget*, a legenda.

- **Número de opções**

  Corresponde ao número de opções que serão exibidas no *widget*, cada uma correspondente a um botão.

- **Rótulo**

  Corresponde ao texto que será exibido acima do *widget*, identificando o propósito do elemento no formulário.

- **X**

  Corresponde à coordenada x do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Y**

  Corresponde à coordenada y do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Largura**

  Corresponde à largura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget*.

- **Altura**

  Corresponde à altura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget*.

- **Cor de fundo**

  Corresponde à cor de fundo do elemento.

- **Tamanho da fonte**

  Corresponde ao tamanho da fonte do rótulo de cada opção.

- **Cor da fonte**

  Corresponde à cor da fonte do rótulo de cada opção no elemento.

- **Rótulos das opções**

  Corresponde aos nomes das opções. Tal parâmetro deve ser informado com todos os rótulos de uma vez, separados por ponto e vírgula.

A seguir, são descritos os eventos relacionados a essa primitiva gráfica:

- **Selecionar opção**

  Quando o usuário pressiona a tecla OK do controle remoto sobre uma das opções listadas, se o botão da opção com foco atual não estiver assinalado, ele é selecionado, marcando sua escolha e anulando a seleção da opção anteriormente assinalada, se houver. Se a opção selecionada já estiver assinalada, nada acontece.

- **Ir para a opção abaixo**

  Quando o usuário pressiona a tecla de seta para baixo do controle remoto, o quadro de item atual é deslocado para a opção abaixo, marcando o novo item com foco atual.

- **Ir para a opção acima**

  Quando o usuário pressiona a tecla de seta para cima do controle remoto, o quadro de item atual é deslocado para a opção acima, marcando o novo item com foco atual.

5.1.2.6. Caixa de listagem (*List box*)

Um elemento que exibe uma lista de opções, permitindo ao usuário selecionar apenas uma ou mais delas. Por possuir uma barra de rolagem vertical, é um *widget*

adequado para a seleção de itens em listas com um número grande de opções. Devido a limitações com relação a controle, essa primitiva gráfica foi implementada possibilitando ao usuário selecionar apenas uma opção, umas vez que seria complexo realizar múltiplas escolhas nesse elemento através do controle remoto. A Figura 11 ilustra a interface gráfica do elemento *list box*.

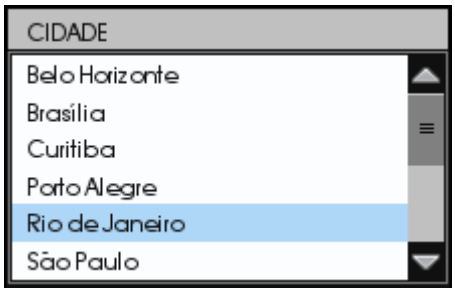

Figura 11. Representação gráfica do *widget list box*.

Os parâmetros do elemento caixa de listagem são apresentados a seguir:

- **Ordem de foco**

  Corresponde ao índice que informa à aplicação em que posição esse elemento se encontra na sequência de elementos do formulário. Parâmetro utilizado para fins de navegação e identificação do elemento com outro *widget*, a legenda.

- **Número de opções**

  Corresponde ao número total de opções existentes no *widget*.

- **Rótulo**

  Corresponde ao texto que será exibido acima do *widget*, identificando o propósito do elemento no formulário.

- **X**

  Corresponde à coordenada x do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Y**

  Corresponde à coordenada y do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Largura**

  Corresponde à largura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget*.

- **Altura**

  Corresponde à altura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget*.

- **Cor de fundo**

  Corresponde à cor de fundo do elemento.

- **Número de opções visíveis**

  Corresponde ao número de opções que serão exibidas simultaneamente. Este valor pode ser menor ou igual ao parâmetro “número de opções”.

- **Cor da fonte**

  Corresponde à cor da fonte do rótulo de cada opção no elemento.

- **Rótulos das opções**

  Corresponde aos nomes das opções. Tal parâmetro deve ser informado com todos os rótulos de uma vez, separados por ponto e vírgula.

A seguir, são descritos os eventos relacionados a essa primitiva gráfica:

- **Selecionar opção**

  Quando o usuário pressiona a tecla OK do controle remoto sobre uma das opções listadas, se a linha da opção com foco atual não estiver em destaque, ela é selecionada, marcando sua escolha e anulando a seleção da opção

anteriormente em destaque, se houver. Se a opção selecionada já estiver destacada, nada acontece, com sua linha permanecendo selecionada.

- **Ir para a opção abaixo**

  Quando o usuário pressiona a tecla de seta para baixo do controle remoto, a linha do item atual deixa de ser destacada, passando a destacar a linha da opção abaixo, marcando o novo item com foco atual.

- **Ir para a opção acima**

  Quando o usuário pressiona a tecla de seta para cima do controle remoto, a linha do item atual deixa de ser destacada, passando a destacar a linha da opção acima, marcando o novo item com foco atual.

5.1.2.7. Legenda

Um elemento que auxilia na navegação pelos demais *widgets*. A legenda é um elemento único associado a cada um dos demais *widgets*, exibindo, por meio de ícones e textos curtos, quais teclas do controle remoto o usuário da aplicação deve pressionar para navegar pelo elemento com foco atual. Essa associação entre uma legenda e um *widget* é possibilitada pelo parâmetro "ordem de foco" de cada um dos demais elementos do formulário. Tal *widget* foi implementado utilizando primitivas de desenho do módulo *canvas* de NCLua, além de imagens referentes à representação de cada uma das possíveis teclas do controle remoto. A Figura 12 ilustra a interface gráfica de *widgets* legenda.

Figura 12. Representações gráficas do elemento legenda.

Os parâmetros do *widget* legenda são apresentados a seguir:

- **Ordem de foco**

  Corresponde ao índice que associa a exibição desse elemento quando o *widget* com o mesmo índice estiver com o foco atual durante a execução da aplicação com o formulário.

- **Número de opções**

  Corresponde ao número total de dicas de navegação existentes no *widget*, sendo um ícone para cada dica de tecla a ser exibida.

- **X**

  Corresponde à coordenada x do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Y**

  Corresponde à coordenada y do eixo cartesiano, indicando em que ponto da tela o *widget* deve ser desenhado.

- **Largura**

  Corresponde à largura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget*.

- **Altura**

  Corresponde à altura em *pixels* do elemento. Importante para indicar a dimensão final da representação gráfica do *widget*.

- **Cor de fundo**

  Corresponde à cor de fundo do elemento.

- **Tamanho da fonte**

  Corresponde ao tamanho da fonte do texto sendo exibido em cada dica de navegação.

- **Cor da fonte**

  Corresponde à cor da fonte do texto sendo exibido em cada dica de navegação.

- **Teclas e texto das dicas**

  Corresponde à informação de quais teclas devem ser exibidas como dicas no elemento atual. À cada tecla deve ser informado um texto que será exibido como rótulo abaixo do ícone da tecla. Nesse parâmetro, todos os pares (rótulo, tecla) devem ser informados em sequência, separados apenas por ponto e vírgula e sem parênteses entre os pares.

