UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS APLICADAS A EDUCAÇÃO

DEPARTAMENTO DE CIÊNCIAS EXATAS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GERAÇÃO DE GUI EM RUNTIME A PARTIR DE METAMODELOS

AUTOR PAULO CÉSAR DE LIRA NÓBREGA

Orientador: Prof. Msc. RODRIGO DE ALMEIDA VILAR DE MIRANDA

RIO TINTO - PB

2013

AUTOR PAULO CÉSAR DE LIRA NÓBREGA

GERAÇÃO DE GUI EM RUNTIME A PARTIR DE METAMODELOS

Monografia apresentada para obtenção do título de Bacharel à banca examinadora no Curso de Bacharelado em Sistemas de Informação do Centro de Ciências Aplicadas e Educação (CCAE), Campus IV da Universidade Federal da Paraíba.
Orientador: Prof. Msc. RODRIGO DE ALMEIDA VILAR DE MIRANDA.

RIO TINTO - PB
2013

N754g Nóbrega, Paulo César de Lira.
Geração de GUI em Runtime a partir de Metamodelos / Paulo César de Lira Nóbrega. – Rio Tinto: [s.n.], 2013.
40f.: il. –
Orientador: Rodrigo de Almeida Vilar de Miranda.
Monografia (Graduação) – UFPB/CCAE.

1. Interface – Sistema. 2. GUI. 3. Rendering Patterns. 3. AOM. 4. Web 2.0. I. Título.

UFPB/BS-CCAE CDU: 004.5 (043.2)

AUTOR PAULO CÉSAR DE LIRA NÓBREGA

GERAÇÃO DE GUI EM RUNTIME A PARTIR DE METAMODELOS

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal da Paraíba, Campus IV, como parte dos requisitos necessários para obtenção do grau de BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Assinatura do autor:_____

APROVADO POR:

Orientador: Prof. Msc. Rodrigo de Almeida Vilar de Miranda
Universidade Federal da Paraíba – Campus IV

Prof.ª Dr.ª Ayla Débora Dantas de Souza Rebouças
Universidade Federal da Paraíba – Campus IV

Prof.ª Dr.ª Yuska Paola Costa Aguiar
Universidade Federal da Paraíba – Campus IV

RIO TINTO - PB
2013

“Cada sonho que você deixa pra trás, é um pedaço do seu futuro que deixa de existir.” Steve Jobs

Aos amigos, colegas e professores, minha eterna gratidão por compartilhar comigo seus conhecimentos.

AGRADECIMENTOS

A Deus, pelo discernimento, sabedoria e saúde para enfrentar essa importante fase na minha vida.

A minha mãe Maria Inês exemplo de mulher batalhadora e guerreira, que fez o possível e impossível para que eu pudesse alcançar essa conquista.

A minha esposa Jezabelly Lira, um exemplo de dedicação, que tanto me apoia e ajuda nessa dura caminhada e meu filho que me traz tanta alegria e força para que eu cresça e me dedique cada vez mais.

A meus irmãos pelo incentivo, em especial a Antônio César, que assumiu o papel de pai e que sempre me ajudou no que precisei.

A meus tios, em especial a José Alexander e Maria Lucia, que durante toda a vida me tiveram como um filho e sempre me apoiaram e ajudaram no que precisei, pessoas que tenho um carinho especial.

A meu tio sogro Vicente Silvino e minha tia sogra Maria José, que me acolhem e me ajudaram em um momento muito importante da minha vida, pessoas a quem serei grato por toda a vida.

A meus avós exemplo de pessoas honestas e batalhadoras, em especial Leopoldina Neves de Lira que já se foi mas deixou um enorme saudade.

A Rodrigo De Almeida Vilar De Miranda , pelo seu exemplo de dinamismo e trabalho que é a maior lição que um professor pode dar a seu aluno, gostaria de agradecê-lo por ter acreditado na minha caminhada.

Aos meus amigos e colegas de infância e os que me ajudaram e contribuíram durante minha formação, e em especial a Jefferson Araújo e Diego Sousa, os quais convivemos durante alguns anos e enfrentamos momentos felizes e difíceis juntos.

Aos professores da UFPB, que contribuíram para minha formação, os quais tive a satisfação de ser aluno e aprender diversos ensinamentos que são de extrema importância na minha vida profissional.

E também as pessoas que fugiram da memória, mas que contribuíram com minha formação, mesmo que direta ou indiretamente.

RESUMO

Esse trabalho visa apresentar uma plataforma para geração de GUI de sistemas adaptáveis, por meio do uso das modernas tecnologias que surgiram com a WEB 2.0. Nesse estudo realizamos uma revisão bibliográfica sobre Adaptive Object Model – um padrão arquitetural para se desenvolver sistemas adaptativos, tendo como foco principal os Rendering Patterns. Como não encontramos trabalhos sobre o desenvolvimento de GUIs AOM em sistemas web, decidimos projetar uma solução para esse tipo de interface, aproveitando para utilizar as tecnologias mais modernas no cenário atual. Para a compressão de todos os conceitos abordados nessa solução foi implementado um protótipo funcional da GUI Web 2.0, por meio do uso de tecnologias relativamente maduras com o Javascript, jQuery e Coffeescript. O resultado da união dessas tecnologias com a arquitetura descrita nesse trabalho é uma plataforma que atende aos requisitos de um sistema adaptativo, pois a GUI pode ser modificada em tempo de execução, pelos desenvolvedores, e o resultado é imediatamente disponibilizado para os usuários finais.

Palavras chave: GUI, AOM, Web 2.0 e Rendering Patterns

ABSTRACT

In this work, we have implemented a platform to generate GUI for adaptive systems, using the newest technologies from WEB 2.0. We have conducted a review about Adaptive Object Model – an architectural pattern to adaptive systems – focusing in the Rendering Patterns. Since we did not find papers about AOM GUIs for web systems, we decided to design a solution for this type of interface, using using the latest technologies in the WEB 2.0 scenario. In order to enhance the compression about the concepts covered in this solution, we have implemented a prototype that generates a WEB 2.0 GUI for AOM, using Javascript, jQuery and Coffeescript. The result of uniting this technologies with the architecture described in work is a platform that meets requirements of a adaptive system, because the GUI can be modified at runtime by developers, and the result is immediately available for the end users.

Keywords: GUI, AOM, Web 2.0 e Rendering Patterns

LISTA DE FIGURAS

LISTA DE SIGLAS

AOM	*Adaptive Object Model*
LOM	*Living Object Model*
GUI	*Graphical User Interface*
OO	Orientado a Objetos
CRUD	Create, Read, Update and Delete

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Sistemas adaptáveis possuem a capacidade de se modificar em tempo de execução, para se adequar a novos requisitos com bastante agilidade. Um sistema adaptável não precisa ser recompilado para que alterações feitas sejam implantadas, podendo facilmente acomodar alterações e reagir de forma adequada sem a intervenção de um programador. Para modificar um sistema adaptável não é preciso um especialista em desenvolvimento de sistemas e sim de um especialista no domínio do problema. Alguns exemplos desse tipo de alterações seriam: acréscimo de uma nova informação ou propriedade do sistema, a criação de uma nova regra de negócio ou desconto em um cálculo de imposto.

O mercado de software vem crescendo fortemente a cada ano e a forma de desenvolvimento de software vêm sofrendo uma evolução paralela. Há anos, os paradigmas de programação e a forma de implementação vêm sofrendo alterações devido ao surgimento de novas tecnologias. Esse crescimento tem reflexo direto em várias áreas que englobam o processo de desenvolvimento de software. Os responsáveis pela condução de projeto devem avaliar (i) o custo de desenvolvimento e manutenção do software, (ii) o prazo de entrega do sistema, e (iii) a qualidade do produto final. Sistemas adaptáveis surgem como uma opção muito boa nesse contexto, pois oferecem baixo custo de manutenção, prototipagem rápida e adaptabilidade.

Uma forma de implementar sistemas adaptáveis é usado o padrão arquitetural chamado de AOM (*Adaptive Object Model*), que tem como base o padrão *Type Square* e outros padrões periféricos, trabalhando com uso de metadados para construir sistemas baseados em uma meta-arquitetura. Yoder [1] define sistemas AOM da seguinte forma: "São sistemas com classes, atributos, relacionamentos e comportamentos representados por metadados permitindo assim, sua alteração em tempo de execução, não apenas por programadores mas também por usuários finais".

