Natan Vinícius Zeferino

UMA ABORDAGEM DIRIGIDA A MODELOS PARA GERAÇÃO DE INTERFACES A PARTIR DE DIAGRAMAS DE INTERAÇÃO COM O USUÁRIO

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.
Orientadora: Prof. Dr. Patrícia Vilain

Florianópolis
2015

Ficha de identificação da obra, elaborada pelo autor através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Zeferino, Natan Vinícius

Uma abordagem dirigida a modelos para geração de interfaces a partir de diagramas de interação com o usuário / Natan Vinícius Zeferino; orientadora, Patrícia Vilain – Florianópolis, SC, 2015 – 104 p.

Dissertação (mestrado) – Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós Graduação em Ciência da Computação.

Inclui Referências

1. Ciência da Computação. 2. Engenharia de Software. 3. Desenvolvimento Dirigido a Modelos (MDD). 4. Diagrama de interação com o Usuário (UID). 5. Geração automática de interfaces. I. Vilain, Patrícia. II. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Ciência da Computação. III. Título.

Natan Vinícius Zeferino

UMA ABORDAGEM DIRIGIDA A MODELOS PARA GERAÇÃO DE INTERFACES A PARTIR DE DIAGRAMAS DE INTERAÇÃO COM O USUÁRIO

Esta Dissertação foi julgada adequada para obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação

Florianópolis, 5 de março de 2015.

_____
Prof., Dr. Ronaldo dos Santos Mello
Coordenador do Programa

Banca Examinadora:

_____
Prof.ª, Dr.ª Patrícia Vilain
Orientadora
Universidade Federal de Santa Catarina

_____
Prof., Dr. Marcelo Soares Pimenta
Universidade Federal do Rio Grande do Sul

_____
Prof., Dr. Ronaldo dos Santos Mello
Universidade Federal de Santa Catarina

_____
Prof., Dr. Ricardo Pereira e Silva
Universidade Federal de Santa Catarina

Dedico a Deus, o único. Que apesar de mim, é bondoso, e soberano sobre todo este trabalho, me dando força de vontade nas madrugadas e com surpresas inesperadas, mesmo após minha consideração de impossibilidade de conclusão.

AGRADECIMENTOS

À minha orientadora Patrícia Vilain pela orientação verdadeira; revisões, lembranças de datas, e horas de dedicação. Aos professores da banca, pela considerada análise, pela avaliação e dicas. Aos meus pais que simplesmente sempre fizeram de tudo para que eu pudesse ter esse curso e essa formação. À minha quase amada esposa pela ajuda, parceria das noites de escrita e acima disso pela paciência e compreensão da questão do tempo. E acima de tudo a Deus, a quem dediquei este trabalho.

Ó profundidade da riqueza, tanto da sabedoria como do conhecimento de Deus! Quão insondáveis são os seus juízos, e quão inescrutáveis, os seus caminhos! Quem, pois, conheceu a mente do Senhor? Ou quem foi o seu conselheiro? Ou quem primeiro deu a ele para que lhe venha ser restituído? Porque dele, e por meio dele, e para ele são todas as coisas. A ele, pois, a glória eternamente. Amém!

(Epístola de Paulo aos Romanos, capítulo 11, versículos 33 a 36)

RESUMO

A complexidade dos *softwares* e o projeto de interface com o usuário são dois dos principais desafios de desenvolvimento, pois exigem muito entendimento dos requisitos. Em resposta a isso, este trabalho propõe a combinação dos Diagramas de Interação com o Usuário (UID) juntamente com a abordagem de transformações automáticas de modelos (MDD). Juntas, essas técnicas permitem a captura, a modelagem da interação do usuário com o sistema em um alto nível de abstração e a geração automática de código. Uma ferramenta foi desenvolvida para transformar UIDs em modelos independentes de plataforma, que, por sua vez, são transformados em interfaces com o usuário implementadas em diferentes tecnologias. Um estudo de caso no qual interfaces com o usuário em JSF e ASP.Net são geradas a partir dos requisitos representados pelos UIDs é apresentado e comparado com aplicações reais para demonstrar a viabilidade da proposta.

Palavras-chave: Desenvolvimento Dirigido a Modelos (MDD). Diagrama de interação com o Usuário (UID). Geração automática de interfaces.

ABSTRACT

The complexity of software and the user interface design are two of the main development challenges, as they demand much understanding of the requirements. To address this issue, this study proposes the combination of User Interaction Diagrams (UID) along with the automatic model transformation approach (MDD). Together, these techniques allow the capturing, the modeling of user interaction with the system at a high level of abstraction, and automatic code generation. A tool was developed to transform UIDs in platform independent models, which, by their turn, are transformed into user interfaces implemented in different technologies. A case study where UIs with JSF and ASP.Net are generated from requirements represented by UIDs is presented and compared with real-world applications to demonstrate the feasibility of the proposal.

Keywords: Model-driven development (MDD). User interaction diagram (UID). User interface automatic generation.

LISTA DE FIGURAS

LISTA DE QUADROS E TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

A complexidade dos *softwares* é um dos principais desafios a serem tratados em seu desenvolvimento. Isso exige um bom entendimento dos requisitos, uma comunicação clara e uma ampla visibilidade deles por todos os interessados, assim como a capacidade de portabilidade e escalabilidade do *software*, uma boa produtividade em resposta às mudanças do mercado e a facilidade da manutenção do *software* sendo desenvolvido (SELIC, 2003).

A abordagem de desenvolvimento dirigida a modelos encontra-se como resposta para esse problema. Ela tem como objetivo elevar o nível de abstração de forma a facilitar o entendimento do *software* à medida que distancia a análise e o código. Ela auxilia também na manutenção, produtividade e reuso do *software*, uma vez que o desenvolvimento e alterações são feitos em um nível de abstração mais elevado, produzindo transformações automáticas que geram os demais artefatos, como o código, por exemplo (SELIC, 2003).

Outra área de grande importância no desenvolvimento de *software* é o projeto da interface com o usuário. A interface é o limite entre a máquina e o homem e, portanto, o que faz a experiência do usuário ser agradável. Entretanto, a interface é uma das partes do desenvolvimento que mais sofrem alterações devido ao surgimento de novas tecnologias e padrões de usabilidade. Além disso, essa fase também é responsável por uma parte significante do tempo de desenvolvimento do *software*. Assim, torna-se extremamente necessário o emprego de uma abordagem de fácil e ágil capacidade de captura, entendimento, geração e alteração de requisitos de interface (MYERS, 1993).

Para tratar essa questão, este trabalho propõe a combinação dos Diagramas de Interação com o Usuário (UID) (VILAIN, 2002) juntamente com a abordagem de transformações automáticas de modelos (MDD). Juntas, essas técnicas permitem a captura e modelagem da interação do usuário com o sistema em um alto nível de abstração. Além disso, essas técnicas permitem a geração automática de código. Portanto, o desenvolvimento da interface é melhorado por ser todo realizado em um único e elevado nível de abstração, o nível conceitual. Esse nível facilita o entendimento humano através de diagramas e permite a geração e alteração automática de códigos diferentes. Assim, os requisitos podem ser facilmente mapeados pelos UIDs, validados e distribuídos através das interfaces funcionais que foram geradas automaticamente. Além disso,

essas interfaces podem ser utilizadas como artefatos iniciais do projeto, bastando que as regras de negócio sejam implementadas e a aparência seja finalizada com imagens e estilos.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo deste trabalho é propor uma abordagem de desenvolvimento de transformação de modelos (MDD) para a geração automática de interfaces com o usuário, a partir dos diagramas de interação com o usuário (UID) a fim de auxiliar no levantamento de requisitos e, consequentemente, melhorar o desenvolvimento e gerar automaticamente interfaces como protótipos ou como ponto inicial para o desenvolvimento.

1.1.2 Objetivos Específicos

Além de definir uma abordagem de geração automática de interfaces com o usuário, outros objetivos nesse contexto são:

- Definir e implementar os mapeamentos dos elementos dos UIDs para um modelo conceitual de interface genérica.
- Definir e implementar os mapeamentos do modelo conceitual de interface genérica para diferentes interfaces específicas de tecnologias diferentes, como JSF e ASP.NET, por exemplo.
- Avaliar a implementação (i) através da comparação das interfaces de aplicações existentes e das interfaces geradas pela ferramenta proposta; e (ii) através de estudos de casos.