O *widget* legenda não apresenta nenhum evento com interação direta do usuário da aplicação.

5.2. Utilização

Após identificar, na Seção 5.1.2, todos os parâmetros possíveis para cada um dos *widgets* desenvolvidos, é importante destacar a forma correta de uso de tais elementos. Também é apresentada, nessa seção, a forma adequada de incorporar uma GUI desenvolvida com a API NCLForms em um documento NCL, sendo possível inserir um formulário em uma aplicação interativa de TV Digital.

A Figura 13 exibe um elemento `<media>` fazendo referência a um *script* Lua. Esse trecho de código informa à aplicação NCL qual o objeto NCLua que deverá ser executado quando for solicitado. Deve-se destacar a importância de se definir um elemento `<property>` denominado *values* para o objeto de mídia. Tal elemento será importante na passagem de parâmetros em seguida. Um elemento `<property>` denominado *result* também deve ser incluído no objeto de mídia. Esse elemento poderá

armazenar, posteriormente, os resultados das seleções feitas em cada um dos *widgets* do formulário. O *script* NCLForms.lua é o arquivo principal da API, sendo responsável por exibir os elementos do formulário na aplicação.

```
<media id="forms" src="NCLForms.lua" descriptor="dsForms">
      <property name="values"/>
      <property name="result"/>
</media>
```

Figura 13. Incorporando *script* Lua em um documento NCL.

Após inserir o elemento `<media>` com a indicação do arquivo Lua a ser incorporado, deve-se informar quais os parâmetros de formulário para a API, a fim de saber quais os *widgets* que serão exibidos, bem como a forma deles. Isso é realizado através do elemento `<link>`, atribuindo os parâmetros à interface de nome *values*. A Figura 14 ilustra esse passo.

```
<link xconnector="onBeginSetN">
      <bind role="onBegin" component="forms"/>
      <bind role="set" component="forms" interface="values">
            <bindParam name="var"
      value="
textbox(1;NOME:;430;50;70;0;350;16;white;black;12;black;10),
passwordbox(2;SENHA:;430;90;70;0;350;16;white;black;12;black;10),
radiobuttons(3;2;SEXO;430;150;220;80;white;12;black;Masculino;Femin
ino)”
      </bind>
</link>
```

Figura 14. Passagem de parâmetros para o *script* NCLua.

Após a passagem de parâmetros no documento NCL, a API irá interpretar esses dados e desenhar na tela os componentes indicados na interface *values*. A Figura 15 apresenta a função *nclHandler*, presente no arquivo NCLForms.lua, contido no pacote que compõe a API, que foi registrada para tratar eventos de atribuição. Tal função é responsável por processar tanto eventos voltados para a digitação de texto após a exibição dos componentes na tela, eventos de nome “*text*”, como evento de atribuição

de valores dos parâmetros, eventos de nome “*values*”, reconhecendo quais primitivas gráficas foram informadas.

```
local function nclHandler (evt)
      if evt.class ~= 'ncl' then return end

      if evt.type == 'attribution' then
            if evt.name == 'text' then
                  setText(evt.value, true)
            elseif evt.type == 'attribution' then
                  if evt.name == 'values' then
                        if evt.action == 'start' then
                              _, stbN = string.gsub(evt.value,
'statebutton', "")
                              _, ckbN = string.gsub(evt.value,
'checkbox', "")
                              _, rdbN = string.gsub(evt.value,
'radiobuttons', "")
                              _, ipbN = string.gsub(evt.value,
'textbox', "")
                              _, pwbN = string.gsub(evt.value,
'passwordbox', "")
                              _, ltbN = string.gsub(evt.value,
'listbox', "")
                              _, lgdN = string.gsub(evt.value,
'legend', "")

                              local stbCount=stbN
                              local ckbCount=ckbN
                              local rdbCount=rdbN
                              local ipbCount=ipbN
                              local pwbCount=pwbN
                              local ltbCount=ltbN
                              local lgdCount=lgdN

                              while(ltbCount>0) do
                                    formParser(evt.value, 'listbox')
                                    ltbCount=ltbCount-1
                              end
                              while(stbCount>0) do
                                    formParser(evt.value,
'statebutton')
                                    stbCount=stbCount-1
                              end
                              while(ckbCount>0) do
                                    formParser(evt.value,
'checkbox')
                                    ckbCount=ckbCount-1
                              end
                              while(rdbCount>0) do
                                    formParser(evt.value,
'radiobuttons')
                                    rdbCount=rdbCount-1
                              end
```

Figura 15. Recepção de parâmetros pela GUI com a função *nclHandler*.

```
                                while(rdbCount>0) do
                                      formParser(evt.value,
'radiobuttons')
                                      rdbCount=rdbCount-1
                                end
                                while(ipbCount>0) do
                                      formParser(evt.value, 'textbox')
                                      ipbCount=ipbCount-1
                                end
                                while(pwbCount>0) do
                                      formParser(evt.value,
'passwordbox')
                                      pwbCount=pwbCount-1
                                end
                                while(lgdCount>0) do
                                      formParser(evt.value, 'legend')
                                      lgdCount=lgdCount-1
                                end

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

        redraw()
        return true
end
event.register(nclHandler)
```

Figura 15 (continuação). Recepção de parâmetros pela API com a função *nclHandler*.

A função *nclHandler* faz chamadas à função *formParser*, que recebe como parâmetros o valor da *string* com todos os parâmetros e uma *string* indicando qual seu *widget* correspondente. A função *formParser* irá criar efetivamente as metatabelas[2] para cada elemento, separando os parâmetros específicos de cada elemento. A Figura 16 apresenta um trecho do código da função *formParser*.

[2] Lua não é uma linguagem que segue o paradigma de orientação a objetos, mas apresenta flexibilidade para simular a criação de classes e métodos. Isso é realizado por meio de metatabelas para representar as classes e metamétodos como os métodos referentes à classe.

```
local function formParser(values, form)

      {…}

      lastSep=string.find(values,')')
      beginSep=string.find(values,sep,formSep,lastSep)
      endSep=string.find(values,")",beginSep+5)
      params=string.sub(values, beginSep+1, endSep-1)

      action = {
            ["statebutton"] = function (x) stb =
stateButton.create(x) table.insert(stbList, stb) end,
            ["textbox"] = function (x) ipb = inputBox.create(x)
table.insert(ipbList, ipb) end,
            ["passwordbox"] = function (x) pwb =
passwordBox.create(x) table.insert(pwbList, pwb) end,
            ["checkbox"] = function (x) ckb = checkBox.create(x)
table.insert(ckbList, ckb) end,
            ["radiobuttons"] = function (x) rdb =
radioButtons.create(x) table.insert(rdbList, rdb) end,
            ["listbox"] = function (x) ltb = listBox.create(x)
table.insert(ltbList, ltb) end,
            ["legend"] = function (x) lgd = legend.create(x)
table.insert(lgdList, lgd) end,
      }
      return action[form](params)
end
```

Figura 16. Criando as estruturas dos elementos na função *formParser*.