AOM é uma arquitetura capaz de produzir sistemas adaptáveis, mesmo que fazendo uso de código estático em partes do sistema. Programadores normalmente escrevem código fonte usando alguma linguagem de programação em arquivos textuais e estes são compilados ou interpretados quando o programa é executado. Desse modo, não há como modificar o comportamento do software sem interromper sua execução. Por outro lado, AOM manipula os metadados (do mesmo modo como os sistemas comuns tratam os dados), interpretando-os em

tempo de execução, com a finalidade de implementar a estrutura e o comportamento dos sistemas. Desse modo, os metadados podem ser alterados em tempo de execução e o sistema reagirá imediatamente.

O padrão arquitetural AOM que é composto por vários padrões de projeto menores interligados, oferece suporte para construção de um sistema inteiro desde a camada de *view* até a camada de persistência de dados. O escopo desse trabalho será apenas a camada de GUI, para qual são propostos na literatura *Rendering Patterns*[1] que são um conjunto de três padrões de projeto para a construção de GUI adaptável em AOM: *Property Renderer*, *Entity View* e *Entity-Group View*.

Nesse trabalho o foco é o estudo e desenvolvimento de sistemas de infomação Web adaptáveis e principalmente interfaces para tais sistemas. Nos últimos anos, o desenvolvimento Web tem evoluído bastante, apresentando novos recursos e novas formas de interação com o usuário. Isso se deve principalmente pela transição da Web 1.0 para Web 2.0. Para uns foi apenas uma estratégia de marketing, para outros um grande avanço tecnológico. Na visão da plataforma Web 1.0, como descrito por O'Reilly[3], as páginas eram bastante pobres no que diz respeito à interação com usuário. Não existia a possibilidade de alteração de dados ou até mesmo a possibilidade de se avaliar ou contribuir com o conteúdo da página, características hoje comuns em páginas baseadas na versão 2.0 da Web. Na Web 1.0 só existia o conceito de busca e leitura, os sites se resumiam apenas a informações estáticas. Pereira[5] cita que as páginas podiam ser visitadas, mas não podiam ser modificados pelos leitores, assim não possuíam um atrativo que fizesse o usuário voltar a buscar aquela página.

Com a necessidade de interação surgiu a Web 2.0, alavancada pelas redes sociais, blogs, RSS e etc. Proporcionando cada vez mas dinamismo na experiência do usuário com a Web, novas formas de sistemas interativos vem surgindo e consequentemente trabalhos que dissertam sobre o assunto, com por exemplo as novas ferramentas voltadas para educação citadas em Ribeiro[7], que fala sobre esses novos tipos de sistemas. Até onde sabemos, não foram desenvolvidos trabalhos com AOM para a Web 2.0, portando exploraremos essa lacuna neste trabalho.

1.2 OBJETIVOS

1.2.1 GERAL

Esse trabalho busca desenvolver um renderizador de GUI Web 2.0 100% adaptável, dispondo dos conceitos de meta-arquitetura AOM e sem o uso de código estático, produzindo assim uma GUI totalmente baseada em metadados capaz de se atualizar em tempo de execução.

1.2.2 ESPECÍFICOS

- Analisar os trabalhos existentes sobre implementação de GUI para AOM.
- Definir tecnologias de *front-end* que possam facilitar a implementação dos *rendering patterns* no browser.
- Projetar, implementar e avaliar um protótipo de renderizador de GUI Web 2.0 baseado em AOM.

1.3 METODOLOGIA

Para a alcance dos objetivos desse estudo, as seguintes atividades foram realizadas:

- Revisão bibliográfica sobre AOM e *Rendering patterns*.
- Revisão bibliográfica sobre Web 2.0
- Projeto arquitetural do renderizador de GUI Web 2.0 AOM
- Implementação de um protótipo de renderizador de GUI Web 2.0 AOM

1.4 ESTRUTURA DO TRABALHO

O capítulo I apresenta uma breve introdução sobre todo o conteúdo do trabalho, no capítulo II temos a definição da fundamentação teórica do trabalho, apresentando os principais conceitos utilizados, já no capítulo III temos o detalhamento do projeto da solução desenvolvida, o capítulo IV descreve detalhes sobre a implementação do sistema e por fim o capítulo V contém a conclusão do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo abordaremos os principais temas pesquisados nesse estudo, inicialmente o tema abordado é AOM que serve de base para a plataforma GUI Web 2.0, depois uma explicação detalhada sobre os *Rendering Patterns*, em seguida é detalhando o que é LOM (*Living Object Model*) um plataforma 100% adaptável que depende bastante dos Rendering Patterns.

2.1 AOM

Adaptive *Object Model* (AOM)[2] é um padrão arquitetural para sistemas que demandam alta capacidade de adaptação a novos contextos ou requisitos. Para tanto, AOM propõe que as partes adaptáveis dos sistemas interpretem os metadados em tempo de execução. Por exemplo, as informações de que entidades são armazenadas e quais as suas propriedades podem ser alteradas dinamicamente, de acordo com o estado dos metadados estruturais.

Em AOM, o usuário do sistema pode fazer alterações no sistema sem que seja preciso reconstruí-lo, pois os metadados são armazenados em arquivos XML ou bancos de dados, e podem ser manipulados como se fossem dados operacionais. Desse modo, qualquer alteração feita nos metadados é refletida imediatamente, em todas as camadas do sistema, inclusive para a interface do usuário final, sem a necessidade de recompilar o projeto. Nesse sentido, AOM diferente dos sistemas orientados a objetos, que precisam ser reescritos, compilados e implantados, a fim de acomodar novas funcionalidades.

O AOM possibilita a criação de um sistema com um nível a mais de interação com o usuário, que pode personalizar seu sistema no que tange desde a forma de exibição de sua GUI até as regras que são executadas pela lógica de negócio. AOM permite que a adaptação seja realizada até mesmo por usuários que não tenham conhecimento em desenvolvimento de sistemas, desde que tenham domínio sobre o negócio.

2.1.1 TYPESQUARE

O padrão de projeto chave de AOM é o *Type Square*, que define a estrutura das entidades adaptáveis. O *Type Square* é composto pela associação de dois outros padrões de projeto: *Type Object* e *Property*. O gabarito do *Type Square* pode ser visto na Figura 1.

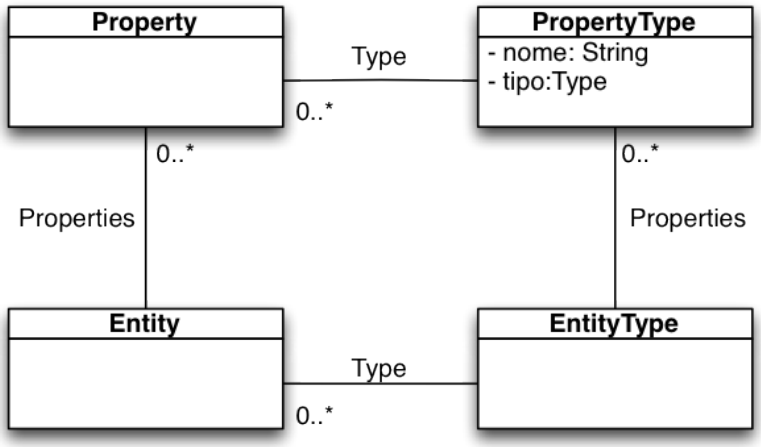

Figura 1 – TYPESQUARE

TYPE OBJECT

A função desse padrão é "desmembrar as instâncias de suas classes, de modo que essas classes possam ser construídas como instâncias de outra classe", [8]. Ele determina que as classes tenham tipos, ou seja, que cada objeto (Class) esteja associado ao tipo de objeto (ClassType), conforme mostra a Figura 2.

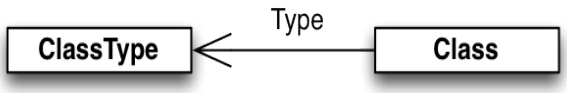

Figura 2 - TYPE OBJECT

PROPERTY

Esse padrão trata as propriedades dinâmicas dos objetos. Nas linguagens com suporte a orientação a objetos como Java, o comum é encontrar em uma classe contendo propriedades com o tipo definido. Por outro lado a proposta do padrão *Property* é que essas propriedades sejam transformadas em instâncias de uma classe genérica *Property*. Nome, idade e cpf são instancias do *Property* e cada uma desses campos possui um *PropertyType* como mostrado na Figura 3.

