ARQUITETURA DE COMPONENTES E PRÁTICAS ÁGEIS PARA O MIDDLEWARE GINGA-NCL

Laécio Freitas CHAVES (1); Fábio de Jesus Lima GOMES (2)
(1) *Laboratory of Innovation on Multimedia Systems* (LIMS)
Instituto Federal de Educação, Ciência e Tecnologia do Piauí (IFPI)
Praça da Liberdade, n°1597, Centro. Tel:(86) 3215-5224, Fax: (86) 3215-5206,
e-mail: laeciofreitas@gmail.com
(2) LIMS - IFPI, e-mail: fabio@ifpi.edu.br

RESUMO

O padrão brasileiro de TV Digital é baseado no *ISDB-T* (Integrated Services Digital Broadcasting Terrestrial), padrão japonês e também é constituído por tecnologias desenvolvidas por universidades brasileiras. Uma destas tecnologias é o middleware Ginga, que possibilita o desenvolvimento de aplicações interativas. As normas brasileiras de TV Digital, publicadas pelo Sistema Brasileiro de TV Digital e a ABNT, padronizam a TV Digital, mas não definem regras e práticas para o desenvolvimento de aplicações.

Esse trabalho apresenta uma arquitetura orientada a componentes para facilitar e modularizar o desenvolvimento de aplicativos que utilizam o ambiente Ginga-NCL, permitindo o reuso de código entre aplicativos e a criação de novos componentes a partir da extensão de qualquer componente padrão. O trabalho também trata da utilização de DSLs para a criação de ferramentas que facilitam a adoção de práticas ágeis no desenvolvimento de aplicações declarativas, tais como: TDD (Test Driven Development), BDD (Behaviour-Driven Development) e refatoração.

Palavras-chave: TV Digital, testes, framework, DSL, práticas ágeis

1 INTRODUÇÃO

A TV Digital traz uma série de vantagens em relação a TV analógica, tais como: imagens de alta definição, melhor qualidade de áudio e vídeo. Além disso, uma camada de software que atua como intermediária entre as aplicações e a infra-estrutura de execução (hardware receptor do sinal digital e o sistema operacional) vai possibilitar o desenvolvimento de aplicações interativas, tais como: *t-learning* (aprendizagem pela TV), *t-gov* (governo televisivo), *t-commerce* (comercio pela TV) e *t-health* (aplicações relacionadas a saúde). Essa camada de software é chamada de middleware, e no Brasil recebeu o nome de Ginga (2010).

O Ginga dispõe de uma arquitetura interligada com dois subsistemas principais: Ginga-J, que é o ambiente para o processamento de aplicações imperativas utilizando Java e o Ginga-NCL, ambiente para a apresentação de aplicações declarativas desenvolvidas com a linguagem NCL e a linguagem Lua com módulos adicionados a sua biblioteca padrão, para permitir a comunicação entre um documento NCL e um script Lua (NCLua).

Há décadas a industria de software vem buscando práticas para diminuir riscos nos projetos de software e tornar o desenvolvimento mais ágil. Esse trabalho descreve uma abordagem para prover uma arquitetura de desenvolvimento e criar ferramentas que facilitem o desenvolvimento de aplicações utilizando práticas ágeis. A abordagem se baseia no padrão brasileiro de TV Digital, especificamente no ambiente Ginga-NCL e prevê a utilização de práticas ágeis, como refatoração e TDD para o desenvolvimento de aplicações de melhor qualidade e de forma mais econômica.

Esse documento está organizado da seguinte forma: A seção 2 discorre sobre os objetivos e a metodologia adota no trabalho. A seção 3 trata dos trabalhos relacionados. A seção 4 apresenta uma arquitetura baseada em componentes para o desenvolvimento de aplicações declarativas. A seção 5 discorre sobre a utilização de DSLs para a criação de suítes de testes no desenvolvimento de aplicações declarativas e sobre práticas ágeis. A seção 6 apresenta as pesquisas em andamento e os trabalhos futuros.