Após serem criadas as metatabelas de cada elemento, os métodos de cada estrutura podem ser executados pelos componentes de controle e desenho da API, a fim de exibir o resultado na tela. A Figura 17 exibe o código do arquivo stateButton.lua, com a estrutura da metatabela do elemento botão de estado (Seção 5.1.2.1), assim como todos os seus métodos. Cada elemento tem sua própria estrutura, com métodos próprios que farão uso dos parâmetros informados ainda no documento NCL e interpretados ao longo do processo aqui descrito.

```
FRAMECOLOR = 'silver'
stateButton = {}
stateButton.__index = stateButton

function stateButton.create(params)
   local stButton = {}
```

Figura 17. Estrutura da metatabela do elemento botão de estados.

```
   setmetatable(stButton,stateButton)
   stButton.params = params
   stButton.isOn = false
   stButton.focus = false
   stButton.result = ''
   return stButton
end

function stateButton:__tostring()
  return self.params
end

function stateButton:getIsOn()
      return self.isOn
end

function stateButton:setIsOn(o)
      self.isOn = o
end

function stateButton:getFocus()
      return self.focus
end

function stateButton:setFocus(o)
      self.focus = o
end

function stateButton:getFocusOrder()
      return tonumber(parsers.parser(self.params, 1))
end

function stateButton:getText()
      return parsers.parser(self.params, 2)
end

function stateButton:getX1()
      return parsers.parser(self.params, 3)
end

function stateButton:getY1()
      return parsers.parser(self.params, 4)
end

function stateButton:getWidth()
      return parsers.parser(self.params, 5)
end

function stateButton:getHeight()
      return parsers.parser(self.params, 6)
end

function stateButton:getBkgColor()
      return parsers.parser(self.params, 7)
end
```

Figura 17 (continuação). Estrutura da metatabela do elemento botão de estados.

```
function stateButton:getFontSize()
      return parsers.parser(self.params, 8)
end

function stateButton:getFontColor()
      return parsers.parser(self.params, 9)
end
function stateButton:getResult()
      return self.result
end

function stateButton:setResult(res)
      self.result = '('..self:getFocusOrder()..res..')'
end

function stateButton:redraw()
      if (self:getIsOn()==true) then
            canvas:attrColor('gray')
            canvas:drawRect('fill',
self:getX1(),self:getY1(),self:getWidth(),self:getHeight())
            canvas:attrColor(FRAMECOLOR)
            canvas:drawRect('frame',
self:getX1(),self:getY1(),self:getWidth(),self:getHeight())
            canvas:drawRect('frame',
self:getX1()+1,self:getY1()+1,self:getWidth()-2,self:getHeight()-2)
            canvas:drawRect('frame',
self:getX1()+2,self:getY1()+2,self:getWidth()-4,self:getHeight()-4)
            canvas:attrFont('vera', self:getFontSize())
            canvas:attrColor(self:getFontColor())

      canvas:drawText(self:getX1()+self:getWidth()/6,self:getY1()+s
elf:getHeight()/7, self:getText())
      self:setResult(',On')
      else
            canvas:attrColor('silver')
            canvas:drawRect('fill',
self:getX1(),self:getY1(),self:getWidth(),self:getHeight())
            canvas:attrColor(FRAMECOLOR)
            canvas:drawRect('frame',
self:getX1(),self:getY1(),self:getWidth(),self:getHeight())
            canvas:drawRect('frame',
self:getX1()+1,self:getY1()+1,self:getWidth()-2,self:getHeight()-2)
            canvas:drawRect('frame',
self:getX1()+2,self:getY1()+2,self:getWidth()-4,self:getHeight()-4)
            canvas:attrFont('vera', self:getFontSize())
            canvas:attrColor(self:getFontColor())

      canvas:drawText(self:getX1()+self:getWidth()/6,self:getY1()+s
elf:getHeight()/7, self:getText())
      self:setResult(',Off')
      end
end
```

Figura 17 (continuação). Estrutura da metatabela do elemento botão de estados.

Após apresentar a parte principal do processo de criação de *widgets*, desde os parâmetros sendo informados no documento NCL até a estrutura de cada elemento, é

importante destacar a função *redraw* no arquivo NCLForms.lua, responsável por executar o método *redraw* da metatabela de cada elemento. A Figura 18 apresenta um trecho do código dessa função.

```
function redraw ()
	local stbCount=stbN
	local ckbCount=ckbN
	local rdbCount=rdbN
	local ipbCount=ipbN
	local pwbCount=pwbN
	local ltbCount=ltbN
	local lgdCount=lgdN

	result.value = '('

	while(stbCount>0) do
		if(stbList[stbN-stbCount+1]~=nil) then
			if(stbList[stbN-stbCount+1]:getFocus() or
firstRedraw) then
				stbList[stbN-stbCount+1]:redraw()
			end
			if(result.value=='(') then
				result.value = result.value..stbList[stbN-
stbCount+1]:getResult()
			else
				result.value =
result.value..','..stbList[stbN-stbCount+1]:getResult()
			end
		end
		stbCount=stbCount-1
	end

	while(ipbCount>0) do
		if(ipbList[ipbN-ipbCount+1]~=nil) then
			if(ipbList[ipbN-ipbCount+1]:getFocus() or
firstRedraw) then
				ipbList[ipbN-ipbCount+1]:redraw()
			end
			if(result.value=='(') then
				result.value = result.value..ipbList[ipbN-
ipbCount+1]:getResult()
			else
				result.value =
result.value..','..ipbList[ipbN-ipbCount+1]:getResult()
			end
		end
		ipbCount=ipbCount-1
	end
```

Figura 18. Função *redraw* para exibir cada *widget* na tela.

```
      while(pwbCount>0) do
            if(pwbList[pwbN-pwbCount+1]~=nil) then
                  if(pwbList[pwbN-pwbCount+1]:getFocus() or
firstRedraw) then
                        pwbList[pwbN-pwbCount+1]:redraw()
                  end
                  if(result.value=='(') then
                        result.value = result.value..pwbList[pwbN-
pwbCount+1]:getResult()
                  else
                        result.value =
result.value..','..pwbList[pwbN-pwbCount+1]:getResult()
                  end
            end
            pwbCount=pwbCount-1
      end

      while(ckbCount>0) do
            if(ckbList[ckbN-ckbCount+1]~=nil) then
                  if(ckbList[ckbN-ckbCount+1]:getFocus() or
firstRedraw) then
                        ckbList[ckbN-ckbCount+1]:redraw()
                  end
                  if(result.value=='(') then
                        result.value = result.value..ckbList[ckbN-
ckbCount+1]:getResult()
                  else
                        result.value =
result.value..','..ckbList[ckbN-ckbCount+1]:getResult()
                  end
            end
            ckbCount=ckbCount-1
      end

      while(rdbCount>0) do
            if(rdbList[rdbN-rdbCount+1]~=nil) then
                  if(rdbList[rdbN-rdbCount+1]:getFocus() or
firstRedraw) then
                        rdbList[rdbN-rdbCount+1]:redraw()
                  end
                  if(result.value=='(') then
                        result.value = result.value..rdbList[rdbN-
rdbCount+1]:getResult()
                  else
                        result.value =
result.value..','..rdbList[rdbN-rdbCount+1]:getResult()
                  end
            end
            rdbCount=rdbCount-1
      end
```