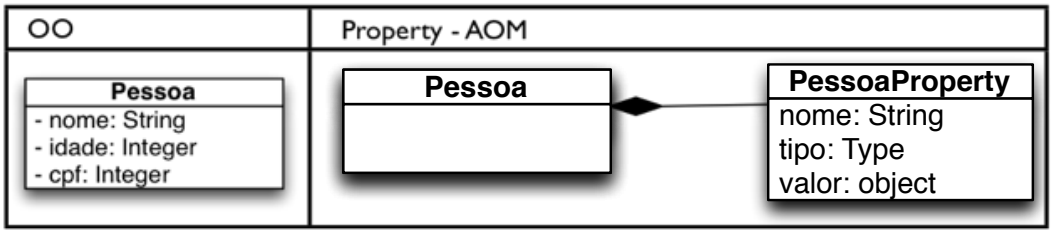

Figura 3 - PROPERTY

Esses dois padrões se relacionam de forma a constituir o *Type Square*, definindo que o tipo se relacione com as propriedades, assim todo *Type Object* tem um conjunto de *Property*. O *Type Object* também se aplica às propriedades, pois no AOM elas passam a ser objetos com instancias de propriedades.

2.2 RENDERING PATTERNS

Para que se torne possível a construção e renderização de GUI de sistemas baseados em AOM, Welicki[1] sugere a utilização de três padrões de projeto (Rendering Patterns), para a renderização de GUI adaptável em AOM[1]. Os padrões são: Property Renderer, responsável por renderizar as propriedades das entidades; Entity View, que tem a função de renderizar um ou vários Property Renderer de uma única entidade, definido seu modo de exibição na GUI; e o Entity-Group View, que é responsável por renderizar uma coleção de Entity Views, definindo a forma como os mesmo serão apresentados na GUI, sendo assim, cada um tem sua função no trabalho da construção da GUI. Sendo assim, cada um desses padrões tem sua função e seu escopo na construção da GUI adaptável AOM.

2.2.1 PROPERTY RENDERER

Esse padrão de projeto é responsável por definir como cada propriedade de uma entidade deve ser apresentada na GUI. Usualmente, existe um PropertyRenderer para cada tipo de Property Type do Type Square. Por exemplo, um mesmo StringPropertyRenderer pode ser reutilizado para desenhar todas as Property Types que sejam String, como nome, e-mail, descrição, etc.

O Property Renderer utiliza metadados para remover o acoplamento que é comumente visto em GUIs, onde o código de renderização de propriedades é estático e possui muita repetição.

Outro aspecto importante do Property Renderer é que esse padrão demanda o uso de mais metadados do que os disponíveis no Type Square. Definimos esses metadados extra como “metadados de GUI”. Por exemplo, o nome de um cliente é do tipo String e deve ser desenhado como uma caixa de texto, mas o sexo do cliente, que também é String, deve ser desenhado como um combo box com os valores “Masculino “ e “Feminino”. Dado que o

Type Square sabe apenas que os campos são String, são os metadados de GUI os responsáveis por definir qual widget será utilizado na GUI de cada propriedade. Outros exemplos de metadados de GUI são: a propriedade é visível e/ou editável, validação, posicionamento, cores e fontes.

A seguir, são exibidos alguns cenários de uso do Property Renderer.

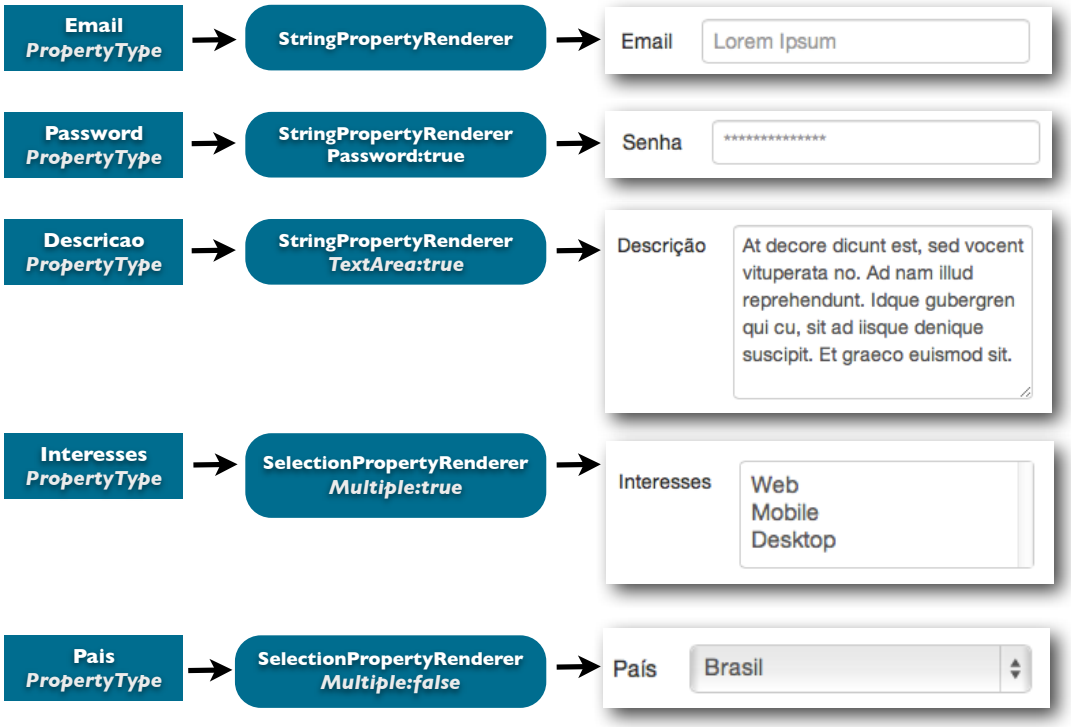

Figura 4 - StringPropertyRenderer

No exemplo mostrado na Figura 4, temos várias demonstrações de uso do *Property Renderer,* que tem a função de renderizar propriedades String utilizando as classes StringPropertyRender e SelectionPropertyRenderer. Para o sistema, a forma de se renderizar propriedades são as mesmas, informando e alterando apenas valores do renderizador é possível definir como o mesmo vai renderizar a propriedade na GUI.

Na Figura 5 temos o exemplo do FilePropertyRenderer, sendo renderizado de duas formas diferentes:

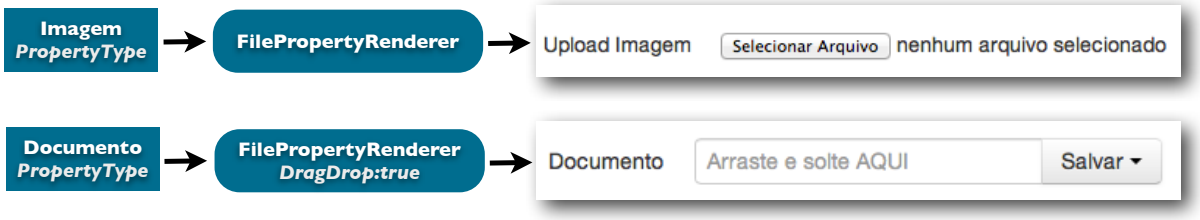

Figura 5 - FilePropertyRenderer

Nesta imagem temos o renderizador de arquivos definido como FilePropertyRenderer, que tem como função desenhar na GUI de forma diferente uma propriedade do tipo File.

Segue na Figura 6 um exemplo de uso do renderizador BooleanPropertyRenderer:

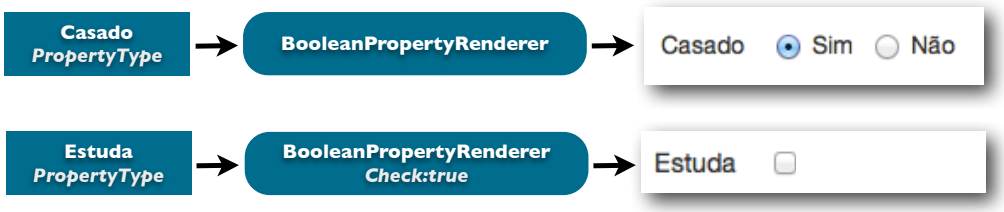

Figura 6 - BooleanPropertyRenderer

Neste ultimo exemplo é apresentado um renderizador de propriedades do tipo boolean, definido como BooleanPropertyRenderer, que possibilita a inserção de formas diferentes na GUI propriedades do tipo boolean.