1.2 ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho está organizado da seguinte maneira. O capítulo 2 trata dos fundamentos conceituais necessários ao entendimento deste trabalho, contendo tecnologias, metodologias, técnicas e

arquiteturas de desenvolvimento, *frameworks* de interface com o usuário e padrões.

O capítulo 3 traz um levantamento dos trabalhos relacionados, apresentando suas características, com o objetivo de compará-las à proposta deste trabalho.

Apresentados os fundamentos necessários ao entendimento e os trabalhos mais próximos da área, é então explicada a proposta no capítulo 4. Essa proposta trata do desenvolvimento de um *software* que permite gerar interfaces automaticamente, através de transformações de modelos que se iniciam com o UID.

No capítulo 5, são apresentados dois estudos de caso como validação da proposta.

Por fim, no capítulo 6, são feitas as últimas considerações, sendo apresentados os resultados da proposta, sua validação e a comparação do *software* produzido com os trabalhos relacionados levantados.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os conceitos necessários ao entendimento da proposta.

2.1 UID

O Diagrama de Interação do Usuário (do inglês *User Interaction Diagram* – UID) é uma notação diagramática para representar a interação entre o usuário e uma aplicação (VILAIN, 2002).

Define-se UID como um conjunto de estados conectados através de transições. Os estados representam as informações que são trocadas entre o usuário e a aplicação, enquanto as transições são responsáveis pela troca do foco da interação de um estado para outro. Diz-se que um estado da interação é o foco da interação quando as informações contidas nesse estado representam as informações que estão sendo trocadas entre o usuário e a aplicação em um dado momento. As informações participantes da interação entre o usuário e a aplicação são apresentadas dentro dos estados de interação, mas algumas seleções e opções são associadas às transições. As transições são acionadas, geralmente, pela entrada ou seleção de informações pelo usuário (VILAIN, 2002).

A Figura 1 apresenta um exemplo de UID contendo os artefatos citados. Esse exemplo representa a interação que ocorre quando um usuário deseja encontrar hotéis a partir de um endereço destino de sua viagem, para conhecer o hotel e fazer sua reserva. Inicialmente, no estado inicial <1>, o usuário informa o Destino de sua viagem, representado por um item de dado. Então, no segundo estado <2>, o sistema apresenta a lista de Hotéis, representado por um conjunto de estruturas, contendo uma coleção de informações, como nome, descrição, endereço, contato, preço e avaliação. Neste ponto da interação, o usuário tem a opção de visualizar os detalhes do Hotel ou fazer a reserva do Hotel, representados pela chamada dos UIDs “Visualizar Hotel” e “Fazer Reserva”.

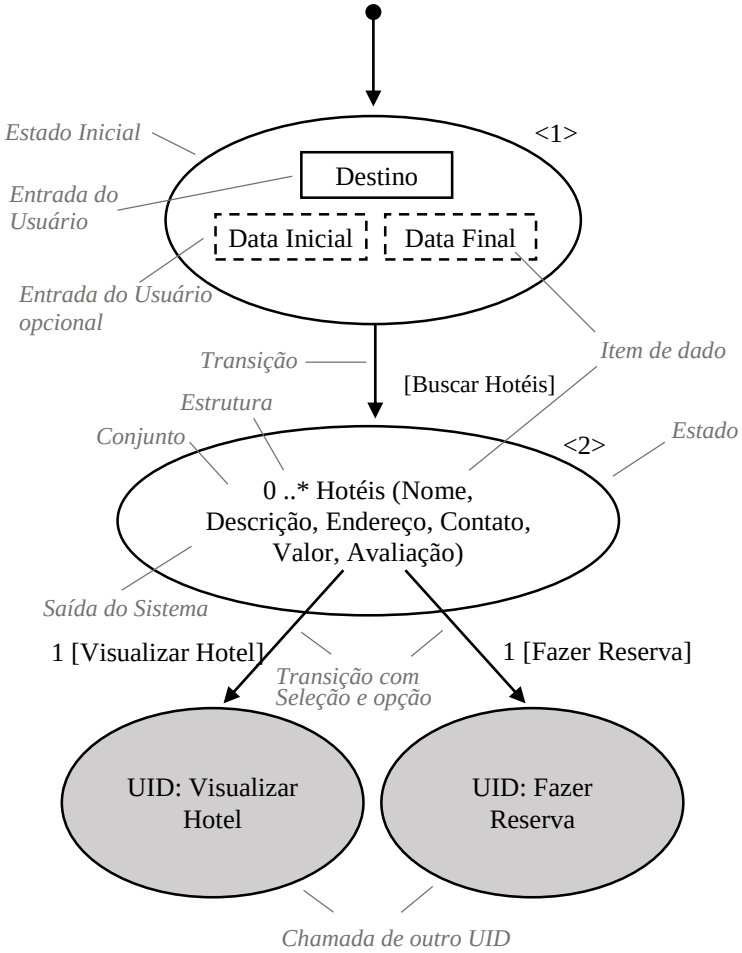

Figura 1 – Exemplo de UID

A seguir estão descritos os artefatos que compõem os UIDs:

- *Item de dado:* Representa uma informação única (simples) que aparece durante a interação. Ele pode estar acompanhado do seu domínio, sendo, neste caso, seguido por dois pontos e o nome do domínio;
- *Estrutura:* Representa uma coleção de informações (itens de dados, estruturas, conjunto de itens de dados ou conjunto de

estruturas) que estão relacionadas de alguma maneira. A coleção de informações de uma estrutura é especificada entre parênteses após o nome da estrutura;

- *Conjunto:* Representa um conjunto de itens de dados ou estruturas. A multiplicidade de um conjunto é representada por min..max em frente ao item de dado ou estrutura. A multiplicidade default é 1..N e é representada somente por reticências (...);
- *Entrada do Usuário:* Representa um item de dado ou estrutura fornecidos pelo usuário. Toda informação entrada pelo usuário é sempre colocada dentro de um retângulo;
- *Entrada do Usuário Opcional:* Representa uma entrada de usuário não obrigatória para a transição para o próximo estado da interação do usuário com o sistema;
- *Saída do Sistema:* Representa um item de dado ou estrutura retornados pelo sistema. Toda informação retornada pelo sistema é colocada diretamente no estado de interação, ou seja, tudo que aparece fora dos retângulos é informação retornada pelo sistema;
- *Estado da Interação:* Representa um estado da interação entre o usuário e o sistema. As informações fornecidas pelo usuário e as retornadas pelo sistema são usualmente colocadas dentro da elipse;
- *Estado Inicial da Interação:* Representa o estado inicial da interação entre o usuário e o sistema. Ele é representado por uma pequena transição sem origem;
- *Chamada de Outro UID:* Representa que o foco da interação foi transferido para outro UID;
- *Transição do Estado da Interação:* Representa que o estado destino torna-se o novo foco da interação após o sistema retornar as informações necessárias e o usuário fornecer os dados requisitados no estado origem.

Uma transição sempre está associada, no mínimo, à seleção de um elemento, à seleção de uma opção ou ao fornecimento de uma informação. A origem de uma transição pode ser todo um estado ou, quando um ou mais elementos precisam ser selecionados pelo usuário, um item de dado

ou mesmo uma estrutura, retornados pelo sistema. Quando o usuário seleciona alguma opção que não muda o foco da interação, a origem e o destino de uma transição podem ser o mesmo estado de interação.

É possível associar rótulos às transições de estado da interação. Esses rótulos podem ser combinados e associados às transições. A seguir são apresentados os possíveis rótulos:

- *Transição com Seleção de N Elementos:* Representa que N elementos devem ser selecionados pelo usuário antes que o estado de interação destino se torne o foco da interação;
- *Transição com Seleção da Opção X:* Representa que a opção X é selecionada para que o estado de interação destino se torne o foco da interação. O nome da opção é colocado entre parênteses.
- *Transição com Condição Y:* Representa que a condição Y deve ser satisfeita para que o estado de interação destino se torne o foco da interação. A condição é colocada entre colchetes.