Figura 18 (continuação). Função *redraw* para exibir cada *widget* na tela.

```
      while(ltbCount>0) do
            if(ltbList[ltbN-ltbCount+1]~=nil) then
                  if(ltbList[ltbN-ltbCount+1]:getFocus() or
firstRedraw) then
                        ltbList[ltbN-ltbCount+1]:redraw()
                  end
                  if(result.value=='(') then
                        result.value = result.value..ltbList[ltbN-
ltbCount+1]:getResult()
                  else
                        result.value =
result.value..','..ltbList[ltbN-ltbCount+1]:getResult()
                  end
            end
            ltbCount=ltbCount-1
      end

      while(lgdCount>0) do
            if(lgdList[lgdN-lgdCount+1]~=nil or firstRedraw) then
                  if(lgdList[lgdN-lgdCount+1]:getFocus()) then

                        lgdList[lgdN-lgdCount+1]:redraw()
                  end
            end
            lgdCount=lgdCount-1
      end

      {…}

      canvas:flush()
      firstRedraw = false
end
```

Figura 18 (continuação). Função *redraw* para exibir cada *widget* na tela.

É importante destacar que a API possui outros módulos, responsáveis por controlar a navegação entre os elementos no formulário, gerenciar o foco para os elementos, além de processar os eventos de teclas para viabilizar a interação pelo controle remoto.

Alguns exemplos do formato de parâmetro para todos os *widgets* disponíveis na API são apresentados a seguir:

- Botão de estados

```
“statebutton(1;OK;800;655;105;35;blue;24;black)”
```

Figura 19. Exemplo de parâmetros para o botão de estados.

- Caixa de texto

```
“textbox(2;Nome:;430;50;70;0;350;16;white;black;12;black;10)
```

Figura 20. Exemplo de parâmetros para a caixa de texto.

- Caixa de senha

```
“passwordbox(3;Senha:;430;90;70;0;350;16;white;black;12;black;10)”
```

Figura 21. Exemplo de parâmetros para a caixa de senha.

- Caixa de seleção

```
“checkbox(4;4;Linguagens;430;490;220;120;white;12;black;C;C++;Java;
Python)”
```

Figura 22. Exemplo de parâmetros para a caixa de seleção.

- Botão de opção

```
“radiobuttons(5;3;Cor;430;150;220;80;white;12;black;Vermelho;Verde;
Azul)”
```

Figura 23. Exemplo de parâmetros para o botão de opção.

- Caixa de listagem

```
“listbox(6;7;Estado;680;490;220;118;white;6;black;BA;MG;PR;RJ;RS;SC
;SP)”
```

Figura 24. Exemplo de parâmetros para a caixa de listagem.

- Legenda

```
“legend(1;3;1050;660;210;60;white;9;black;OK;ENTER;Apagar;RED;Próxi
mo;CURSOR_RIGHT)”
```

Figura 25. Exemplo de parâmetros para a legenda.

Após a execução da aplicação com a GUI criada, será retornado, na forma de um evento Lua de nome “*result*”, uma *string* com os resultados das seleções em cada *widget* da GUI. Tal *string* tem o formato de par ordenado (ordem de foco, dado(s) de saída), com o item “ordem de foco” atuando como identificador do elemento. A Figura 19 apresenta um exemplo de formato para a *string* retornada. Esses dados podem ser utilizados pelo desenvolvedor da aplicação de modo que sejam processados no restante da execução.

```
((7,On),(1,texto),(2,senha),(5,Televisão,Computador),(3,Masculino),
(4,Solteiro),(6,Rio de Janeiro))
```

Figura 26. Dados de saída retornados.

Por último, a fim de mostrar uma utilização completa da API implementando uma GUI, com todos os *widgets* sendo utilizados em um formulário, um documento NCL que incorpora o arquivo NCLForms.lua é apresentado como exemplo na Figura 20.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<ncl title='NCLua'
xmlns='http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd'>
      <head>
            <regionBase>
                  <region width="1280" height="680" left="0" top="0"
id="rgForms"/>
                  <region width="1280" height="40" left="0"
top="680" id="rgVideo"/>
```

Figura 27. Documento NCL exemplificando o uso da GUI NCLForms.

```
            </regionBase>
            <descriptorBase>
                  <descriptor id="dsVideo" region="rgVideo"/>
                  <descriptor id="dsForms" region="rgForms"
focusIndex="formsIdx"/>
            </descriptorBase>
            <connectorBase>
                  <causalConnector id="onBeginSetN">
                        <connectorParam name="var"/>
                        <simpleCondition role="onBegin"/>
                        <simpleAction role="set" value="$var"
max="unbounded" qualifier="par"/>
                  </causalConnector>
                  <causalConnector id="onEndAttributionSet">
                        <connectorParam name="var"/>
                        <simpleCondition role="onEndAttribution"/>
                        <simpleAction role="set" value="$var"/>
                  </causalConnector>
            </connectorBase>
      </head>
      <body>
            <media type="application/x-ginga-settings"
id="programSettings">
            <property name="service.currentKeyMaster"
value="formsIdx"/>
            </media>
            <port id="entryPoint" component="forms"/>
            <media id="forms" src="NCLForms.lua"
descriptor="dsForms">
                  <property name="result"/>
                  <property name="values"/>
            </media>
            <port id="entryPoint2" component="saida"/>
            <media id="saida" src="output.lua"
descriptor="dsVideo">
                  <property name="text"/>
            </media>
            <link xconnector="onBeginSetN">
                  <bind role="onBegin" component="forms"/>
                  <bind role="set" component="forms"
interface="values">
                        <bindParam name="var"
value="textbox(1;NOME:;430;50;70;0;350;16;white;black;12;black;10),
passwordbox(2;SENHA:;430;90;70;0;350;16;white;black;12;black;10),ra
diobuttons(3;2;SEXO;430;150;220;80;white;12;black;Masculino;Feminin
o), radiobuttons(4;4;ESTADO
CIVIL;680;150;220;120;white;12;black;Solteiro;Casado;Divorciado;Viú
vo),checkbox(5;4;APARELHOS;430;320;220;120;white;12;black;TV
Digital;Computador;Geladeira;Ar-
Condicionado),listbox(6;7;CIDADE;680;320;220;118;white;6;black;Belo
Horizonte;Brasília;Curitiba;Porto Alegre;Rio de Janeiro;São
Paulo;Salvador),statebutton(7;ENVIAR;600;500;105;35;blue;24;black),
legend(1;3;1050;610;210;60;white;9;black;OK;ENTER;Apagar;RED;Próxim
o;CURSOR_RIGHT),legend(2;3;1050;610;210;60;white;9;black;OK;ENTER;A
pagar;RED;Next;CURSOR_RIGHT),legend(3;5;1050;610;210;60;white;9;bla
ck;OK;ENTER;Up;CURSOR_UP;Down;CURSOR_DOWN;Next;CURSOR_RIGHT;Previou
s;CURSOR_LEFT),
```

Figura 27 (continuação). Documento NCL exemplificando o uso da GUI NCLForms.