2 OBJETIVOS E METODOLOGIA

O trabalho tem por objetivo desenvolver um framework baseado em uma arquitetura orientada a componentes para o desenvolvimento de aplicações declarativas para o Sistema Brasileiro de TV Digital, mas especificamente, no subsistema Ginga-NCL do middleware brasileiro Ginga. O trabalho também propõe o desenvolvimento de ferramentas para criar e automatizar uma suíte de testes para a linguagem NCL e para os scripts Lua (NCLua).

Dois modelos foram adotados para o desenvolvimento das ferramentas: *top-down* e *bottom-up*. A semântica do *top-down* preocupa-se com o que fazer e não no como fazer. Por outro lado, a semântica do *bottom-up* preocupa-se no como, sem fazer muito planejamento no começo, melhorando continuamente e utilizando refatoração.

3 TRABALHOS RELACIONADOS

Outros trabalhos discutem a utilização de uma arquitetura de componentes para o desenvolvimento de aplicações declarativas. (SOUSA JÚNIOR, 2009), apresenta uma ferramenta de autoria para o desenvolvimento de aplicações declarativas para TV Digital utilizando o framework de componentes gráficos LuaOnTV. Esta ferramenta apresenta um resultado WYSIWYG (What You See Is What You Get), permite a criação de templates em arquivos XML e dispõe de uma interface gráfica para o desenvolvimento no formato *drag-and-drop* (arrastar e soltar), onde os componentes podem ser arrastados e manipulados através do mouse.

(PROTA, 2009) apresenta uma arquitetura de componentes para formalizar o desenvolvimento de aplicações declarativas em NCL e Lua, permitindo o reuso e minimização do código no desenvolvimento de aplicações.

Guimarães et al. (2007) desenvolveram o ambiente de autoria Composer, que possibilita a autoria de aplicações declarativas em NCL em quatro visões: Visão gráfica estrutural, visão gráfica de leiaute, visão gráfica temporal e visão textual. As quatro visões funcionam sincronamente, a fim de oferecer um ambiente integrado aos autores. Além da ferramenta de autoria propriamente dita, o Composer também oferece um simulador, que permite, que o autor de um documento possa disparar sua exibição. No entanto, o Composer foi descontinuado (GINGA-NCL, 2010).

Em comparação aos trabalhos citados acima, este trabalho apresenta (além de uma arquitetura orientada a componentes visuais) a criação de componentes para tratamento de eventos entre o formatador NCL e o script NCLua, componentes para o uso do canal de interatividade e a criação de ferramentas que possibilitam a criação de um conjunto sólido de testes no desenvolvimento de aplicações declarativas, incentivando a adoção de práticas consolidadas no desenvolvimento de software.

4 FRAMEWORK DE COMPONENTES

O framework de componentes aqui descrito visa propor uma arquitetura orientada a componentes para facilitar a autoria de aplicações no ambiente Ginga-NCL. A arquitetura dos componentes (figura 1) é baseada na arquitetura do LWUIT (Lightweight UI Toolkit Library) (LWUIT, 2010) e no conceito de modularidade para oferecer funcionalidades para o desenvolvimento de aplicações para TV Digital, tais como: suporte a componentes visuais, navegação entre os componentes, i18n (internacionalização), tratamento de eventos de interface gráfica com o usuário e uso do canal de interatividade. Outras funcionalidades são deixadas para o reuso de código e o desenvolvimento de plug-ins.