2.2 MODEL-DRIVEN DEVELOPMENT (MDD)

O desenvolvimento de *software* é uma área conhecida por sua rápida evolução, e cada novo salto evolutivo tem-se caracterizado por uma melhora no nível de abstração que tende a aproximar as metodologias de desenvolvimento ao pensamento humano. Nesse contexto, Selic (2003) considera o desenvolvimento dirigido a modelos (do inglês *Model-Driven Development* – MDD) como a promessa de ser o primeiro verdadeiro salto evolutivo em desenvolvimento de *software*, desde a introdução do compilador.

Para entender o MDD, é necessário conhecer seu contexto. O MDD é parte da engenharia dirigida a modelos (do inglês *Model-Driven Engineering* – MDE), que, de acordo com Ameller (2009), pode ser mais bem explicada através das iniciativas históricas.

A primeira iniciativa foi a Arquitetura Orientada a Modelos (do inglês *Model-Driven Architecture* – MDA), que foi primeiramente publicada em 2000, seguindo o princípio de que "tudo é um modelo" (BÉZIVIN, 2005). Em 2004, ela foi definida oficialmente como um conjunto de padrões não proprietários que especificam que tecnologias interoperáveis irão realizar o desenvolvimento com transformações automáticas de modelos. Assim, para garantir esses padrões, as

linguagens dessas transformações precisam seguir os termos da linguagem de especificação de metamodelos, MOF (*Meta-Object Facility*).

Então foi definido o Desenvolvimento Dirigido a Modelos (MDD), que é a simples noção de que se pode construir um modelo de um sistema que pode ser transformado no sistema real (MELLOR et al., 2003). A diferença para o conceito da MDA é que o MDD não é associado com nenhum padrão OMG(Organização internacional que aprova padrões abertos para aplicações orientadas a objetos), o que o torna uma definição mais flexível do processo de desenvolvimento.

Finalmente, temos o conceito de Engenharia Dirigida a Modelos (MDE), que se refere à abordagem para tratar a incapacidade das linguagens de terceira geração para aliviar a complexidade de plataformas e expressar conceitos de domínio de forma eficaz (SCHMIDT, 2006).

Em resumo, conforme representado na Figura 2, o MDD é um paradigma de desenvolvimento que usa modelos como artefato base para a implementação. Esse paradigma foi utilizado na ferramenta desenvolvida nesta proposta. MDE, por sua vez, é toda a engenharia baseada em modelos, englobando, além das atividades de desenvolvimento (MDD), outras atividades, como engenharia reversa de sistemas legados. Por sua vez, MDA é a arquitetura específica baseada no modelo de desenvolvimento MDD que segue os padrões OMG.

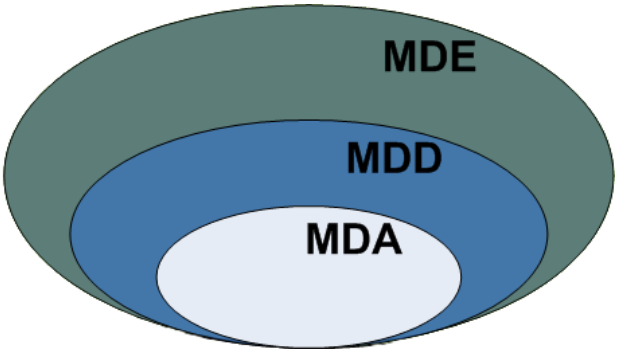

Figura 2 – Representação das iniciativas orientadas a modelos (adaptado de AMELLER, 2009, p. 13)

2.2.1 Model-Driven Architecture (MDA)

Conforme apresentado anteriormente, MDA é a arquitetura padrão para o desenvolvimento dirigido a modelos. Ela é definida pelo órgão de padronização de aplicações orientadas a objeto, a OMG. Portanto é o modelo a ser seguido no desenvolvimento dirigido a modelos. Ela define os seguintes padrões:

- *Meta-Objects Facility (MOF):* linguagem para especificação de metamodelos;
- *Unified Modeling Language (UML):* linguagem para especificação de sistemas;
- *Query/View/Transformation (QVT):* padrão de definição para linguagens de transformação;
- *XML Metadada Interchange (XMI):* padrão para troca de informações de metadados.

2.2.2 Modelos

Segundo Rothenberg (1989, p. 1), modelos podem ser definidos como:

> [...] representação da realidade para um determinado propósito; um modelo é a abstração da realidade no sentido de que ele não pode representar todos os aspectos da realidade. Isso nos permite lidar com o mundo de uma forma mais simplificada, evitando a complexidade da realidade.

Modelos são a base da abordagem de desenvolvimento utilizada e em MDA, são definidos em três tipos:

- *Modelo independente de computação (do inglês Computation Independent Model – CIM):* Também conhecido como modelo de negócios ou de domínio, o CIM é o modelo mais abstrato de MDA. Ele representa o contexto e o propósito do sistema sem nenhuma complexidade computacional, ou seja, sem considerar detalhes de tecnologia;
- *Modelo independente de plataforma (do inglês Platform Independent Model – PIM):* É o modelo que descreve o

comportamento e a estrutura do sistema sem considerar os aspectos relacionados a plataforma (*software* ou *hardware*);

- *Modelo específico de plataforma (do inglês Platform Specific Model – PSM):* É o modelo que combina as especificações do PIM com os detalhes de tecnologia particulares de uma plataforma, sendo assim o nível mais baixo de abstração e cujos elementos estão prontos para a geração de código.

2.2.3 Metamodelos

Modelos precisam ser construídos em uma forma não ambígua para que possam ser transformados. Portanto são necessários os metamodelos, que especificam sua estrutura, semântica e restrições. Nesse contexto, Atkinson e Kuhne (2003) classificam a estrutura dos metamodelos em quatro níveis:

- *M3:* camada que contém meta-meta-metadados para descrever as propriedades que o os meta-metadados podem exibir. Por exemplo, o diagrama de classes UML;
- *M2:* camada que contém meta-metadados para descrever as propriedades que os metadados podem assumir. Por exemplo, classes, atributos e operações UML;
- *M1:* camada que contém os metadados da aplicação. Por exemplo, classes de um sistema orientado a objetos ou um esquema de um banco de dados relacional;
- *M0:* camada que contém os dados da aplicação. Por exemplo, instâncias de classes ou registros de bancos de dados relacionais.

2.2.4 Transformação de Modelos

Segundo Czarnecki e Helsen (2003), transformações de modelos são o processo pelo qual um modelo é transformado em outro de acordo com seus metamodelos. Transformações são definidas por regras de mapeamentos de informações entre o modelo fonte e o modelo alvo.

Conforme mostrado na Figura 3, um modelo fonte descrito por seu metamodelo é transformado por uma máquina de transformação, de

acordo com suas regras de transformação, descritas em uma linguagem de transformação, em um modelo alvo descrito por seu metamodelo. Tudo isso é feito conforme especificado pela linguagem de definição de metamodelos MOF padrão da arquitetura MDA proposta pelo OMG.

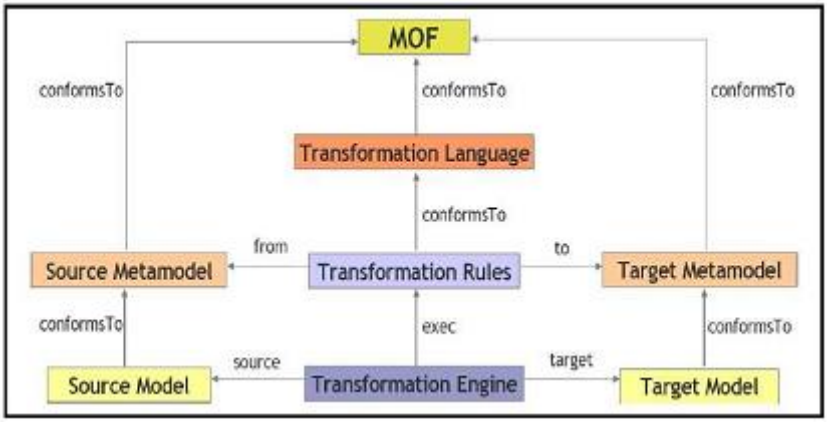

Figura 3 – Transformação de modelos (Di Ruscio, 2007)

Além da definição das transformações de modelos descrita acima, elas podem ser divididas também em três tipos:

- *Em diferentes níveis de abstração:* por exemplo, de CIM para PIM;
- *Em engenharia reversa:* no sentido contrário; por exemplo, de código para PSM;
- *Em refinamento de mesmo nível de abstração:* por exemplo, de um diagrama estrutural para um comportamental de UML.