```
legend(4;5;1050;610;210;60;white;9;black;OK;ENTER;Up;CURSOR_UP;Down
;CURSOR_DOWN;Next;CURSOR_RIGHT;Previous;CURSOR_LEFT),legend(5;5;105
0;610;210;60;white;9;black;OK;ENTER;Up;CURSOR_UP;Down;CURSOR_DOWN;N
ext;CURSOR_RIGHT;Previous;CURSOR_LEFT),legend(6;5;1050;610;210;60;w
hite;9;black;OK;ENTER;Up;CURSOR_UP;Down;CURSOR_DOWN;Next;CURSOR_RIG
HT;Previous;CURSOR_LEFT),legend(7;2;1050;610;210;60;white;9;black;O
K;ENTER;Previous;CURSOR_LEFT)"/>
                    </bind>
              </link>
              <link xconnector="onEndAttributionSet">
                    <bind role="onEndAttribution" component="forms"
interface="result"/>
                    <bind role="set" component="saida"
interface="text">
                          <bindParam name="var" value="$get"/>
                    </bind>
                    <bind role="get" component="forms"
interface="result"/>
              </link>
        </body>
</ncl>
```

Figura 27 (continuação). Documento NCL exemplificando o uso da GUI NCLForms.

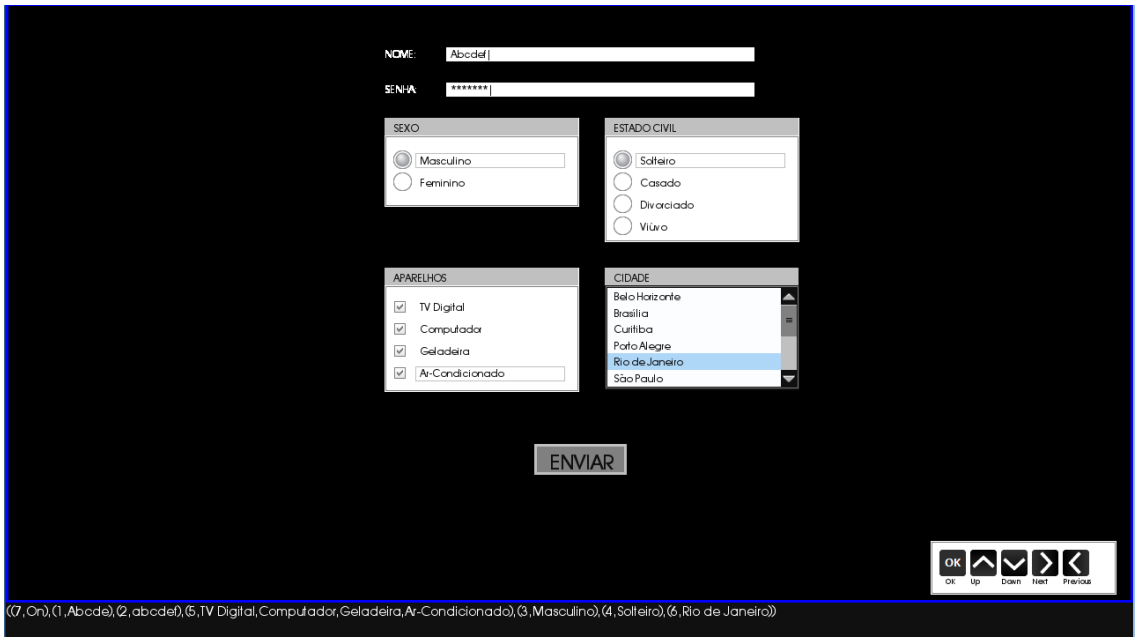

Figura 28. Aplicação usando a API NCLForms com apresentação de parâmetros.

A aplicação, cujo documento NCL é apresentado na Figura 20 e cuja tela é exibida na Figura 21, demonstra o uso da API, adicionando o objeto NCLua através do

objeto de mídia com *id* de valor "*forms*". Outro arquivo NCLua foi incorporado ao documento através do objeto de mídia com *id* de valor "saida". Tal objeto NCLua ficará responsável apenas por exibir na tela a *string* retornada pela implementação da API com os resultados das seleções.

Dentro do cenário de aplicações interativas de TV Digital, levando em consideração as orientações de *design patterns* de interação, a API NCLForms é uma proposta para incentivar a reutilização de software no ambiente Ginga-NCL. Em termos de trabalhos futuros, tal proposta é passível de adição de novas funcionalidades, como a implementação de um teclado virtual a ser exibido na tela e o desenvolvimento de novos *widgets*. Do ponto de vista do que já foi implementado, também seria possível pensar em outro formato para os parâmetros, como diminuir o número de parâmetros obrigatórios ou incluir valores *default* para algumas variáveis, de modo a tornar tal passo mais simples.

5.3. Adoção em *Design Patterns*

As soluções desenvolvidas não ficam restritas ao uso apenas como biblioteca de código em aplicações interativas com formulários. Os elementos de interface gráfica que compõem a API NCLForms (i.e., primitivas gráficas) podem ser aplicados na extensão da linguagem de *patterns* proposta por KUNERT (2009) e apresentada no Capítulo 4.

De forma a ilustrar as associações entre alguns padrões e as primitivas de interface gráfica incluídas na API proposta, a Tabela 11 apresenta essas relações sugeridas. Tal abordagem ocorre com o uso dos elementos em sugestões de solução, apresentadas na Seção 5.3.1.

Design pattern	**Primitiva gráfica**
B2 - Menu	Caixa de listagem (Seção 5.1.2.6)
B4 - Índice	Caixa de listagem (Seção 5.1.2.6)
C6 – Teclas especiais	Botão de estados (Seção 5.1.2.1) Legenda (Seção 5.1.2.7)
D5 – Ocultando aplicação	Botão de estados (Seção 5.1.2.1)
D6 – Ir um nível acima	Botão de opção (Seção 5.1.2.5)
F2 – Votação e perguntas de múltipla escolha	Caixa de seleção (Seção 5.1.2.4) Botão de opção (Seção 5.1.2.5)
F4 – Completar texto	Caixa de listagem (Seção 5.1.2.6)
G3 – Teclado de telefone móvel	Caixa de texto (Seção 5.1.2.2) Caixa de senha (Seção 5.1.2.3)

Tabela 11. Correspondências entre padrões e elementos gráficos da API.

5.3.1. Estendendo a linguagem de *patterns*

Nesta seção, são propostas algumas adições a certos padrões na linguagem apresentada por KUNERT (2009). São citados aqui o *pattern* correspondente e o seu componente que teve extensão analisada e sugerida a partir do estudo conduzido e das características relacionadas identificadas nos *widgets* suportados pela API.

- B2 – Menu
    - Problema:
        - Apresentação do menu: menus estáticos requerem um pouco mais de espaço na tela, mas já apresentam todos os itens do menu de forma visível para o usuário da aplicação.

- Solução: uma forma de menu estático com a exibição de uma parte das opções ou até de todas elas de uma única vez é a primitiva caixa de listagem (Seção 5.1.2.6). Tal elemento possui funcionalidades para exibir, ao mesmo tempo, o número de itens que o desenvolvedor da aplicação desejar, além de destacar a opção selecionada pelo usuário.

- B4 – Índice
  - Problema:
    - Estrutura: um índice pode ser exibido na forma de uma lista em ordem alfabética, ideal para listas com poucos itens, ou em classes de acordo com a primeira letra ou grupos de letras, exibindo os itens cuja primeira letra é igual à letra da página atual ou pertence ao grupo de letras atual. Nessa organização em classes, a cada letra ou grupo de letras, é exibida uma página na tela.
  - Solução: essa lista para representar um índice pode ser implementada com a primitiva caixa de listagem (Seção 5.1.2.6). A lista em ordem alfabética, por ser única, já seria incorporada à aplicação diretamente pelo *widget* da API. As listas organizadas em classes de letras podem fazer uso do mesmo elemento da API, porém necessitam de uma implementação adicional para incluir componentes de navegação entre uma lista e outra.

- C6 – Teclas especiais
  - Problema:

        - Desvantagem: algumas teclas especiais não estão disponíveis para todos os usuários, pois existem vários modelos de controle remoto, que podem não oferecê-las em sua interface. Além disso, quando disponíveis, elas costumam apresentar dificuldades para serem encontradas somente através do tato, necessitando da atenção direta do usuário da TV ao controle remoto.