2.2.2 ENTITYVIEW

Esse padrão busca englobar a nível de entidade o uso do padrão anterior, que cria renderizadores para as propriedades de uma entidade. O *Entity View* coordena várias propriedades da entidade, definindo uma relação do tipo composição, onde vários *Property Renderers* compõem uma *Entity View,* o padrão descrito nesse tópico gera uma saída que pode ser desde um fragmento de tela até mesmo uma tela inteira. Segue na Figura 7 uma demonstração do uso deste padrão:

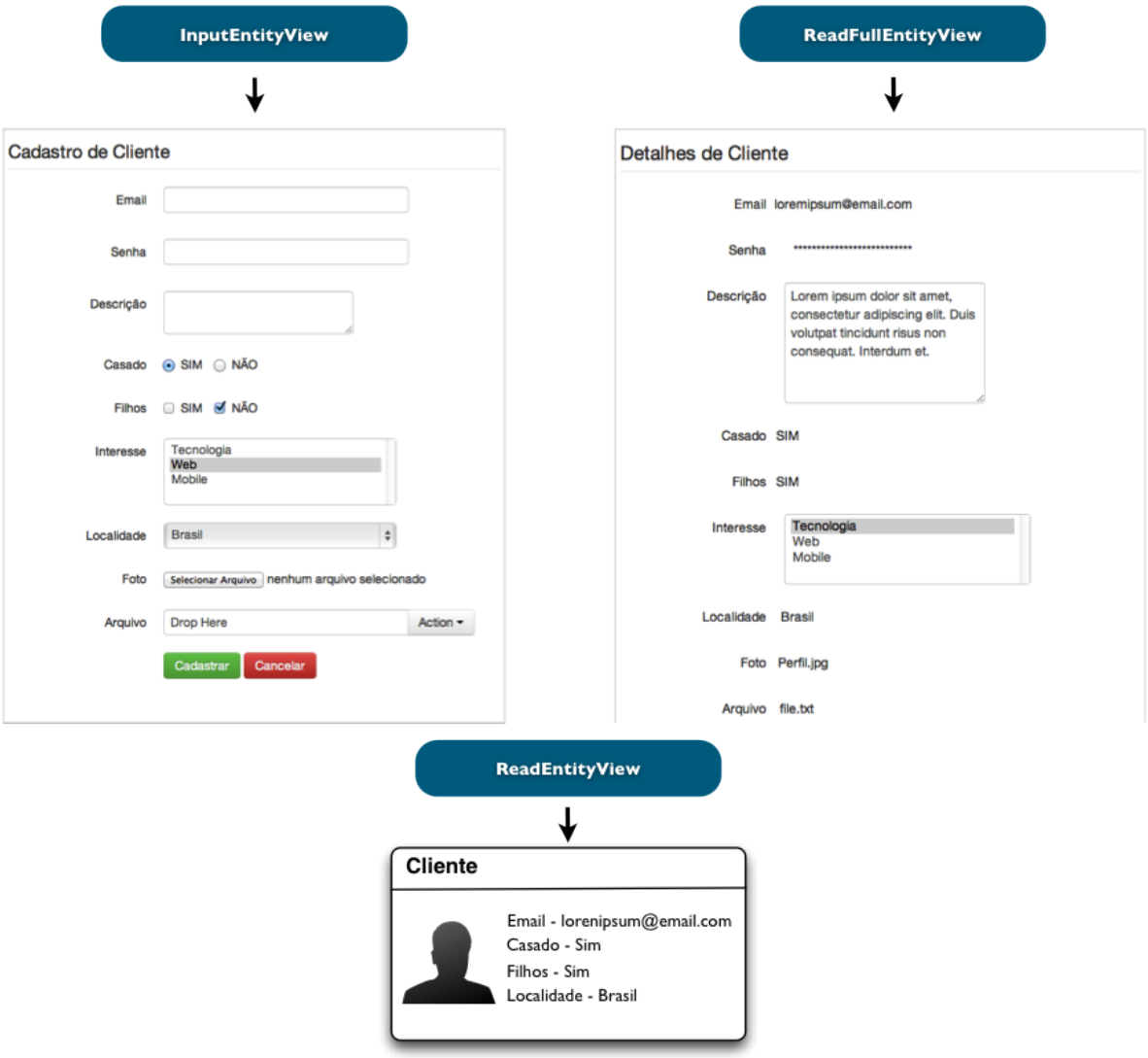

Figura 7 - Entity View

Neste exemplo, mostrado na Figura 7, temos um cadastro de clientes. Para exemplificar o uso do padrão *Entity View*, foram criado três tipos de *Entity View*: uma para o contexto de inserção de clientes (InputEntityView) outro para o contexto de visualização detalhada de cliente (ReadFullEntityView) e por fim o terceiro que mostra um cliente de forma resumida (ReadEntityView).

2.2.3 ENTITY-GROUP VIEW

Define uma classe que controla uma coleção de entidades para que sejam ajustadas a algum contexto. O *Entity-Group View* é um tanto mais complexo que os demais padrões, consequentemente o mais poderoso, pois tem a capacidade de inserir um grupo de entidades onde cada entidade pode conter uma ou varias propriedades, em um determinado contexto. A Figura 8 mostra devidamente o uso do *Entity-Group View* :

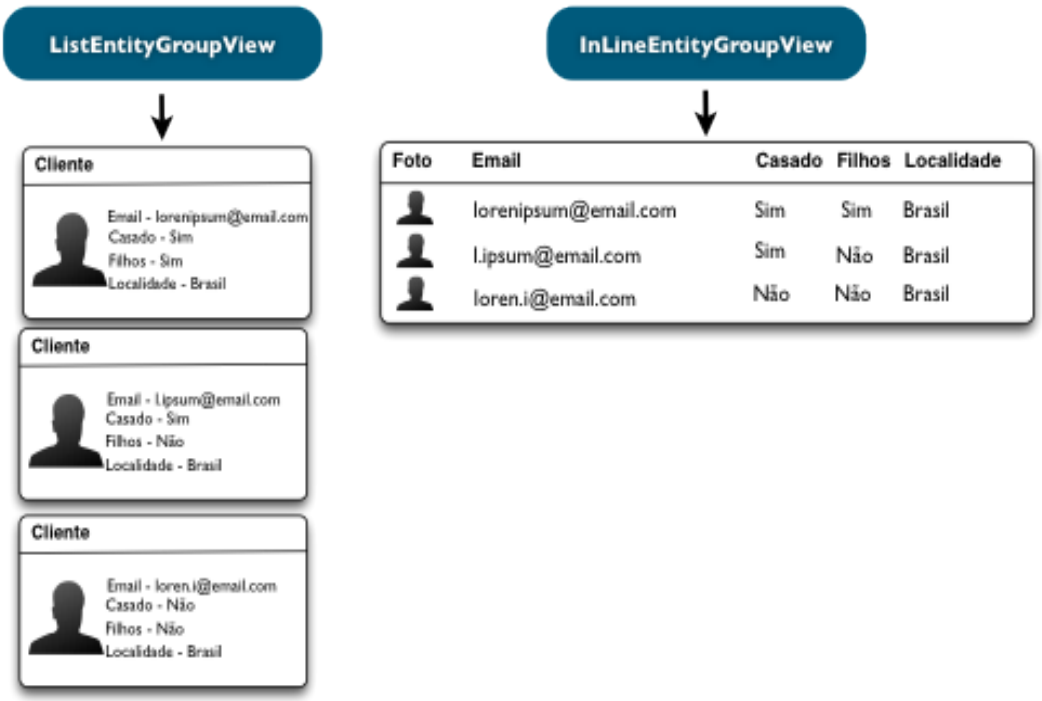

Figura 8 - Entity-Group View

Neste exemplo da Figura 8, são apresentados dois tipos de *Entity-Group View* onde, o ListEntityGroupView define um contexto de listagem de cliente e o renderiza na GUI, exibindo uma lista de clientes em forma de cartão e temos o InLineEntityGroupView que renderiza a mesma lista de clientes em um contexto de listagem em forma de tabela.