Outro ponto importante para o entendimento das transformações de modelos, são duas categorias. Segundo Kleppe et al. (2003), as transformações de modelos possuem duas categorias, conforme mostrado na Figura 4:

- *Modelo-para-Modelo (do inglês Model-to-Model – M2M):* que representa a transformação entre modelos;
- *Modelo-para-Texto (do inglês Model-to-Text – M2T):* que representa a transformação de modelo para código textual.

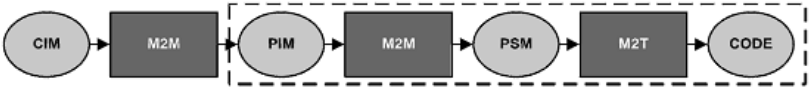

Figura 4 – Categorias de transformações de modelos
(AMELLER, 2009, p. 16)

O último aspecto a ser comentado são os pontos de vista que se pode ter sobre as transformações de modelos. Conforme OMG (2013), existem os seguintes dois pontos de vista:

- *Translationist:* sentido único abstrato para código;
- *Elaborationist:* ambos os sentidos, permitindo atualização de modelos mais abstratos a partir de alterações em código ou modelos mais específicos.

Esses pontos de vista são apresentados na Figura 5.

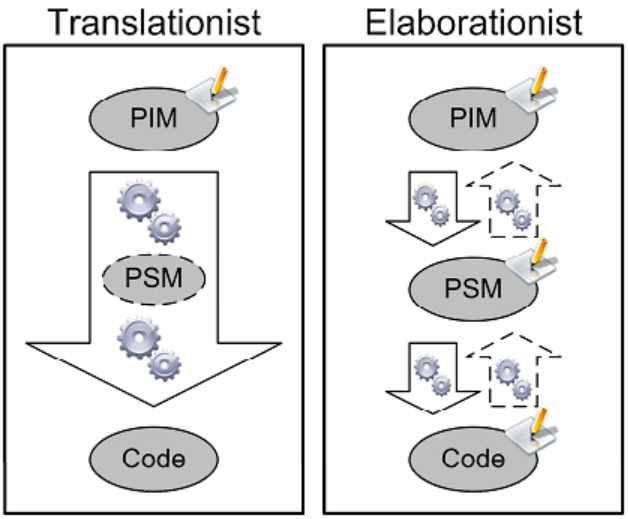

Figura 5 – Pontos de vista de transformações de modelos
(AMELLER, 2009, p. 17)

2.2.5 Eclipse Modeling Framework (EMF)

EMF é o *framework* da fundação Eclipse para geração de código e modelagem, que auxilia na criação de aplicações dirigidas por modelo. Ele usa o metametamodelo Ecore que é baseado no MOF. De acordo com Louhichi et al. (2011), o EMF é a plataforma MDE mais utilizada.

O projeto EMF iniciou em 2004, e é a tecnologia usada pela maioria das ferramentas de modelagem tanto do Eclipse quanto de empresas comerciais, diz Milinkovich, diretor executivo da fundação Eclipse. "Ele é usado como uma estrutura fundamental para a implementação, de, por exemplo, ferramentas UML e por produtos como o Rational Software Architect", explica ele (WATERS, 2009). O *framework* e sua facilidade de geração de código podem ser usados para construir ferramentas e outras aplicações baseadas em um modelo de dados estruturado. O EMF fornece ainda ferramentas e suporte de tempo de execução para a produção de classes Java a partir de um modelo. Por isso foi escolhido como *framework* sobre o qual a ferramenta foi implementada.

2.2.5.1 Atlas Transformation Language (ATL)

Como linguagem de transformação de modelos, a plataforma EMF utiliza a ATL. Ela é a mais popular da MDE, segundo Obeo (2013). Ela é uma linguagem da categoria M2M, híbrida, contendo a mistura de construtores declarativos e imperativos baseada na linguagem de restrições Object Constraint Language (OCL) para escrever expressões. As transformações ATL são unidirecionais. Assim, modelos fonte podem ser lidos e navegados, porém não alterados, enquanto modelos destino não podem ser navegados (LOUHICHI et al., 2011).

Fragal (2013) resume os componentes ATL basicamente nos seguintes elementos:

- *Headers:* Definem os atributos relativos à transformação. Por exemplo, o(s) metamodelo(s) de entrada e de saída;
- *Import:* seção opcional usada para importar bibliotecas ATL;
- *Helpers:* funções ATL semelhantes à linguagem Java;
- *Rules:* regras de transformação baseadas nos metamodelos. Existem dois tipos de regras: (i) construtores declarativos que

representam mapeamentos simples; e (ii) construtores imperativos que representam mapeamentos complexos chamados *called rules*.

Os construtores declarativos são divididos em 3 tipos:

- *Matched rules:* São aplicadas uma vez para cada correspondência e são executadas sequencialmente;
- *Lazy rules:* São aplicadas várias vezes para cada correspondência, porém precisam ser chamadas por outras regras para serem executadas;
- *Unique lazy rules:* São aplicadas uma vez a cada correspondência e necessitam ser referenciadas para serem executadas.

O modelo de transformação da linguagem ATL pode ser ilustrado pela Figura 6, na qual o modelo de Autor (Author.ecore), definido pelo seu metamodelo (authors.ecore), é transformado no modelo de Pessoa (Person.ecore) conforme seu metamodelo (persons.ecore) através da transformação ATL (Author2Person.atl).

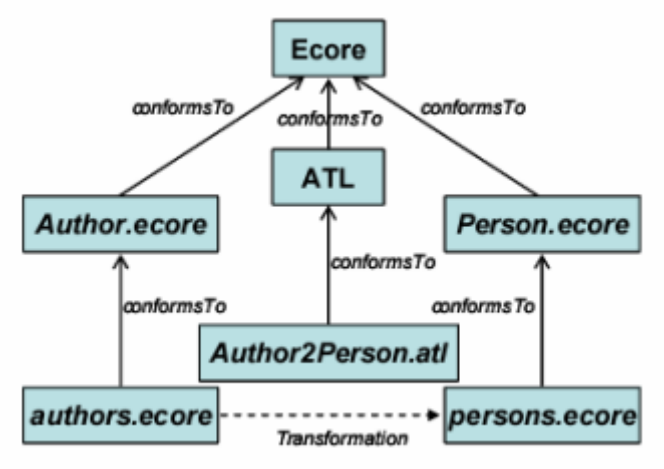

Figura 6 – Linguagem de transformações de modelos ATL (ATLAS, 2006 apud MEDEIROS, 2008, p. 46)

O código dessa transformação é mostrado na Figura 7, na qual o Autor é declarado na linha 2, "tipado" por Author!Author (par

metamodel_name!entity_name) na linha 6 e transformado em Pessoa, "tipada" na linha 8, conforme as ligações com Autor definidas nas linhas 9 e 10.

```
1   module Author2Person;
2   create OUT : Person from IN : Author;
3
4    rule Author2Person {
5     from
6               a : Author!Author
7     to
8               p : Person!Person (
9                       name <- a.name
10                      surname <- a.surname
11              )
12  }
```

Figura 7 – Código da transformação ATL (MEDEIROS, 2008, p. 47)

2.2.5.2 Kernel MetaMetaModel (KM3)

KM3 é o formato para definição de metamodelos da plataforma EMF. Ele é a resposta à proposta MOF da OMG, que não possuía um ambiente prático para o desenvolvimento. Sua notação é semelhante à da linguagem Java e é baseada em Ecore e EMOF, padrão OMG. Além disso, seus arquivos podem ser serializados em formato XMI, também conforme o padrão OMG.

As Figuras 8 e 9 apresentam um metamodelo e seu respectivo código que representam a estrutura de programas Java. Programas Java contêm pacotes (Package), com, por exemplo, um atributo nome (name do tipo String) e consistem de classes (Class). Classes, por sua vez, também possuem um atributo nome (name) e definem métodos (Method).