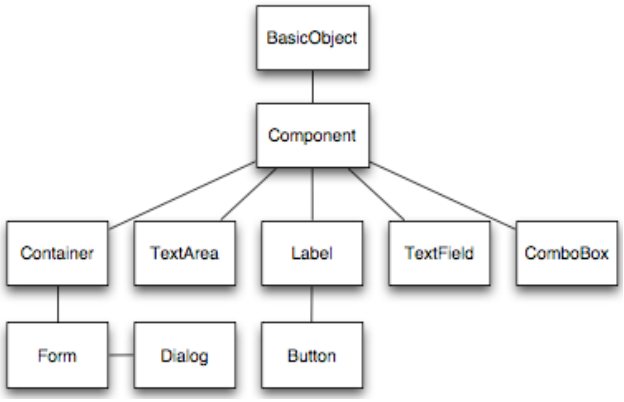

Figura 1 – Hierarquia de classes do framework

LOS (Lua Object System) (LOS, 2010) é um poderoso sistema de programação orientada a objetos para a linguagem Lua que está sendo utilizado para o desenvolvimento do framework.

O framework proposto é agnóstico, ou seja, o desenvolvedor pode escolher o framework de testes de sua preferência, quais componentes serão carregados nas suas aplicações e excluir os componentes que não serão utilizados.

4.1 Componentes Gráficos.

Composite é o padrão de projeto utilizado no desenvolvimento do framework proposto. Segundo FREEMAN (2007) esse padrão permite a composição de objetos em estrutura de árvore para representar hierarquias parte-todo.

A estrutura de componentes é modularizada e extensível, permitindo que o desenvolvedor crie novos componentes a partir da extensão de qualquer componente padrão. Todos os componentes visuais herdam da classe base Component, seguem padrões de usabilidade para TV Digital e utilizam os módulos adicionados à biblioteca padrão de Lua. Abaixo seguem detalhes dos componentes visuais.

- Component: Esse é a classe base dos componentes. É uma representação gráfica abstrata dos componentes que podem interagir com o usuário.
- Container: É uma classe padrão para composição de componentes. Permitir organizar múltiplos componentes utilizando uma arquitetura de componentes plugáveis.
- Form: É um container composto por um título que pode ser escrito na parte superior e um menu na parte inferior. O espaço entre a parte superior e inferior é para o conteúdo do *form*, que carrega os componentes a serem colocados no formulário.
- Button: Componente para dar suporte a exibição de botões coloridos do controle-remoto (vermelho, verde, amarelo e azul). Permite texto descritivo ou imagem nos botões.
- ComboBox: Componente gráfico que representa uma lista de valores *drop-down*. Permite apenas uma seleção por vez através da interação do usuário.
- TextArea: Componente que permite exibir e editar texto em múltiplas linhas utilizando os botões do controle remoto.
- TextField: Componente para recebimento de entrada de texto (em uma única linha) do usuário
- Label: Componente que permite a exibição de *labels* e imagens que não são selecionáveis e não interagem com o usuário. É a classe base de Button.
- Dialog: Componente que permite transmitir informações para o usuário em tempo de execução para obter um feedback do usuário. Imagens podem ser incluídas a um *dialog* para representar um diálogo graficamente.
- Image: Abstração para manipular imagens como objetos.

4.2 Estrutura de Diretórios de uma Aplicação.

A estrutura de diretórios de uma aplicação que utiliza o framework é baseada na estrutura de diretórios do framework Web Ruby on Rails (FERNANDEZ, 2008). A intenção dessa estrutura de diretórios é permitir que desenvolvedores tenham foco nos problemas reais, que não incluem "Qual nome dar ao diretório onde determinados arquivos estão?". Abaixo veja uma breve descrição dos diretórios.

- app: Diretório onde fica o código e as mídias da aplicação.
- app/events: Diretório onde ficam a classes NCLua para tratar os eventos do documento NCL.
- app/models: Diretório onde ficam as classes Lua de lógica de negócios.
- app/declarative: Diretório onde ficam os documentos declarativos da aplicação (NCL).
- app/public: Diretório onde ficam as mídias.
- app/public/static: Diretório onde ficam as mídias estáticas, como imagens.
- app/public/continues: Diretório onde ficam as mídias contínuas, como vídeos.
- doc: Diretório onde fica a documentação da aplicação.
- spec: Diretorio para testes.
- spec/models: Local dos testes unitários dos modelos (lógica de negócios).
- spec/events: Local dos testes funcionais das classes de eventos.
- spec/declarative: Local dos testes funcionais nos documentos declarativos (NCL).
- vendor: Diretório dos plugins criados pelos desenvolvedores