2.3 LOM

O padrão AOM gera sistemas híbridos, onde parte de suas funcionalidade são adaptáveis, mas o restante do código é implementado como um sistema OO comum [13]. A plataforma para sistemas 100% adaptáveis *Living Object Model* (LOM) utiliza os mesmo padrões de projeto de AOM, aplicando-os em 100% do sistema. LOM visa a criação de um sistema 100% adaptável, ou seja, um sistema que não possui código estático e que seja completamente capaz de em tempo de execução se adaptar a qualquer contexto sem a necessidade de escrever, compilar e implantar o novo código. A principal diferença entre AOM e LOM está no uso do padrão *Type Square*. Em AOM, cada trecho adaptável do código utiliza a sua própria instanciação do *Type Square*. No LOM existe apenas um *Type Square* que gerencia os dados de todas as entidades que são armazenadas no sistema. Como o *Type Square* original lida apenas com entidades e propriedades, nós adicionamos o padrão *Accountability*, para representar os relacionamentos entre objetos de entidades distintas. Além disso, decidimos renomear alguns elementos do *Type Square*, para evidenciar o seu uso diferente no LOM. O projeto resultante pode ser visto na Figura 9. Os padrões de renderização são ainda mais importantes para o LOM, pois se tornam responsáveis por desenhar toda a interface dos sistemas.

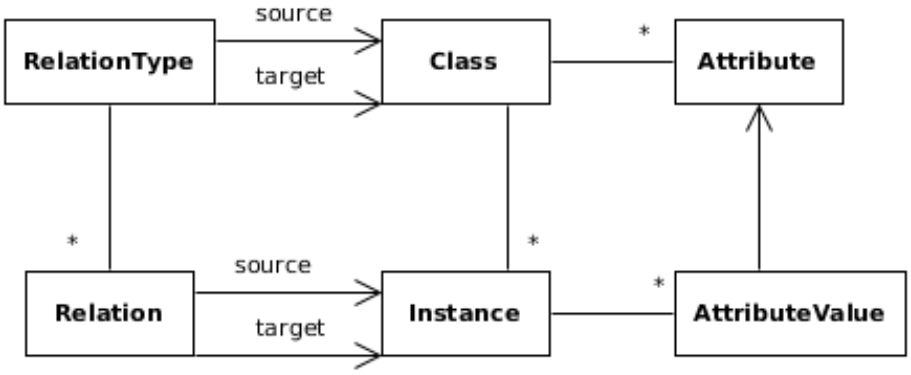

Figura 9 – LOM (*Living Object Model*)

3 INTERFACES WEB 2.0 PARA SISTEMAS AOM

Nesse capítulo inicialmente definiremos o problema em que se concentrou este trabalho de conclusão de curso. Logo após, detalhamos a análise e projeto de uma plataforma para construção de sistemas Web 2.0 com GUI's adaptáveis construídas utilizando AOM, detalhando os casos de uso da plataforma. Posteriormente veremos a arquitetura e projeto detalhado da solução.

3.1 DEFINIÇÃO DO PROBLEMA

Na revisão bibliográfica do capítulo anterior, não encontramos trabalhos sobre a geração de GUI para web em sistemas AOM. De fato, decidimos implementar uma GUI AOM utilizando as tecnologias mais modernas da web 2.0. No entanto, como os *Rendering Patterns* serão implementados no browser, é preciso achar uma linguagem que tenha boa abstração de OO, o que não é o caso de JavaScript.

No restante deste capítulo, detalharemos a solução proposta para o problema acima descrito

3.2 CASOS DE USO

A plataforma desenvolvida atende a dois tipos de usuário: O desenvolvedor (*Developer*), que define dinamicamente os *widgets* que renderizarão a interface do sistema AOM; e o usuário final (*End User)*, que utiliza o sistema com interface similar a dos sistemas estáticos (não AOM). Obviamente a interface para o usuário final não é estática, pois as alterações feitas pelo desenvolvedor (*Developer*) são refletidas em tempo de execução para o usuário final. A Figura 10 descreve os casos de uso da solução.

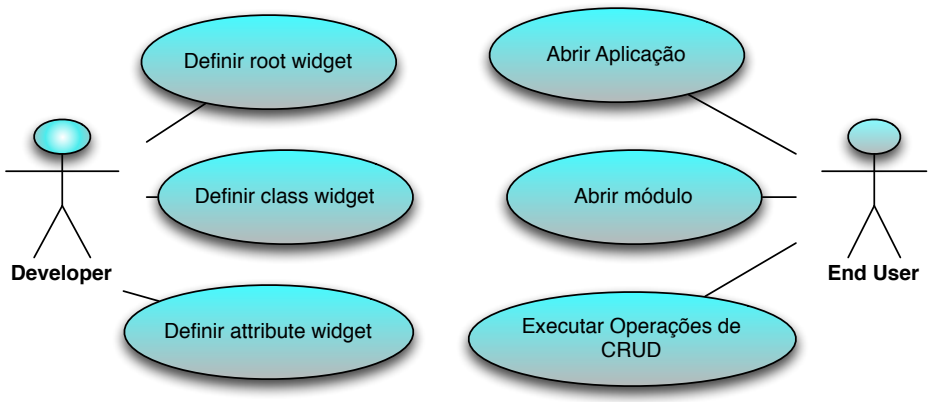

Figura 10 – Casos de Uso

Esses casos de uso serão detalhados na seção 3.4

3.3 ARQUITETURA

Com a finalidade de atender às funcionalidades representadas pelos casos de uso ilustrados pela Figura 10 e ao requisito não funcional de ser um sistema web 2.0, definimos a arquitetura que pode ser vista na Figura 11.

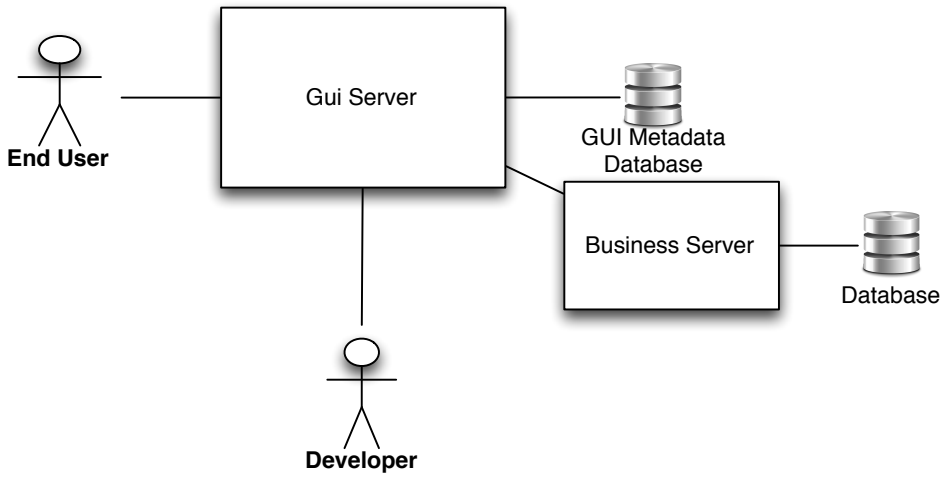

Figura 11 – Arquitetura da Solução

Definimos dois tipos de atores no sistema. Primeiramente, o desenvolvedor (*Developer*) que é responsável por criar componentes de renderização GUI (*widgets*) e definir quais serão utilizados para cada entidade e propriedade do sistema baseado em AOM. Esses atores utilizam browsers para acessar o sistema. O código de renderização da GUI e acesso aos dados é baixado do servidor para o browser quando o usuário abre uma sessão web. Todavia, esse código pode ser atualizado no meio da sessão, pois essa arquitetura permite a adaptação em tempo de execução. O servidor é dividido em duas partes: o Business Server, que trata dos metadados e dados da lógica de negócio, utilizando seu próprio banco de dados adaptativo (*database*); e o GUI Server, que utiliza um banco de metadados de GUI para compor os *widgets* que desenharão a interface do cliente e os envia para o browser. Os dois aspectos mais importantes para a arquitetura da solução são o uso de metadados e a interface Web 2.0.

3.3.1 METADADOS

Em sistemas AOM os metadados são a chave para a construção de um sistema adaptável em tempo de execução, pois as entidades, atributos, associações e regras de negócio são armazenados em banco de dados, juntamente com os dados operacionais.

Os metadados são recuperados e interpretados em tempo de execução, assim qualquer alteração desses metadados é refletida imediatamente no sistema, sem que seja necessário reconstruí-lo.