    - Solução: oferecer um controle remoto estendido, sendo visualizado na tela. Para navegar pelo controle remoto virtual e selecionar qual tecla deveria ser pressionada, o usuário deveria utilizar apenas as teclas de setas e OK, que normalmente são fáceis de serem identificadas pelo tato. O elemento botão de estados da GUI (Seção 5.1.2.1) poderia ser adotado em conjunto com o elemento legenda (Seção 5.1.2.7) nesse passo, representando as teclas do controle remoto estendido na tela, além de indicações sobre navegação no controle estendido e sobre a função de cada tecla.

- D5 – Ocultando aplicação
    - Problema:
        - Posição do indicador de navegação em tela: o indicador de que a aplicação ainda está em execução deve ficar em um dos cantos da tela, sendo visível, mas não atrapalhando a exibição do conteúdo atual na tela da TV.
    - Solução: utilizar como indicador um elemento cuja interface se altere de forma a transmitir a mudança de estados entre visível e oculta da aplicação. Como sugestão de implementação, vem o uso da primitiva

gráfica botão de estados (Seção 5.1.2.1) com dimensões reduzidas. Sua caraterística de alterar a cor de fundo ao trocar seu “estado” permite uma indicação intuitiva. Como exemplo, pode-se utilizar como rótulo para o botão a palavra “Visível”, cabendo ao usuário selecionar o seu estado ativo para exibir a aplicação e seu estado inativo para ocultá-la.

- D6 – Ir um nível acima
    - Problema:
        - Indicadores de navegação em tela: normalmente não se usam indicadores para esta função, pois a mesma é ativada por meio de uma tecla própria no controle remoto.
    - Solução: dependendo da complexidade da aplicação, onde haja várias camadas com diversas opções a serem selecionadas, pode ser interessante adotar alguma indicação em tela, a fim de permitir ao usuário retornar ao nível acima do nível atual. A implementação dessa proposta poderia fazer uso do elemento botão de estados (Seção 5.1.2.1) de forma a exibir o nível acima na estrutura hierárquica de níveis de uma aplicação.

- F2 – Votação e perguntas de múltipla escolha
    - Problema:
        - Apresentação das opções de resposta: as opções das questões de múltipla escolha podem ser apresentadas em diferentes formatos, como texto, imagens ou animações, variando-se de acordo com o tipo de pergunta ou grupo alvo de usuários.

- Solução: quando a aplicação demandar opções de respostas em formato de texto, sua implementação pode ser facilmente alcançada através dos elementos caixa de seleção (Seção 5.1.2.4) e botão de opção (Seção 5.1.2.5), dependendo do número de respostas permitidas.

- F4 – Completar texto
  - Problema:
    - Apresentação de opções de resposta: como a digitação de caracteres em texto pelo controle remoto é uma tarefa complicada, uma sugestão é apresentar opções de resposta ao usuário para que o mesmo selecione uma para completar o texto em questão.
  - Solução: apresente as opções em formas de menu, para que o usuário possa selecionar o item que considerar correto. Essa solução pode ser implementada por meio do elemento caixa de listagem (Seção 5.1.2.6), com cada lacuna do texto disparando a exibição de uma *list box* por vez.

- G3 – Teclado de telefone móvel
  - Problema:
    - Escolha de caracteres: os caracteres são selecionados através do teclado numérico do controle remoto, assim como nos teclados numéricos de aparelhos celulares. Tal sistema é conhecido como "*Text on* 9 *keys*" ou T9 (T9, 2012). Como exemplo, se o usuário pressionar a tecla 2 do controle remoto uma vez, selecionará o

caractere “a”. Se pressionar a tecla duas vezes, será selecionado “b”. Pressionando-se três vezes a tecla 2, o caractere “c” é selecionado e quatro vezes, será selecionado o caractere “2”. Somente pressionando-se cinco vezes o caractere “a” será selecionado novamente, reiniciando o ciclo de seleção de opções para este botão.

- Solução: optando-se pela solução sem exibição de teclado estendido na tela, esse padrão pode ser implementado através das primitivas caixa de texto (Seção 5.1.2.2) ou caixa de senha (Seção 5.1.2.3), dependendo da função planejada para o elemento, pois tais *widgets* já foram desenvolvidos com o sistema T9 incorporado.

Com a API desenvolvida e as propostas de extensão à linguagem de *patterns*, verifica-se a possibilidade de aplicar padrões estabelecidos na literatura no ambiente Ginga-NCL. Cabe destacar também que o presente estudo representa uma contribuição em certa linguagem de *design patterns*. Trabalhos futuros podem ampliar tal discussão, aplicando novas propostas de linguagem de padrões ao cenário de TV Digital no Brasil.

6. Conclusão

Apesar de ter pouco tempo desde sua padronização, o cenário de TV Digital no Brasil ainda tem espaço para muitas contribuições. Nesse sentido, o presente estudo buscou analisar aspectos referentes ao desenvolvimento com reutilização para aplicações de TV Digital.

A fim de apresentar as considerações finais do estudo, a Seção 6.1 reúne as contribuições do presente trabalho, a Seção 6.2 expõe as limitações encontradas e a Seção 6.3 discute possíveis trabalhos futuros.

6.1. Contribuições

Constatando-se a carência de aplicações e, sobretudo, de ferramentas voltadas para tarefas de desenvolvimento no ambiente Ginga-NCL, buscou-se basear a parte de implementação desse estudo nessa área.

A partir do estudo inicial das linguagens NCL e NCLua, a dificuldade de implementar elementos de interface gráfica de modo simplificado inspirou a decisão da proposta de uma API voltada para o desenvolvimento de GUIs. A fase inicial do desenvolvimento, criando aplicações de exemplo somente com NCL e, posteriormente, incorporando objetos NCLua, serviu de treinamento e foi fundamental para se atingir o objetivo final do estudo.

O conjunto inicial de *widgets* foi desenvolvido de maneira individual, ainda em fase de estudos da linguagem NCLua. Tal conjunto tinha o tamanho aproximado de metade do conjunto final de elementos. Com a decisão de se unificar a GUI em um