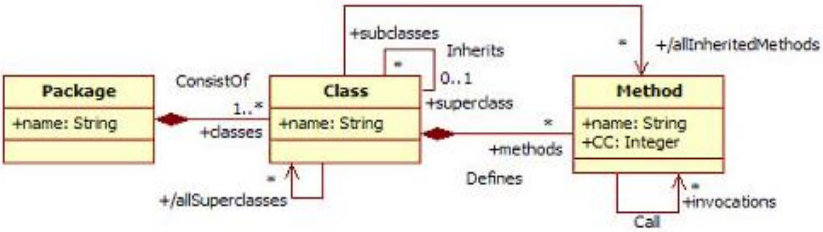

Figura 8 – Metamodelo KM3 (SCALISE et al., 2010)

```
package SourceCode {
  class Package {
    attribute name : String ;
    reference classes[1-*] container :
                Class oppositeOf "package" ;
  }
  class Class {
    attribute name : String ;
    reference "package" :
                Package oppositeOf classes ;
            reference superclass[0-1] :
                Class oppositeOf subclasses ;
            reference subclasses[*] :
                Class oppositeOf superclass ;
            reference methods[*] container :
                Method oppositeOf "class" ;
            reference allSuperclasses[*] : Class ;
            reference allInheritedMethods[*] : Method ;
  }
  class Method {
            attribute name : String ;
            reference "class" :
                Class oppositeOf methods ;
            reference invocations[*] : Method ;
  }
}
package PrimitiveTypes {
 datatype String;
}
```

Figura 9 – Código KM3 (SCALISE et al., 2010)

2.2.5.3 Acceleo

Como última parte do desenvolvimento dirigido a modelos (MDD) está a geração de código. No *framework* de modelagem do Eclipse (EMF),

o Acceleo é a alternativa de implementação do padrão da OMG para esse tipo de transformação, M2T (transformação modelo para texto).

Segundo Eclipse (2013), o projeto Acceleo é o resultado de vários anos de pesquisa e desenvolvimento iniciados na companhia francesa Obeo. Juntamente com o padrão de MTL (linguagem de transformação de modelos) da OMG, a experiência de sua equipe com a geração de código no contexto industrial e as mais recentes pesquisas no campo M2T dão ao Acceleo vantagens marcantes, como alta capacidade de personalização, interoperabilidade, fácil inicialização, rastreabilidade, manutenção, entre outros benefícios.

O Acceleo trabalha com o padrão de *templates*. Assim, um projeto destino, como uma aplicação *web* JSF ou ASP.NET, pode ser utilizado como *template*, ou seja, base para que seus arquivos sejam gerados. Assim, o processo de implementação da transformação de modelo para código é facilitado, bastando apenas copiar o conteúdo dos arquivos desse projeto *template* para arquivos Acceleo e substituir as partes variáveis por entradas que serão preenchidas pelos valores dos modelos e assim produzirão o código final, como exemplificam as Figuras 10 e 11.

Poligono.java | Poligono.mtl

```java
package br.natan;

public class Poligono {

    double ladoA;
    double ladoB;
    double ladoC;

    public Poligono(double ladoA, double ladoB, double ladoC) {
        this.ladoA = ladoA;
        this.ladoB = ladoB;
        this.ladoC = ladoC;
    }

    public double calcularPerimetro() {
        return this.ladoA + this.ladoB + this.ladoC;
    }

    public double getLadoA() {
        return ladoA;
    }

    public void setLadoA(double ladoA) {
        this.ladoA = ladoA;
    }

    public double getLadoB() {
        return ladoB;
    }
```

Figura 10 – Classe base a ter seu código copiado e utilizado pelo arquivo mtl para gerar a mesma classe como código final

.

Poligono.java | Poligono.mtl

```
[comment encoding = Cp1251 /]
[module Bean('http://www.ufsc.br/natan')/]

[template public Poligono(forma : Forma)]
    [file ('src/br/natan/'.concat(forma.id.toUpperFirst()), false, 'Cp1252')]

package br.natan;

public class Poligono {

    [for (lado : Lado | forma.lados)]
    double [lado.name.toLowerFirst()/];
    [/for]

    public Poligono(
    [for (lado : Lado | forma.lados)]
        [if (forma.lados.current(lado) <> forma.lados.size - 1)]
        double [lado.name.toLowerFirst()/],
        [/if]
        [if (forma.lados.current(lado) = forma.lados.size - 1)]
        double [lado.name.toLowerFirst()/]
        [/if]
    [/for]
    ) {
    [for (lado : Lado | forma.lados)]
        this.[lado.name.toLowerFirst()/] = [lado.name.toLowerFirst()/];
    [/for]
    }

    public double calcularPerimetro() {
        return
    [for (lado : Lado | forma.lados)]
        [if (forma.lados.current(lado) <> forma.lados.size - 1)]
            this.[lado.name.toLowerFirst()/] +
        [/if]
        [if (forma.lados.current(lado) = forma.lados.size - 1)]
```

Figura 11 – Arquivo mtl (Acceleo) baseado na classe java com as partes variáveis alteradas por código Acceleo para serem produzidas

2.3 WIDGETS ABSTRATOS

Com o objetivo de gerar interfaces com o usuário de diferentes tecnologias, os Widgets Abstratos tornam-se um conceito chave. Segundo Moura e Schwabe (2004), a ontologia de Widgets Abstratos é utilizada para especificar essas interfaces abstratas. Nela estão representados os Widgets essenciais que dizem respeito às trocas de informações entre o usuário e a aplicação, mas de forma independente das tecnologias e dos padrões de projetos utilizados.

Ainda segundo Moura e Schwabe (2004), a ontologia de Widgets Abstratos é composta por 11 conceitos (Figura 12), e esses conceitos tem

suas primitivas semelhantes as primitivas dos UIDs utilizados neste trabalho.

Dessa forma, os Widgets Abstratos constituem o modelo de representação em forma de elementos de interface mais relacionado ao modelo de trocas de informações do usuário com o Sistema, o UID. Além disso, por serem elementos de interface abstratos, não restringem a tecnologia a ser utilizada para a interface concreta final. Outro fator importante é que segundo Moura e Schwabe (2004), os widgets são traduzíveis em elementos de interface disponíveis nos ambientes de implementação comumente encontrados, como por exemplo, HTML.

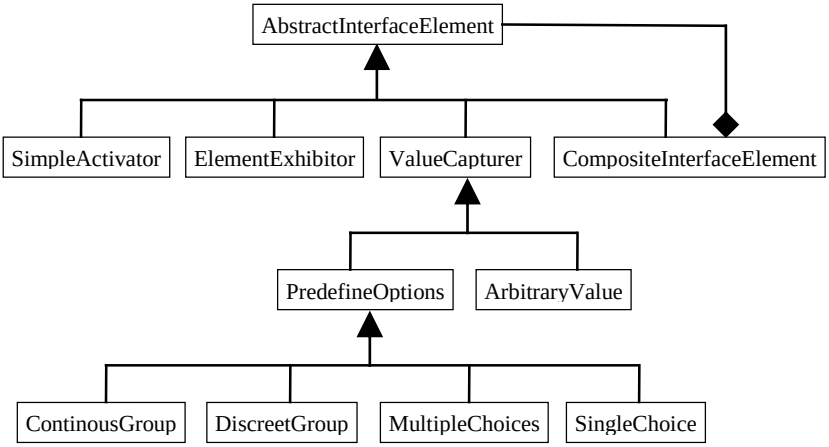

Figura 12 – Ontologia de Widgets Abstratos
(MOURA; SCHWABE, 2004, p. 108)

3 TRABALHOS RELACIONADOS

Foi realizada uma extensa pesquisa baseada nos termos “geração automática de interface”, “geração de interface”, “mdd”, “mde”, “mda”, “model-driven development”, “model-driven engineering”, “model driven architecture” e “UID”. Essa pesquisa utilizou a ferramenta de busca de literatura acadêmica Google Acadêmico[1], além da ferramenta de busca da *web* da Google e das próprias referências da literatura encontrada.

A seguir são apresentados os trabalhos encontrados que geram interfaces automáticas a partir do mapeamento de modelos conceituais para modelos que representam interfaces concretas.