A nossa solução divide os metadados em dois grupos: metadados de lógica de negócio, que são as entidades, atributos, relacionamentos e regra de negócio do sistema; e metadados de GUI, que definem os *widgets* que serão responsáveis por desenhar o arcabouço da GUI, desenhar a interfaces das classes e dos atributos.

3.3.2 WEB 2.0

A tendência na Web 2.0 é que a GUI deixe de ser renderzada no servidor de aplicação e passe a ser construída no browser, através da manipulação do DOM (Modelo de Objetos de Documentos HTML) com JavaScript. Essa característica, aliada com AJAX[9], permite que os sistemas Web melhorem sua usabilidade, atualizando pedaços da página em vez de recarregá-la completamente em cada iteração com o usuário.

Nesse contexto, o servidor fica responsável por fornecer apenas os dados do sistema. As páginas HTML são montadas pelo código JavaScript estático, que é baixado previamente. Assim sendo, o servidor passa a ser utilizado como um Webservice, cujo conceito é fornecer dados ao invés de páginas sobre o protocolo HTTP.

Todavia em AOM o código JavaScript não pode ser estático pois o desenvolvedor (Developer) a qualquer momento pode trocar ou adicionar *widgets* no sistema. Desse modo criamos o componente chamado GUI Server que é responsável por manter os metadados de GUI e enviar dinamicamente código JavaScript para o browser.

3.4 DIAGRAMAS DE SEQUÊNCIA

Esse seção detalha o projeto das funcionalidades dos casos de uso mostrados na Figura 13, dividido em duas sub seções que são relativas aos atores o desenvolvedor (*Developer*) e usuário final (*End User*).

3.4.1 DEVELOPER

O diagrama da Figura 12, demonstra o fluxo dos casos de uso que possui o desenvolvedor (*Developer*) como ator do cenário. Esses casos de uso são focados na construção dos *widgets* usados no sistema e todos são implementados pelo método `defineWidget(),` variando apenas o tipo de *widget* criado (parâmetro *type*).

A seguir descrevemos os parâmetros de cada caso de uso.

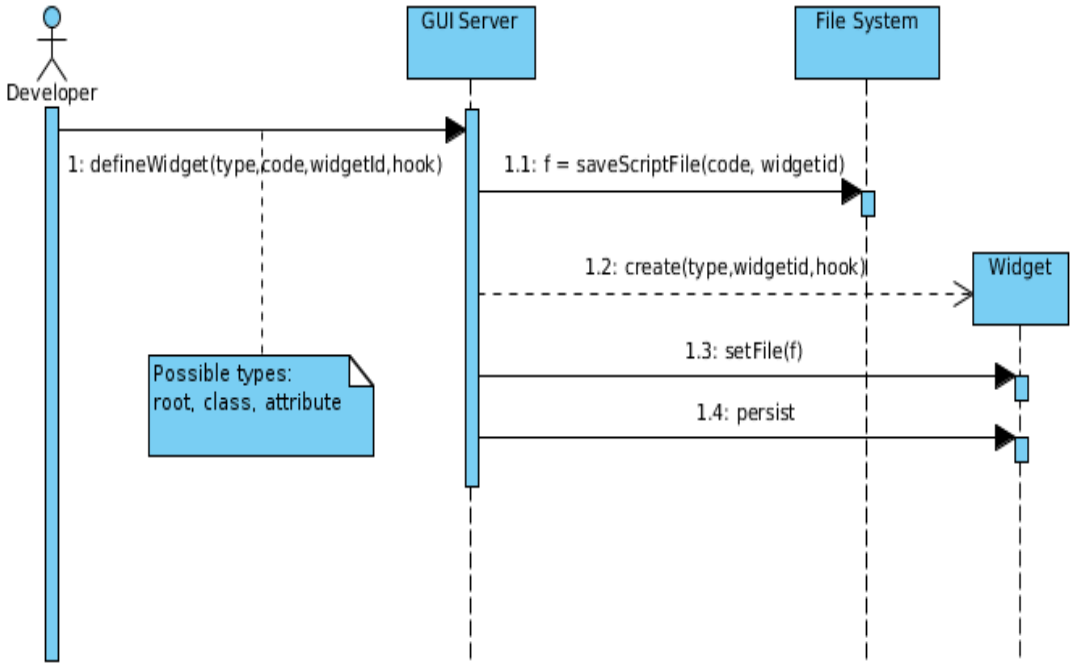

Figura 12 - Diagrama de Sequência Developer

3.4.1.1 DEFINIR ROOT WIDGET

Esse caso de uso descreve a ação de definir o root *widget* no sistema, efetuada pelo desenvolvedor (*Developer*). Esse *widget* é de extrema importância para o sistema, pois ele é responsável por desenhar a interface inicial da aplicação e define os ganchos (*hooks*) para a inserção dos demais *widgets*. O método `defineWidget`() recebe quatro parâmetros ao ser invocado, que são *type* que nesse caso de uso é *root*, *code* que é o código em JavaScript do *widget*; o widgetId o id único de cada *widget* inserido; e *hook*, que define o local onde o *widget* será inserido – no caso do root *widget* o *hook* é nulo.

3.4.1.2 DEFINIR CLASS WIDGET

Nesse caso de uso, o *Developer* define os *widgets* que desenharão as telas das classes ou entidades, os *Entity Views*. Os parâmetros do método `defineWidget`() assumirão os seguintes valores: *type*, *Class*; *code* e widgetId, código JavaScript e id do *widget* de classe; e *hook*, alguns dos ganchos definidos pelo root *widget* para as Classes, por exemplo, Tela de listagem e formulário de edição.

3.4.1.3 DEFINIR ATTRIBUTE WIDGET

O *Developer* usa esta funcionalidade para definir quais os *Property Renderers* de cada atributo ou propriedade. Os parâmetros do método `defineWidget`() assumirão os seguintes valores: *type*, *Attribute*; code e widgetId, código JavaScript e id do *Property renderer*; e *hook*, alguns dos ganchos definidos pelo *widget* de classe para as Propriedades.

3.4.2 END USER

O diagrama da Figura 13 detalha o fluxo da experiência de uso do usuário com o sistema. O foco nesse caso é mostrar o uso da plataforma como um sistema comum. No entanto, esse uso é completamente vinculado às funcionalidades explanadas na seção anterior, pois a GUI será renderizada pelos *widgets* definidos acima.

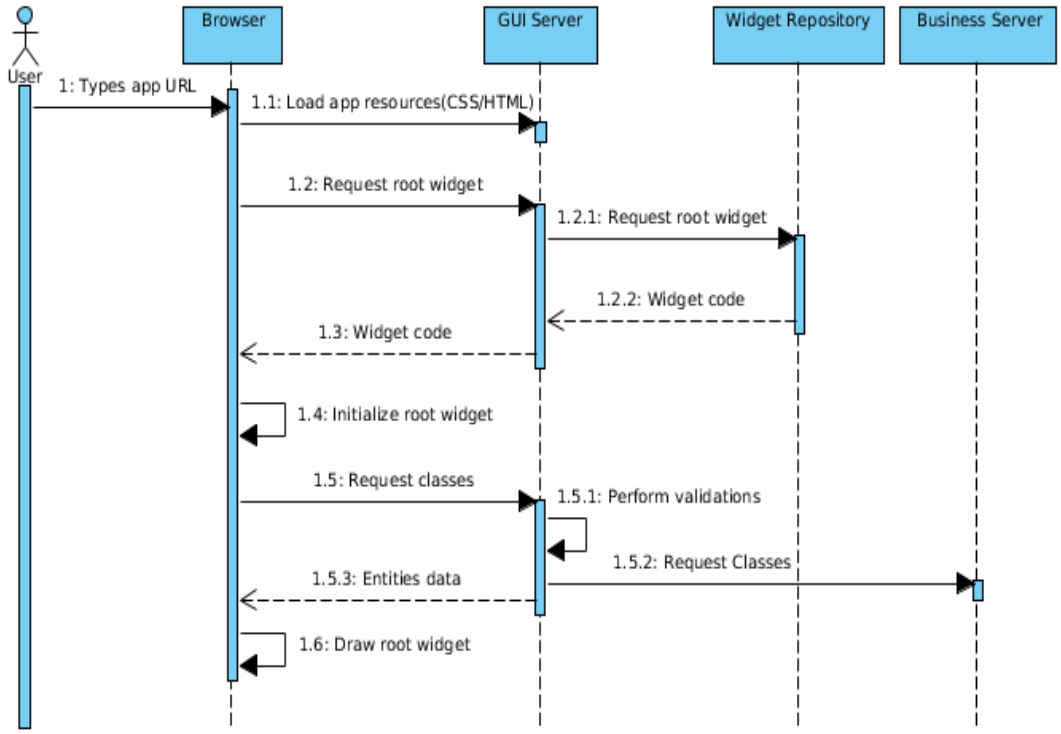

Figura 13 - Diagrama de Sequência User

3.4.2.1 ABRIR APLICAÇÃO

Esse caso de uso representa o acesso inicial à aplicação, o usuário insere a URL e a página inicial é renderizada no browser por meio do *root widget* que é o responsável por construir a primeira página a ser exibida ao usuário como representado na Figura 13. Pode-se ver que a primeira interação do usuário com a plataforma envolve alguns passos. Inicialmente, todo o código estático e o framework da GUI é baixado para o browser. Posteriormente, o browser requisita o código do *root widget*. Após baixar e iniciar o *root widget*, o browser se comunica novamente com o servidor para obter os dados que serão exibidos na GUI. Nesse caso, os dados correspondem as classes que estão cadastradas no *Business Server*. Só depois de todos esses passos é que o root *widget* pode executar o código de renderização da interface.