único elemento NCLua, foi possível padronizar o projeto, o que facilitou a integração entre os elementos oferecidos na API.

Dessa forma, a API NCLForms, desenvolvida para o ambiente Ginga-NCL, que viabiliza a construção de GUIs para aplicações interativas em TV Digital, representa a principal contribuição deste presente trabalho. Sua proposta vem acompanhada de instruções de uso e exemplos com código-fonte de aplicações, de forma a tornar sua adoção mais fácil.

Com a descrição dos padrões de projeto de interface em aplicações de TV Digital, sua descrição, na forma de uma linguagem, colabora no entendimento de problemas e propostas de soluções voltados para esse tipo de aplicações.

Após a identificação da linguagem de *patterns*, sua associação com elementos de GUI, revelando possíveis aplicações de primitivas de interface gráfica suportadas pela API, também representam possíveis aplicações para a própria API, além de buscar promover uma abordagem de padrões de projeto no ambiente de TV Digital brasileiro. Assim sendo, o apoio à implementação de *design patterns* representa uma possível contribuição secundária do estudo.

De forma geral, o trabalho busca contribuir para o desenvolvimento de trabalhos voltados para a área de interatividade em termos de TV Digital no Brasil, com destaque para o ambiente Ginga-NCL.

6.2. Limitações

O trabalho apresenta algumas limitações, que podem ser analisadas a fim de se buscar a sua minimização.

No que diz respeito à implementação da API, a mesma é dependente da plataforma e não é aplicável em outros modelos como o Ginga-J, por exemplo. Outro obstáculo existente está na dificuldade em se customizar todas as características possíveis de certos elementos. Aumentar a flexibilidade da API implica elevar a sua complexidade.

Quanto à extensão da linguagem de *patterns*, não foi possível realizar testes de validação de cada padrão. Com um período de tempo maior, seria possível testar não apenas os padrões relacionados com *widgets* da API NCLForms, mas também todo o conjunto de *design patterns.*

6.3. Trabalhos Futuros

Devido a limitações de prazo ao longo do desenvolvimento do trabalho, alguns objetivos iniciais não puderam ser implementados. Além desses pontos, outras propostas foram pensadas durante o estudo.

Como possíveis trabalhos futuros, pode-se focar na extensão da API, com a implementação de novos *widgets*, na forma de primitivas de maior nível de abstração baseadas nas primitivas já disponíveis, e contêineres para tais elementos, como primitivas recursivas. Outras mudanças na API incluiriam a alteração no formato dos parâmetros. Uma possibilidade de implementação seria a adoção de expressões regulares no elemento caixa de texto (Seção 5.1.2.2), de forma que se possa controlar e validar os dados digitados em tal *widget*. Seria interessante também implementar mecanismos de verificação dos parâmetros passados através do documento NCL, como uma forma de tratamento de exceções. Para contornar o problema da dependência de

plataforma, pode-se buscar também a extensão da API para outros dispositivos, conforme proposto por (PABLO & PETRI, 2002).

Outro trabalho futuro poderia estar relacionado à integração da API NCLForms com o projeto aNa (API NCL para Autoria) (ANA, 2012), que visa a geração de documentos NCL através da linguagem Java.

No que diz respeito à linguagem de *patterns* estudada, é possível pensar em identificar novas formas de apoiar a implementação sobre cada padrão. Durante essa tarefa, seria importante realizar testes de usabilidade a fim de validar cada solução proposta e confirmar a utilidade dos padrões.

Trabalhos futuros nesse contexto podem cooperar para estimular o desenvolvimento de novas aplicações e ferramentas no ambiente Ginga-NCL, o que é benéfico para colaborar na confirmação de tal ambiente como um padrão de grande valor e no qual investimentos devem ser realizados.

Referências Bibliográficas

ABNT, 2008, *Televisão Digital Terrestre – Codificação de dados e Especificações de Transmissão para radiodifusão Digital – Parte 4: Ginga-J – Ambiente para a Execução de Aplicações Procedurais.* Rio de Janeiro, ABNT.

ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I., ANGEL, S., 1977 *apud* BORCHERS, J. O., 2000, Interaction Design Patterns: Twelve Theses. In: *Proceedings of CHI 2000.*

ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I., ANGEL, S., 1977 *apud* KUNERT, T., 2009, *User-Centered Interaction Design Patterns for Interactive Digital Television Applications.* London, Springer-Verlag.

ALNUSAIR, A., ZHAO, T., BODDEN, E., 2010, Effective API Navigation and Reuse. In: *Information Reuse and Integration*, pp.7-12.

ANA, *aNa*, 2012. Disponível em: <http://joel.dossantos.eng.br/aNa/>. Acesso em: 14 fev 2012, 20:00:00.

APACHE STRUTS, *Struts*, 2012. Disponível em: <http://struts.apache.org/>. Acesso em: 14 fev 2012, 20:00:00.

APPLETON, B., 1997 *apud* MALDONADO, J. C., BRAGA, R. T. V., GERMANO, F. S. R., MASIERO, P. C., 2002, *Padrões e Frameworks de Software*. Notas Didáticas, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, ICMC/USP, São Paulo, SP, Brasil.

BUSCHMANN, F., 1996 *apud* MALDONADO, J. C., BRAGA, R. T. V., GERMANO, F. S. R., MASIERO, P. C., 2002, *Padrões e Frameworks de Software*. Notas

Didáticas, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, ICMC/USP, São Paulo, SP, Brasil.

COMMON CONTROLS, *Common Controls Library*, 2012. Disponível em: <http://msdn.microsoft.com/en-us/library/windows/desktop/bb775493(v=vs.85).aspx>. Acesso em: 15 fev 2012, 10:00:00.

EUROPEAN BROADCASTING UNION, 2004a, apud KUNERT, T., 2009, *User-Centered Interaction Design Patterns for Interactive Digital Television Applications*. London, Springer-Verlag.

FREEMAN, M., 1980 *apud* MILER JÚNIOR, N., 2000, *A Engenharia de Aplicações no Contexto da reutilização Baseada em Modelos de Domínio*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., 1998, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Mass.