4.3 Classes de Eventos e Interatividade.

O framework fornece a classe RemoteControlEvent para tratamento de eventos do teclado (interação com o usuário), a classe NclEvent para receber e enviar eventos para o formatador NCL, além da classe InteractivityEvent para o uso do canal de interatividade e da classe I18n para internacionalização. Todas essas classes (de eventos, interatividade e internacionalização) são filhas da classe BasicObject, que é a raiz hierárquica do framework. BasicObject é uma classe muito simples, que não possui muitos métodos e é utilizada para delegar as classes de componentes, eventos e interatividade. (figura 2).

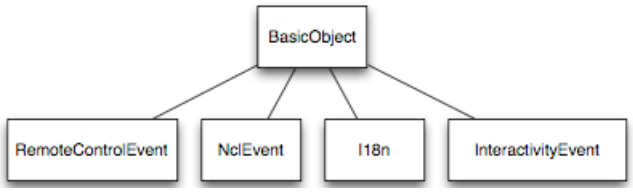

Figura 2 – Hierarquia das classes

5 DSL PARA TESTES EM NCL E NCLUA

Uma DSL (Domain-Specifc Language) é definida como uma linguagem de programação ou uma linguagem de especificação executável que oferece, através de notações e abstrações apropriadas, poder expressivo focado geralmente em um problema de domínio específico (DEURSEN, 2010).

Segundo Martin Fowler (DSL, 2010), podemos classificar uma DSL como interna e externa. DSLs internas são escritas na linguagem do sistema e as DSLs externas tem sua própria sintaxe e precisam de um compilador ou interpretador para serem executadas.

A nova geração de frameworks para BDD (Behaviour-Driven Development ou Desenvolvimento Guiado por Comportamento) são exemplos da utilização de DSLs no desenvolvimento de software. RSpec (RSpec, 2010) é uma DSL interna para BDD desenvolvida com a linguagem Ruby que permite testes mais legíveis, onde os desenvolvedores expressam suas intenções com mais fluidez e o Cumcuber (Cumcuber, 2010) é uma ferramenta para BDD que utiliza uma DSL externa chamada Gherkin para descrever o comportamento do software, sem detalhar como o comportamento é implementado.

O uso de uma DSL desenvolvida em Lua para testes em NCL, NCLua e nos componentes do framework permite a criação de um conjunto de testes para as duas linguagens (NCL e NCLua) e visa proporcionar ao desenvolvimento de aplicações declarativas para SBTVD, a utilização de práticas ágeis, tais como: TDD (Test Driven Development) e BDD (Behaviour-Driven Development). A DSL analisa sintaticamente (parsing) documentos NCL, possibilitando a criação de testes unitários utilizando frameworks de testes para a linguagem Lua (figura 3).

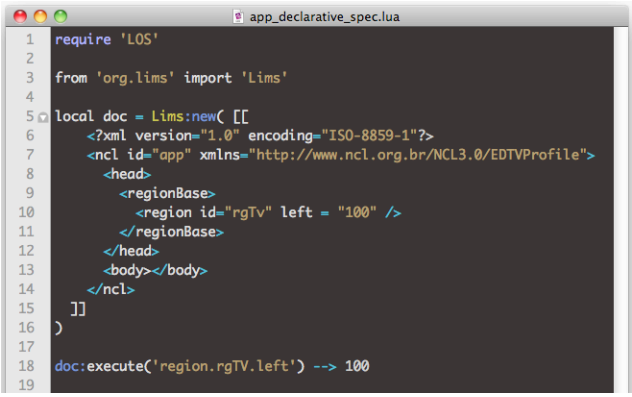

Figura 3 – Análise sintática do documento NCL

Parte da DSL será baseada na biblioteca java FEST-Swing (FEST-SWING, 2010), que fornece uma API para facilitar a criação e a manutenção de testes de GUI (Interface gráfica com o usuário).