3.4.2.2 ABRIR MÓDULO

O caso de uso Abrir Módulo é executado quando o usuário seleciona alguma das classes desenhadas pelo root *widget*. Daí, a plataforma deve desenhar a GUI relativa à classe selecionada, para que o usuário possa executar as operações de CRUD. Ao abrir o módulo, o browser requisita o código do *widget* de classe (*Entity View*). Quando ele é baixado e inicializado, o browser requisita os dados a serem exibidos. Somente após esses passos, o *widget* efetivamente renderiza a GUI. Esse caso de uso é bem parecido com o anterior (Figura 13), diferindo apenas pelo fato que que o código estático não precisa ser baixado novamente.

3.4.2.3 EXECUTAR OPERAÇÕES DE CRUD

Este caso de uso representa as ações básicas do usuário dentro de algum módulo, como listar, inserir, atualizar e remover algum dado. Nesse momento, a plataforma deve ser capaz de desenhar a GUI dos atributos da classe referente ao módulo em execução. Nesse contexto, o browser requisita o código dos *widget* de atributo (*Property Renderers*). Quando eles são baixados e inicializados, o browser requisita os dados a serem exibidos. Somente após esses passos, os *widgets* efetivamente renderizam a GUI.

4 IMPLEMENTAÇÃO

Neste capítulo são abordados os detalhes da implementação, a descrição das tecnologias envolvidas e algumas telas de protótipo da plataforma proposta.

4.1 TECNOLOGIAS

Aqui se encontra uma breve descrição sobre as principais tecnologias aplicadas no desenvolvimento desse trabalho, que tem como tecnologia base o JavaScript, que é o padrão de fato para a execução de scripts no browser.

No entanto, orientação a objetos com JavaScript não é uma tarefa trivial, ou simples de se implementar. Devido a essa dificuldade surgiram linguagens como CoffeScript[11] e TypeScript[12] que têm como objetivo facilitar a construção de sistemas Orientados a Objetos com JavaScript.

Como visto no capítulo anterior os *Rendering Patterns* fazem uso dos conceitos de OO[14] para implementar a GUI adaptável. Esses conceitos facilitam a reutilização de código, a extensão do sistema e agilizam o desenvolvimento de sistemas.

Portanto, se tornou necessária a escolha de uma plataforma que facilitasse a implementação dos *Rendering Patterns*. Nesse contexto, pesquisamos três tipos de tecnologias: frameworks baseados em JS, como jQuery[18], Backbone.js[19], SproutCore[20], Spine[21] e AngularJs[22]; outra linguagem OO executada no browser (DART)[19]; e uma linguagem OO que é compilada para JavaScript (Coffeescript).

jQuery se mostrou interessante, mas ainda não dá um aspecto totalmente OO ao código. Os outros frameworks já possuem um modelo MVC para interação entre seus objetos, que poderia conflitar com o projeto dos *rendering patterns*. DART fornece um bom modelo OO, mas sua execução requer a instalação de um plug-in no browser, o que pode restringir a aplicabilidade da solução.

A solução mais interessante para o construção da GUI adaptável foi o uso da linguagem de programação CoffeeScript, que possui uma boa API orientada a objetos e quando é compilada gera código Javascript.

4.1.1 JAVASCRIPT

Uma linguagem de programação interpretada, desenvolvida em setembro de 1995, para ser usada por browsers Web, para facilitar a execução de scripts no lado cliente da aplicação (client side). Desse modo, comportamentos são definidos em scripts e executados sem a necessidade de serem requisitados a um servidor, diminuindo a sobrecarga em servidores e melhorando o desempenho de serviços Web. Inicialmente, ela não foi bem aceita por desenvolvedores. Mas com a criação do Ajax (Asynchronous JavaScript and XML), a linguagem ganhou força, e vem crescendo e se popularizando exponencialmente. Atualmente a linguagem é utilizada até no lado servidor, através de tecnologias como Node.JS[17]. Uma enorme quantidade de frameworks e bibliotecas vêm aparecendo constantemente para facilitar o desenvolvimento com essa linguagem. Alguns desses frameworks estão sendo utilizados na aplicação que serve de base para esse estudo, e serão descritos a seguir.

4.1.2 JQUERY

Uma biblioteca desenvolvida para facilitar o uso do JavaScript, que dispõe de vários componentes personalizáveis, validadores e funções JavaScript simplificadas. É uma excelente forma de se trabalhar com JavaScript. Além de facilitar o uso do mesmo ela também proporciona uma menor curva de aprendizado.

4.1.3 COFFEESCRIPT

Criada para facilitar a vida dos desenvolvedores, CoffeeScript é uma linguagem que quando compilada gera código JavaScript. Ela torna desnecessário o uso dos vários parênteses e chaves que existem no JavaScript. Ela simplifica a orientação a objetos com JavaScript tornando seu uso bem mais simples.

4.2 DIAGRAMA DE CLASSES

A Figura 14 representa o diagrama de classes de uma parte da plataforma para renderização de GUIs AOM Web 2.0. As classes do Rendering Patterns – EntityGroupView, EntityView e PropertyRenderer – são extendidas para criar os widgets concretos da GUI: UlRootWidget, TableInstanceListing e

BulletInstanceListing. O uso desses widgets concretos será demonstrado na próxima seção.