3.1 UWE4JSF

A UWE4JSF é uma ferramenta de geração automática de aplicações *web*. Ela aplica a abordagem de Engenharia Web baseada em UML (do inglês *UML-based Web Engineering* – UWE). Ela utiliza a plataforma JSF como tecnologia de componentes de interface e, portanto, é específica para essa tecnologia.

Além disso, ela segue o modelo MDD, construindo um conjunto de modelos e transformando-os do nível mais abstrato para o mais concreto, até alcançar o nível de código. Esse processo utiliza as linguagens UML para modelagem, ATL para transformação do modelo independente de plataforma para o modelo conceitual JSF e JET (gerador de código do Eclipse) para transformação de modelo para o código final. Por fim, ela é completamente integrada com o Eclipse, sendo um conjunto de *plugins* (KROISS et al., 2009).

Sua diferença principal em relação à ferramenta proposta no presente trabalho é a restrição da tecnologia do código gerado, o *framework* JSF, enquanto a ferramenta por nós proposta propicia uma independência neste quesito.

[1] scholar.google.com.br

3.2 FERRAMENTA DE MAPEAMENTO DE UIDS PARA JSF

Essa é uma ferramenta de geração automática de páginas *web*. Ela realiza o mapeamento dos UIDs para a plataforma JSF, sendo assim específica para essa tecnologia, diferentemente da proposta deste trabalho. Ela utiliza a ontologia de Widgets Abstratos de (MOURA; SCHWABE, 2004), porém não utiliza o modelo de desenvolvimento MDD, implementando esse mapeamento através de uma codificação específica (DAMIANI; VILAIN, 2012), o que não permite alterações em nível de modelos, benefício também buscado em nossa proposta.

3.3 OOH4RIA

OOH4RIA é uma ferramenta que constrói automaticamente interfaces de aplicações de internet ricas (do inglês *Rich Internet Application – RIA*). Ela utiliza a abordagem de desenvolvimento MDD estendendo a metodologia de desenvolvimento hipermídia orientado a objetos, OOH de (GÓMEZ et al., 2001), para construir automaticamente interfaces de aplicações de *internet* ricas (do inglês *Rich Internet Application – RIA*).

Ela mapeia modelos estruturais e comportamentais para interfaces de aplicações RIA. A ferramenta utiliza especificamente, como componentes de interface, o *framework* Google Web Toolkit (GWT) (MELIÁ et al., 2008).

Esta é outra ferramenta que se restringe a uma tecnologia do código gerado, não oferecendo, portanto, uma das vantagens proporcionadas pela ferramenta por nós proposta.

3.4 AUTOWEB SYSTEM

A AutoWeb System é uma ferramenta que segue a abordagem AutoWeb (FRATERNALI; PAOLINI, 2000). A abordagem AutoWeb é baseada no padrão de desenvolvimento dirigido a modelos (MDD).

Sua implementação utiliza, para o levantamento de requisitos, um conjunto de esquemas conceituais da notação de especificação de estrutura, navegação e apresentação de aplicação, a HDM-lite (do inglês *Hipermídia Design Method*). A partir desses esquemas, uma base de dados relacional é criada para armazenar os modelos de navegação e apresentação, e os próprios dados da aplicação. Por fim, esses modelos

são transformados para modelos específicos de uma determinada tecnologia de interface para então gerarem a interface concreta.

Esta ferramenta também difere da proposta neste trabalho por se basear em uma tecnologia final específica.

3.5 MENDIX E WEBRATIO

Mendix e WebRatio são duas ferramentas proprietárias semelhantes baseadas na notação IFML (do inglês *Interaction Flow Modeling Language***)**. Elas suportam toda a abordagem de desenvolvimento dirigido a modelos (MDD). Dessa forma, elas permitem a geração de interfaces automaticamente (MENDIX, 2013; WEBRATIO, 2013).

Uma diferença em relação à proposta deste trabalho é que essas ferramentas não permitem a geração de interfaces para diferentes tecnologias. Elas utilizam tecnologias particulares de componentes de interface.

3.6 THE THREE CORE MODELS

The Three Core Models são descrições de três modelos analisados como principais para o padrão de desenvolvimento dirigido a modelos. O primeiro modelo, o modelo de Tarefa descreve as tarefas de um usuário em um sistema. O modelo de dialogo descreve o conjunto de tarefas que podem ser realizadas pelo usuário em cada estado do sistema. E o último modelo, o modelo de apresentação representa os elementos que uma interface de usuário oferece (MEIXNER et al., 2011). The Three Core Models não alcança o nível de ferramenta, sendo apenas um padrão de tipos de modelos existente atualmente. Devido ao modelo fonte da abordagem aqui proposta se basear no UID, está abordagem não se encaixa, não sendo utilizada.

3.7 THE CAMELEON REFERENCE FRAMEWORK

The Cameleon Reference Framework, se refere a um framework que descreve diferentes camadas de abstração que são importantes para o desenvolvimento baseado em modelos. As camadas propostas estão relacionadas primeiramente a tarefas e conceitos (do inglês *Tasks &*

Concepts), em seguida, se refere a interface de usuário abstrata (do inglês *Abstract User Interface* – AUI), e a terceira se refere a interface de usuário concreta (do inglês *Concrete User Interface* – CUI) e a interface de usuário final real (Final User Interface - FUI) (MEIXNER et al., 2011). Estas três camadas na realidade correspondem as camadas CIM, PIM, PSM e CODE do MDD, que portanto na abordagem aqui proposta também existem e são organizadas pelo framework da Eclipse escolhido, Eclipse Modeling Framework, conforme visto anteriormente.

3.8 USIXML

A Linguagem de Marcação Extensível de Interface de Usuário (do inglês *USer Interface, eXtensible Markup Language*) é estruturada de acordo com os quatro níveis de abstração definido pelo framework, *The Cameleon Reference* e se baseia na transformação do modelo de cada nível até o nível da interface de usuário final) (MEIXNER et al., 2011). Esta abordagem é bastante semelhante a proposta deste trabalho, permitindo inclusive a independência de tecnologia de interface. Ela tem como principal diferença, o modelo fonte, que neste caso é o UIDL, uma linguagem de definição de interface de usuário, enquanto a proposta desse trabalho se baseia no UID.

3.9 USEML + DISL + UIML

UseML + DSL + UIML é uma abordagem que usa três diferentes linguagens baseadas em XML e ela também é se utiliza do framework, *The Cameleon Reference*) (MEIXNER et al., 2011). Assim como a UsiXML esta abordagem também se assemelha muito a proposta deste trabalho, trabalhando com MDD e gerando interfaces de diferentes tecnologias, porém não utiliza como modelo fonte o UID.

3.10 FERRAMENTAS MDD SEM GERAÇÃO DE INTERFACES

Algumas ferramentas trabalham apenas em nível de modelagem de negócios e não alcançam o nível de geração de interfaces, diferenciando-se assim da proposta deste trabalho. São elas: AndroMDA, Borland

Together, Enterprise Architect, MDDi, Modelio, Novulo MDD Tool e Rational Rhapsody.

3.11 CONCLUSÃO

Pôde-se observar, entre os trabalhos relacionados, que nenhum deles utiliza ao mesmo tempo o UID como modelo de entrada; o desenvolvimento dirigido a modelos (MDD) como metodologia de desenvolvimento; e a capacidade de gerar interfaces para diferentes tecnologias.

A vantagem de se utilizar o UID como modelo de entrada, como explicado no capítulo de fundamentação, é sua capacidade de representação da interação do usuário com o sistema, o que não é tão claramente representado em outros modelos, como discutido em Vilain (2002).

Em relação à utilização da abordagem MDD, entre os benefícios da ferramenta aqui proposta, inclui-se a capacidade de desenvolvimento e alteração e manutenção em um alto nível de abstração, como foi explicado no capítulo de fundamentação.

Por fim, a presente proposta traz o benefício da independência de tecnologia. Ou seja, não é necessário se implementar uma aplicação em outra linguagem novamente e sim apenas acoplar os mapeamentos e transformações dos elementos dos Widgets Abstratos para os elementos da nova linguagem. Em uma ferramenta de geração automática de interfaces, isso é uma vantagem, pois permite a adequação a diferentes projetos de desenvolvimento.

O Quadro 1 faz uma comparação entre as ferramentas propostas em cada trabalho pesquisado e a do presente estudo.