Qualquer GUI, mesmo fornecendo recursos simples, provavelmente envolve algum nível de complexidade. Toda complexidade do software deve ser testada, pois o código sem testes é uma fonte potencial de bugs. Durante a manutenção do aplicativo, o código pode ser refeito com frequência para melhorar o design. Ter um sólido conjunto de testes, que inclui código de GUI, pode dar mais confiança para os desenvolvedores não introduzirem bugs inadvertidamente (RUIZ, 2007).

5.1 Refatoração, TDD e BDD.

FOWLER (2004) define refatoração como um processo de alteração de um sistema de software de modo que o comportamento externo do código não mude, mas que sua estrutura interna seja melhorada. O objetivo da refatoração é melhorar a qualidade e a legibilidade do código e não adicionar novas funcionalidades ou corrigir bugs. A medida que novas funcionalidades são desenvolvidas e o código fica desorganizado, pequenas partes do código podem ser refatoradas, tornando o código mais limpo e mais fácil de ser compreendido. A utilização da DSL permitirá a criação de um sólido conjunto de testes, que é uma pré-condição para refatoração, diminuindo as chances de introdução de falhas e o tempo para encontrar erros.

TDD é uma técnica de design que sugere a implementação dos testes antes da implementação, trazendo benefícios como: confiança (teste verifica o comportamento do software no estagio atual de desenvolvimento e no futuro) e ênfase no desenvolvimento (evitando generalizações desnecessárias) BECK (2010).

Segundo TELES (2004) quando o desenvolvedor pensa no teste antes de pensar na implementação, ele é forçado a compreender melhor o problema. Ao se aprofundar no problema, o desenvolvedor está fazendo uma análise mais detalhada. Portanto, neste momento, o desenvolvimento guiado por testes (TDD) atua como uma técnica de análise. Quando o desenvolvedor escreve o teste, ele procura atuar como um cliente dele, se preocupando apenas com a interface externa do método, sem dar atenção à implementação. Isso permite que o design do método seja o mais adequado possível para aqueles que irão utilizá-lo.

Controlar a qualidade das aplicações para TV digital é um grande desafio. A aplicação não deve fazer somente o que o cliente ou telespectador necessita, mas também fazê-lo de forma eficiente, reutilizável, fácil de dar manutenção. No desenvolvimento de aplicações para TV Digital essas características são asseguradas através de testes manuais em módulos da aplicação ou na aplicação inteira.

Behaviour-Driven Development (BDD) ou Desenvolvimento Dirigido por Comportamento é uma evolução no pensamento por trás do Test Driven Development (TDD) e Acceptance Test Driven Planning (BDD, 2010). O BDD é uma reformulação de boas praticas existentes e combina o TDD com o DDD (Domain Driven Design) (DDD, 2010) focando em entregar algo de valor para o cliente.

O BDD se baseia em três princípios:

- A área de negócios e a de tecnologia precisam se referir a mesma parte do sistema da mesma forma. Tudo é comportamento.
- Toda parte do sistema precisa ter um valor identificável e verificável para o negócio. Valor de negócio.
- Analisar, projetar e planejar tudo de cima a baixo tem retorno decrescente. O suficiente é suficiente

6 PESQUISAS EM ANDAMENTO E TRABALHOS FUTUROS

Este trabalho visa contribuir para uma arquitetura de referência para o desenvolvimento de aplicações declarativas para TV Digital no ambiente Ginga-NCL e prover melhorias na qualidade dos aplicativos desenvolvidos. A utilização de DSLs permite a adoção de práticas que visam aumentar a velocidade da entrega do software, aumentando a qualidade do código, reduzindo o número de defeitos, diminuindo o tempo gasto com depuração e até aumentando a produtividade dos desenvolvedores.

O trabalho continua em constante evolução, e alguns tópicos estão sendo levados em consideração, como a utilização de um sistema de gerenciamento de tarefas para a geração automatizada da estrutura de diretórios para apoiar o desenvolvimento das aplicações.