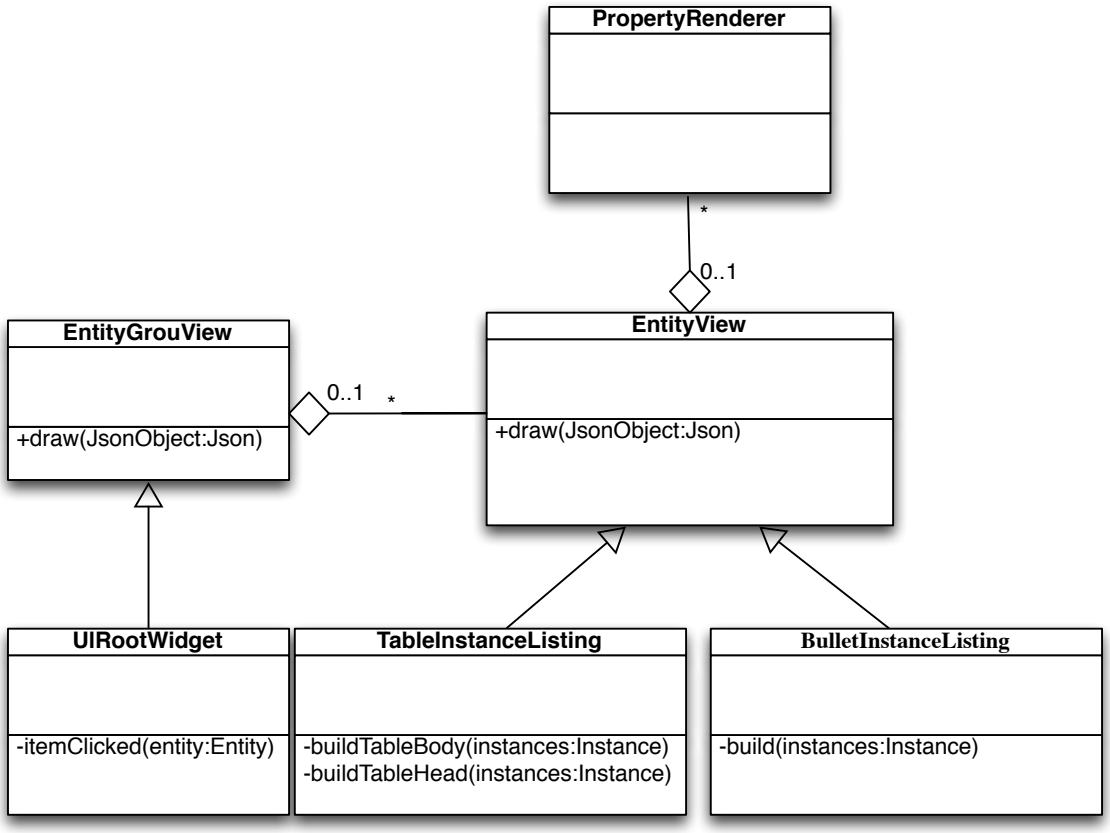

Figura 14 - Diagrama de Classes

4.3 TELAS

Nesta seção são apresentados algumas imagens do protótipo da plataforma GUI Web 2.0. A Figura 15 apresenta a tela inicial do sistema, construída pelo *root widget* UlRootWidget representado na sessão 3.4.2.1.

Figura 15 - Root Widget

A Figura 16, mostra a listagem da instancias dos objetos interpretados pelo sistema. A imagem descreve a lista de Entidades e as instancias dos objetos ligados a elas. Inicialmente foi selecionada a entidade Cliente que possui as propriedades: Nome (de tipo String), CPF (do tipo String) e Data (do tipo Timer). Essa tela foi renderizada pelo *widget* TableInstanceLinting.

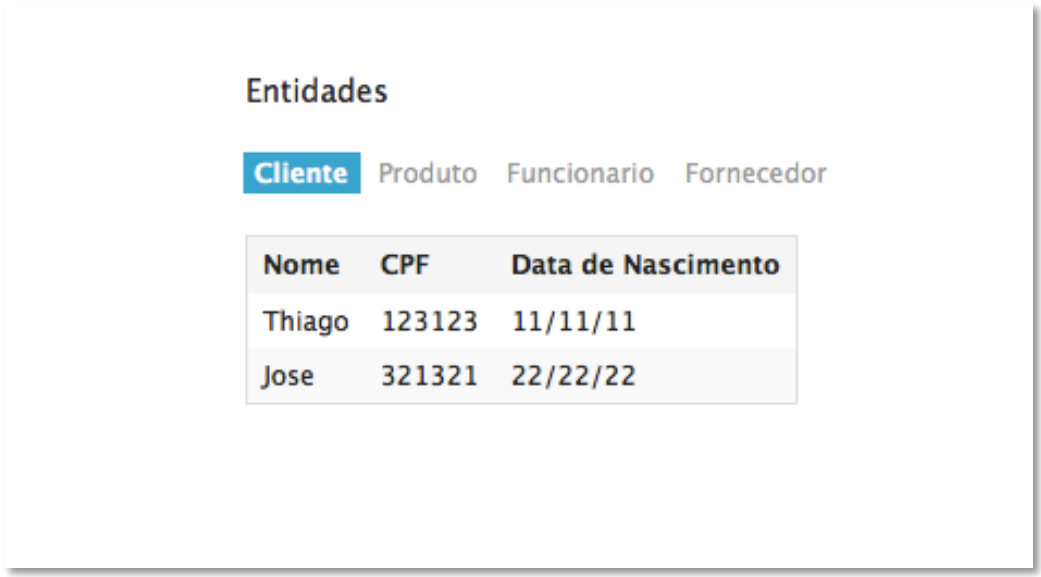

Figura 16 - Classe Cliente

A Figura 17 mostra a seleção da entidade produto que possui como propriedades: Nome (do tipo String), Preço (do tipo double) e Estoque (do tipo Integer). Como o sistema é baseado em Ajax, ao selecionar uma entidade diferente, apenas a tabela de instâncias é atualizada, dispensado a necessidade de atualizar a página inteira. Esse tela foi renderizada pelo *widget* BulletInstanceListing, com a opção *check* = *true*.

Figura 17 - Classe Produto

A Figura 18 mostra a seleção de Funcionário que possui como propriedades: Nome (do tipo String), Função (do tipo String) e Salário (do tipo double). Esse tela foi renderizada pelo widget BulletInstanceListing, com a opção check = false.

Figura 18 - Classe Funcionário

Por fim, a Figura 19, que representa a seleção da entidade Fornecedor e possui as propriedades Nome (do tipo String), CNPJ (do tipo String) e contato (do tipo String), Essa tela foi renderizada pelo widget TableInstanceLinsting.

Figura 19 - Classe Fornecedor

5 CONCLUSÃO

O presente trabalho se propôs a desenvolver uma plataforma para renderização de GUIs AOM utilizando tecnologias da Web 2.0, uma vez que na nossa revisão bibliográfica não encontramos trabalhos que tratassem de AOM para sistemas Web.

Como principais contribuições desse trabalho, podemos destacar o projeto de alto e baixo nível de uma solução capaz de utilizar uma arquitetura Web 2.0 para renderizar interfaces adaptáveis.

Esse projeto foi validado através de um protótipo funcional que foi capaz de utilizar os rendering patterns para desenhar a GUI com sucesso, utilizando metadados de lógica e metadados de GUI.

Como sugestão de trabalhos futuros temos a pesquisa detalhada sobre impacto econômico no custo final da construção de sistemas de informação ocasionado pelo uso AOM, pesquisa sobre o grau de escalabilidade alcançado com o uso do AOM e por fim o desenvolvimento do sistema GUI Web 2.0 até chegar a um nível que possa ser comercializado

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Federico B; J.W. Yoder "How to Builder Systems that can Dynamically Adapt to new Busness Requerements," OOPSLA 2001.

[2] L. Welicki, J. W. Yoder, and R. Wirfs-Brock, "Rendering patterns for adaptive object-models," in *Proceedings of the 14th Conference on Pattern Languages of Programs*, ser. PLOP '07. New York, NY, USA: ACM, 2007, pp. 12:1–12:12. [Online]. Available: http://doi.acm.org/10.1145/1772070.1772085

[3] T. O'Reilly, "Desing Patterns and Business Modesl for the Next Generation of Software," 30-09-2005.

[4] P. Megumi M; F. F. Correia, J.W. Yoder, E. Guerra, Hugo S. F; A. Aguiar "AOM Metadadata Extendion Points,"

[5] Clara Pereira. C; J. B. Bottentuit, "Comunicação Educacional: do modelo unidireccional para a comunicação multidirecional na sociedade do conhecimento" Univesidade do Minho, 2008.

[6] Jesse J. G. "Ajax: A New Approach to Web Applications". Adaptivepath.com/ideas/ajax-new-approach-Web-applications, 18 Fevereiro 2005.

[7] Adriano Carlos. R; Cláudio H. S; "A contribuição da Web 2.0 nos sistemas de educação online" Uni-FACEF 2008.

[8] Jeremy M. Wolfe. "What Can 1 Million Trials Tell Us About Visual Search?" VOL. 9, NO. 1, JANUARY 1998.

[9] W3Schools; "Ajax Introduction", http://www.w3schools.com/ajax/ajax_intro.asp.

[10] http://pt.wikipedia.org/wiki/JavaScript

[11] http://coffeescript.org/

[12] http://www.typescriptlang.org/

[13] http://pt.wikipedia.org/wiki/Orientação_a_objetos

[14] Brett D. McLaughlin, Gary Pollice, Dave West; Head First Object-Oriented Analysis and Design, 4 Dezembro, 2006.

[15] Hélio Engholm Jr., "Engenharia de Software na prática"; p 31. Novatec, ISBN : 8575222171

[16] Alexandre S.,"Soa Aplicado: Integrando com Web services e além" , Casa do Código.

[17] http://nodejs.org.

[18] http://api.jquery.com

[19] http://backbonejs.org

[20] http://guides.sproutcore.com

[21] http://spinejs.com/docs/index

[22] http://angularjs.org

APÊNDICE

• A URL de acesso ao protótipo encontra-se no link: https://lom.herokuapp.com/

• O código fonte do protótipo desenvolvido encontra-se no link: https://github.com/PauloLira/lom