GAWLINSKI, M., 2003 *apud* KUNERT, T., 2009, *User-Centered Interaction Design Patterns for Interactive Digital Television Applications*. London, Springer-Verlag.

GTK, *The GTK+ Project*, 2012. Disponível em: <http://www.gtk.org/>. Acesso em: 15 fev 2012, 10:00:00.

HASEBRINK, U., 2001 *apud* KUNERT, T., 2009, *User-Centered Interaction Design Patterns for Interactive Digital Television Applications*. London, Springer-Verlag.

IERUSALIMSCHY, R., FIGUEIREDO, L. H., CELES, W., 1996, The Evolution of Lua. In: *ACM SIGPLAN History of Programming Languages Conference III*, San Diego, California, USA, 09-10 June.

JÄÄSKELÄINEN, K., 2001, *Strategic Questions in the Development of Interactive Television Programs.* Ph.D. dissertation, University of Art and Design Helsinki, Helsinki, Finland.

JAVA SE DESKTOP, *Swing*, 2012. Disponível em: <http://java.sun.com/javase/technologies/desktop/>. Acesso em: 14 fev 2012, 20:00:00.

JOHNSON, R. E., WOOLF, B., 1991 *apud* MALDONADO, J. C., BRAGA, R. T. V., GERMANO, F. S. R., MASIERO, P. C., 2002, *Padrões e Frameworks de Software*. Notas Didáticas, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, ICMC/USP, São Paulo, SP, Brasil.

KAWRYKOW, D., ROBILLARD, M. P., 2009, Improving API Usage Through Automatic Detection of Redundant Code. In: *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pp.111-122, Washington, DC, USA.

KIRK, D., ROPER, M., WOOD, M., 2002, Defining the Problems of Framework Reuse. In: *International Computer Software and Applications Conference* (COMPSAC 2002), pp. 623-626.

KUNERT, T., 2009, *User-Centered Interaction Design Patterns for Interactive Digital Television Applications.* London, Springer-Verlag.

LABORATÓRIO TELEMÍDIA - PUC-RIO, *SAGGA - Suporte para a geração automática de aplicações para o Ginga-NCL*, 2012. Disponível em: <http://www.telemidia.puc-rio.br/?q=en/projetoSAGGA>. Acesso em: 15 fev 2012, 14:00:00.

MALDONADO, J. C., BRAGA, R. T. V., GERMANO, F. S. R., MASIERO, P. C., 2002, *Padrões e Frameworks de Software*. Notas Didáticas, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, ICMC/USP, São Paulo, SP, Brasil.

MOTIFZONE, *Motif*, 2012. Disponível em: <http://www.openmotif.org/>. Acesso em: 15 fev 2012, 10:00:00.

NANARD, M., NANARD, J., KAHN, P., 1998, Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. In: *Proceedings of 9th ACM Conference on Hypertext and Hypermedia*, Pittsburgh, Pennsylvania.

NCL ECLIPSE, *NCL Eclipse*, 2012. Disponível em: <http://laws.deinf.ufma.br/ncleclipse/#.T12dljES38c>. Acesso em: 07 jan 2012, 16:00:00.

PABLO, C., PETRI, V., 2002, A Graphical User Interface Framework for Digital Television. In: *Proceedings of the 10th International Conference on Computer Graphics, Visualization and Computer Vision, WSCG'2002*, pp. 1-4, Czech Republic.

PAGANI, M., 2003 *apud* KUNERT, T., 2009, *User-Centered Interaction Design Patterns for Interactive Digital Television Applications.* London, Springer-Verlag.

PORTAL DO SOFTWARE PÚBLICO BRASILEIRO, *Ginga,* 2012. Disponível em: <http://www.softwarepublico.gov.br/ver-comunidade?community_id=1101545>. Acesso em: 07 jan 2012, 17:00:00.

RATAMERO, E. M., 2007, *Tutorial sobre a Linguagem de Programação NCL (Nested Context Language).* Disponível em: <http://www.telecom.uff.br/pet/petws/downloads/tutoriais/ncl/ncl2k71203.pdf>. Acesso em: 10 fev 2012, 4:00:00.

ROBILLARD, M. P., 2009, What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software – Special Issue on the Collaborative and Human Aspects of Software Engineering*, v.26, pp.26-34.

RUBART, J., 2008, Hypermedia Design Patterns. In: *ACM Conference on Hypertext and Hypermedia*. Pittsburgh, Pennsylvania, USA.

SCODITTI, A., STUERZLINGER, W., 2009, A New Layout Method for Graphical User Interfaces. In: *2009 IEEE Toronto Internation Conference – Science and Technology for Humanity,* Toronto, Canada.

SHARP, H., ROGERS, Y., PREECE, J., 2007, *Interaction Design: Beyond Human-Computer Interaction*. West Sussex, Wiley.

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

SOARES, L. F. G., SOARES NETO, C. S., 2009, *Nested Context Language 3.0: Reúso e Importação*. Monografias em Ciência da Computação, PUC-Rio, Rio de Janeiro, Brasil.

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.

SOURCEFORGE, *akdebugger*, 2012. Disponível em: <http://sourceforge.net/projects/akdebugger/>. Acesso em: 07 jan 2012, 16:00:00.

SOUZA FILHO, G. L., LEITE, L. E. C., BATISTA, C. E. C. F., 2007, Ginga-J: The Procedural Middleware for the Brazilian Digital TV System. *Journal of the Brazilian Computer Society*, vol.12, n.4.

STYLOS, J., GRAF, B., BUSSE, D., ZIEGLER, C., EHRET, R., KARSTENS, J., 2008, A Case Study of API Redesign for Improved Usability. In: *VL/HCC*, pp.189-192.

T9, *T9 Text Input*, 2012. Disponível em: <http://www.t9.com/portuguese/>. Acesso em: 14 fev 2012, 14:00:00.

THE ECLIPSE FOUNDATION, *Eclipse*, 2012. Disponível em: <http://www.eclipse.org/>. Acesso em: 07 jan 2012, 16:00:00.

THE LINUX INFORMATION PROJECT, 2004, *GUI Definition*. Disponível em: <http://www.linfo.org/gui.html>. Acesso em: 12 fev 2012, 13:00:00.

VAN WELIE, M., VAN DER VEER, G. C., 2003 *apud* KUNERT, T., 2009, *User-Centered Interaction Design Patterns for Interactive Digital Television Applications.* London, Springer-Verlag.

VMWARE PLAYER, *VMware Player*, 2012. Disponível em: <http://www.vmware.com/products/player/>. Acesso em: 07 jan 2012, 16:00:00.

WANG, L. J., SAJEEV, S. M., INCHAIWONG, L., 2006, A Formal Specification of Interaction Widgets Hierarchy Framework. In: *Third International Conference on Information Technology: New Generations*, Las Vegas, USA.

XCODE, *Xcode*, 2012. Disponível em: <https://developer.apple.com/technologies/tools/whats-new.html#interface-builder>. Acesso em: 14 fev 2012, 20:00:00.