O framework de componentes e a DSL utilizam o paradigma orientado a objetos para o desenvolvimento de aplicações e para os testes. Pelo fato da linguagem Lua oferecer mecanismos para o desenvolvimento orientado a aspectos, serão feitos testes para verificar os benefícios da separação, organização e modularização do código oferecidos pelo paradigma orientado a aspectos.

REFERÊNCIAS

BECK, K., **Extreme Programming Explained, Second Edition: Embrace Change**. Boston, Massachusetts, USA, Addison-Wesley, 2004.

BECK, K., **TDD Desenvolvimento Guiado por Testes**. Porto Alegre: bookman, 2010.

BDD. Disponível em <http://behaviour-driven.org/>. Acessado em maio de 2010.

CUMCUBER. Disponível em <http://cukes.info/>. Acessado em maio de 2010.

DDD. Disponível em <http://cukes.info/>. Acessado em maio de 2010.

DEURSEN, ARIE VAN; KLINT,PAUL; E VISSER, J. **Domain-Specific Languages: An Annotated Bibliography**. March 2nd, 1998. Disponível em: <http://homepages.cwi.nl/~arie/papers/dslbib/>. Acesso em março de 2010.

DSL. Disponível em <http://www.martinfowler.com/bliki/DomainSpecificLanguage.html>. Acessado em março De 2010.

EVANS, ERIC. **Domain-Driven Desing: Atacando as Complexidades no Coração do Software.** Rio de Janeiro: AltaBooks, 2009.

FERNANDEZ, OBIE; BAUER, MATT; BLACK, DAVID A.; CASHION, TROTTER; PELLETIER, MATT; SHOWERS, JODI. **Programando Rails: A Bíblia.** Rio de Janeiro: AltaBooks, 2008.

FEST-SWING. Disponível em <http://fest.easytesting.org/wiki/pmwiki.php>. Acessado em: março de 2010.

FOWLER, MARTIN. **Refatoração: Aperfeiçoando o projeto de código existente.** São Paulo: Bookman, 2004.

FREEMAN, ERIC; FREEMAN, ELISABETH; SIERRA, KATHY; BATES, BERT. **Use a Cabeça! Padrões de projetos**. Rio de Janeiro: AltaBooks.

GINGA. Disponível em <http://www.ginga.org.br/>. Acesso em: março de 2010.

GINGA-NCL. Disponível em <http://www.gingancl.org.br/>. Acesso em março de 2010.

GUIMARÃES, RODRIGO LAIOLA; COSTA, ROMUALDO MONTEIRO DE RESENDE; SOARES, LUIZ FERNANDO GOMES. Composer: Ambiente de Autoria de Aplicações Declarativas para TV Digital Interativa. Anais do XIII Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2007), Gramado – RS – Brasil.

LOS. Disponível em <http://luaforge.net/projects/los/>. Acessado em: março de 2010.

LWUIT. Disponível em <https://lwuit.dev.java.net/>. Acesso em: março de 2010.

MARTIN, ROBERT C. **Código Limpo: Habilidades Práticas do Agile Software.** São Paulo: Bookman, 2009.

PROTA, THIAGO MONTEIRO. MoonDo: Um Framework para desenvolvimento de aplicações declarativas no SBTVD. 2010. 63 f. Monografia (Ciência da Computação) – Universidade Federal de Pernambuco, Recife.

RSPEC. Disponível em <http://rspec.info/>. Acessado em Fev. De 2010.

RUIZ, ALEX; PRICE, YVONNE WANG. Test-Driven GUI Development with TestNG and Abbot. IEEE Software May. no 3, p. 51-57, maio/junho 2007.

SOUSA JÚNIOR, PAULO JOSÉ DE. LuaComp: Ferramenta de autoria de aplicações para TV Digital. 2009. 143 f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade de Brasília, Brasília.

TELES, Vinícius M., **Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade**. São Paulo: Novatec, 2004.