Quadro 1 – Comparativo dos trabalhos relacionados

Ferramenta	Modelo de entrada	MDD	Independência de tecnologia de interface
Proposta deste trabalho	UID	Sim	Sim
UWE4JSF	UML	Sim	Não (JSF)
Ferramenta de Map. de UIDs para JSF	UID	Não	Não (JSF)
OOH4RIA	Modelos próprios estruturais e comportamentais	Sim	Não (GWT)
AutoWeb System	Esquemas conceituais da notação de especificação de estrutura, navegação e apresentação de aplicação (HDM-lite)	Sim	Não (Particular)
Mendix e WebRatio	Modelos próprios	Sim	Não (Particular)
UsiXML	UIDL	Sim	Sim
UseML+DISL +UIML	UseML, DISL e UIML	Sim	Sim
Ferramentas MDD sem geração de interfaces	UML ou Modelos próprios	Sim	Não (Não geram interface)

4 PROPOSTA

Com base nos conceitos apresentados anteriormente, foi desenvolvida uma ferramenta que implementa a abordagem proposta de usar o MDD para gerar interfaces com o usuário automaticamente a partir de UIDs. Todo o código da ferramenta pode ser encontrado no repositório do projeto, https://uid2ui.googlecode.com/svn/trunk/.

A abordagem transforma o modelo da interação do usuário com o sistema, representada através de UIDs, para um modelo conceitual de interface genérica, os Widgets Abstratos. Esse modelo genérico facilita a transformação para interfaces concretas específicas de tecnologias diferentes, como JSF e ASP.NET. A Figura 13 representa esse processo.

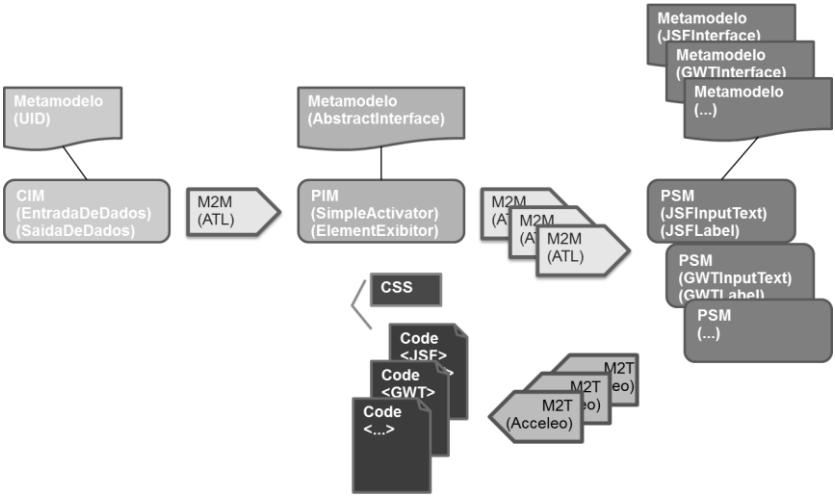

Figura 13 – Processo de transformação

A seguir são apresentadas as transformações de modelos, desde o UID para o modelo independente de plataforma. Primeiramente, apresenta-se a transformação desse modelo genérico para um modelo específico de cada tecnologia. Em seguida, a transformação desses modelos específicos para a real interface com o usuário. Para facilitar a explicação, foi utilizado o UID Buscar Hotéis apresentado na Figura 14.

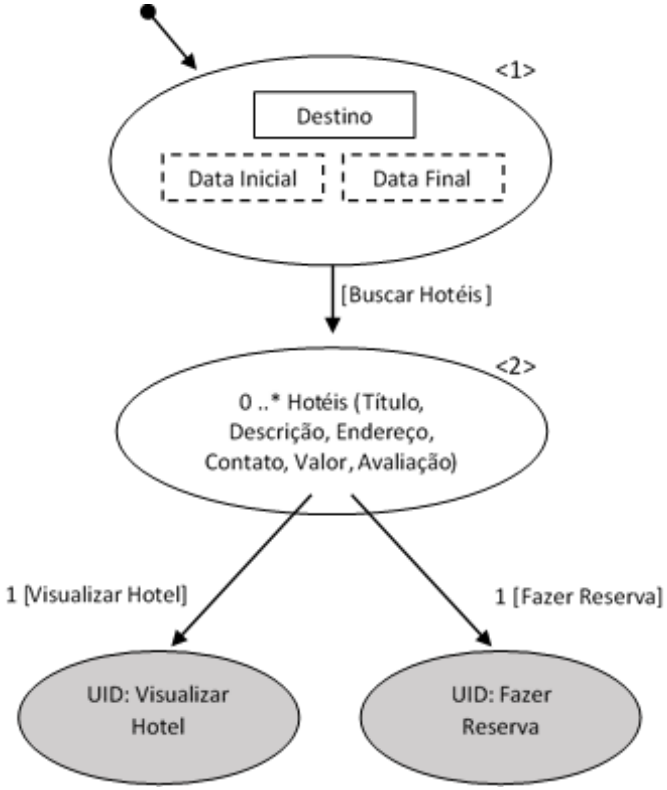

Figura 14 – UID Buscar Hotéis

4.1 TRANSFORMAÇÃO DE UIDS PARA WIDGETS ABSTRATOS (CIM-PARA-PIM)

De acordo com o MDD, o UID representa o modelo CIM, visto que, ele é independente de qualquer plataforma tecnológica e não está ligado a qualquer tecnologia. Por sua vez, os Widgets Abstratos representam o modelo PIM, visto que, eles são independentes de plataforma e não são ligados a nenhum *framework* de interface com o usuário, como JSF ou ASP.NET.

Assim, na primeira etapa, como definido no MDD, um modelo de origem, o UID, que é descrito pelo seu metamodelo, um arquivo Ecore, é transformado em um modelo destino, os Widgets Abstratos. O modelo destino também é descrito por um metamodelo. Então, a transformação

entre o modelo de origem e o modelo destino é feita através do mapeamento dos elementos do metamodelo fonte para seus respectivos elementos do metamodelo destino.

Para exemplificar, nesta seção é mostrada a transformação do UID Buscar Hotéis (Figura 14). No seu estado inicial, o usuário pode entrar com vários itens de dado, como hotel desejado e data de início. Depois disso, uma transição é lançada e o Sistema mostra a informação do hotel que corresponde à informação inserida. O metamodelo (Ecore) e o código (XMI) do UID são mostrados nas Figuras 15 e 16, respectivamente.

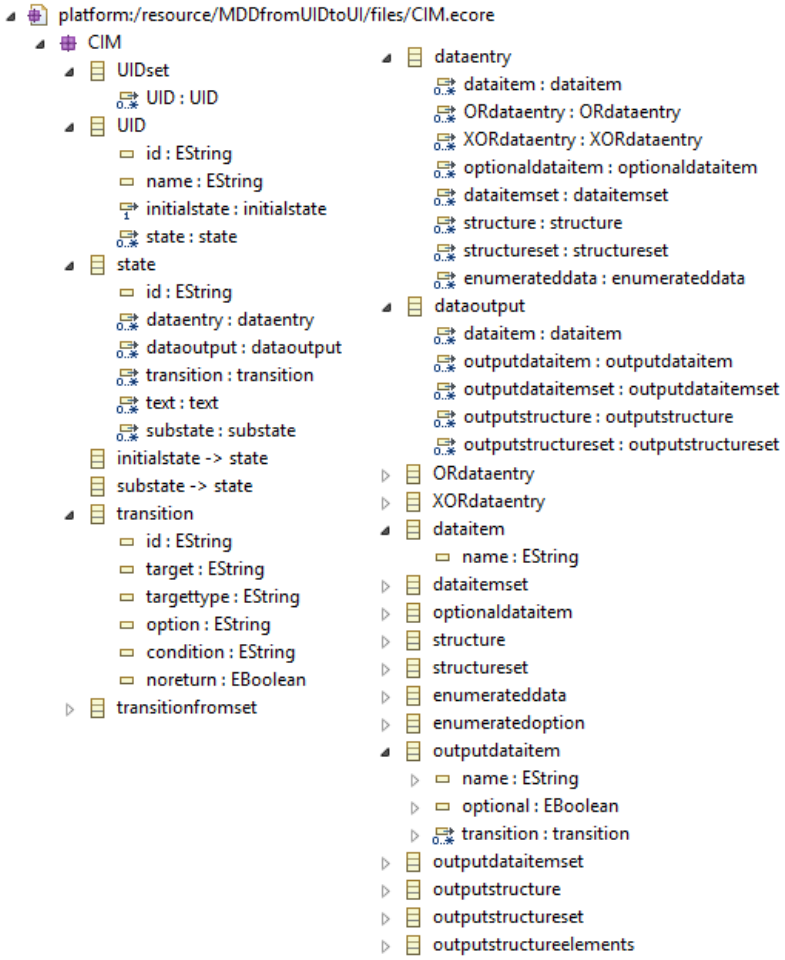

Figura 15 – Metamodelo Ecore do UID (CIM)

```
<?xml version="1.0" encoding="UTF-8"?>
<uid:Uid xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:uid="uid">
  <states>
    <outputs xsi:type="uid:Structure" name="Hotel" data="nome, valor, endereço, contato, ava
descrição"/>
  </states>
  <initialState>
    <inputs xsi:type="uid:LogicalOperatorUserInput" dataItem1="Destino" dataItem2="Hotel"/>
    <inputs xsi:type="uid:OptionalDataItemInput" name="Data Inicial"/>
    <inputs xsi:type="uid:OptionalDataItemInput" name="Data Final"/>
    <inputs xsi:type="uid:OptionalDataItemInput" name="Número de Quartos"/>
    <inputs xsi:type="uid:OptionalDataItemInput" name="Números de Adultos"/>
    <inputs xsi:type="uid:OptionalDataItemInput" name="Número de Crianças"/>
  </initialState>
  <transitions xsi:type="uid:RegularTransitionFromOne" target="//@states.0" source="//@initi
  <point/>
  <initialTransition source="//@point" target="//@initialState"/>
</uid:Uid>
```

Figura 16 – Código do UID (CIM) Buscar Hotéis

Então, o UID é transformado em Widgets Abstratos, como mostrado nas Figuras 17 e 18 (estrutura e código do arquivo Ecore). O modelo resultante consiste em uma *homepage* (CompositeInterfaceElement) que contém elementos de entrada de dados (ArbitraryValue) e um elemento de ativação (SimpleActivator), responsável pela navegação para a página que mostra os elementos do hotel (ElementExibitor). Esses elementos foram mapeados diretamente dos elementos do UID. A transformação ocorre através da execução do código ATL, que contém as regras que transformam cada elemento do UID para um elemento do modelo dos Widgets Abstratos.

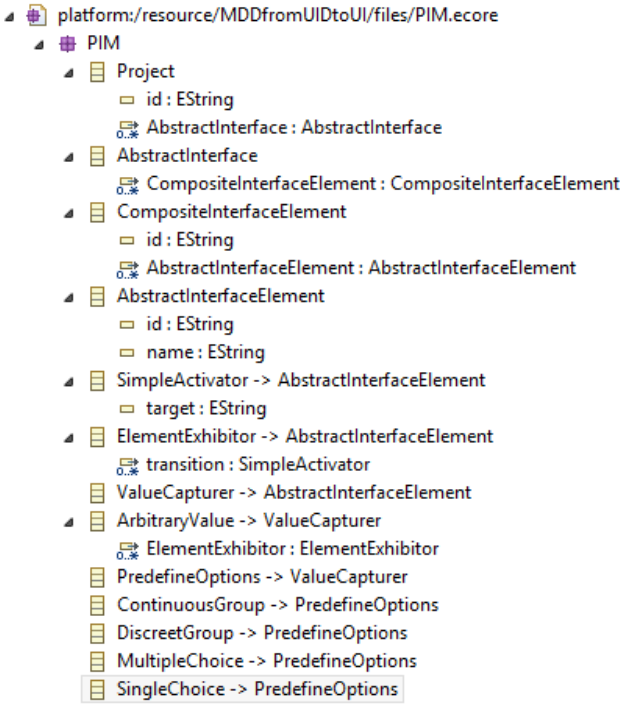

Figura 17 – Metamodelo Ecore dos Widgets Abstratos (PIM)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="PIM">
  <Project>
    <AbstractInterface>
      <CompositeInterfaceElement id="initialState">
        <AbstractInterfaceElement xsi:type="ArbitraryValue" name="Destino"/>
        <AbstractInterfaceElement xsi:type="ArbitraryValue" name="Hotel"/>
        <AbstractInterfaceElement xsi:type="ArbitraryValue" name="Data Inicial"/>
        <AbstractInterfaceElement xsi:type="ArbitraryValue" name="Data Final"/>
        <AbstractInterfaceElement xsi:type="ArbitraryValue" name="Numero de Quartos"/>
        <AbstractInterfaceElement xsi:type="ArbitraryValue" name="Numero de Adultos"/>
        <AbstractInterfaceElement xsi:type="ArbitraryValue" name="Numero de Criancas"/>
        <AbstractInterfaceElement xsi:type="SimpleActivator" id="transition0" target="sta
      </CompositeInterfaceElement>
      <CompositeInterfaceElement id="states0">
        <AbstractInterfaceElement xsi:type="ElementExibitor" name="Nome"/>
        <AbstractInterfaceElement xsi:type="ElementExibitor" name="Valor"/>
        <AbstractInterfaceElement xsi:type="ElementExibitor" name="Endereco"/>
        <AbstractInterfaceElement xsi:type="ElementExibitor" name="Contato"/>
        <AbstractInterfaceElement xsi:type="ElementExibitor" name="Avaliação"/>
        <AbstractInterfaceElement xsi:type="ElementExibitor" name="Descricao"/>
      </CompositeInterfaceElement>
    </AbstractInterface>
  </Project>
</xmi:XMI>
```

Figura 18 – Código do modelo dos Widgets Abstratos (PIM) gerado para a interação Buscar Hotéis

A Figura 19 mostra essas regras de transformação, desde o UID, formado por estados e transições de interação, e os componentes que formam os estados das interações.

```
module CIM2PIM;
create OUT: PIM from IN: CIM;

rule UIDset2Project {
    from
        source: CIM!UIDset
    to
        target: PIM!Project (
            AbstractInterface <- source.UID
        )
}

rule UID2AbstractInterface {
    from
        source: CIM!UID
    to
        target: PIM!AbstractInterface (
            CompositeInterfaceElement <- source.initialstate,
            CompositeInterfaceElement <- source.state
        )
}

rule State2CompositeInterfaceElement {
    from
        source: CIM!state
    to
        target: PIM!CompositeInterfaceElement (
            id <- source.id,
            AbstractInterfaceElement <- source.dataentry -> collect(e | e.ORdataentry -> coll
            AbstractInterfaceElement <- source.dataentry -> collect(e | e.XORdataentry -> col
            AbstractInterfaceElement <- source.dataentry -> collect(e | e.dataitem),

            AbstractInterfaceElement <- source.dataoutput -> collect(e | e.outputdataitem),

            AbstractInterfaceElement <- source.transition
        )
}

rule DataItem2ArbitraryValue {
    from
        source: CIM!dataitem(source.refImmediateComposite().oclType().toString() = 'CIM!datae
    to
        target: PIM!ArbitraryValue (
            name <- source.name
        )
}

rule OutputDataItem2ElementExhibitor {
    from
        source: CIM!outputdataitem
    to
        target: PIM!ElementExhibitor (
            name <- source.name,
            transition <- source.transition
        )
}

rule Transition2SimpleActivator {
    from
        source: CIM!transition
    to
        target: PIM!SimpleActivator (
            id <- source.id,
            target <- source.target
        )
}
```

Figura 19 – Código das regras de transformações ATL do modelo CIM para PIM

A primeira regra a ser destacada é a “UIDset2Project”, que transforma um conjunto de estados de interação em um projeto (Project). Um projeto é composto de interfaces abstratas, então a regra “UID2AbstractInterface” é acionada. Essa regra transforma um UID nessas interfaces abstratas. Como um UID é composto de estados, a regra “State2CompositeInterfaceElement” é acionada. Essa regra transforma um estado em uma composição de elementos de interface. O conjunto de elementos de interface, como seu nome sugere, se refere a um conjunto de elementos de interface. Assim a regra de transformação de elementos de interface é acionada. Como os elementos de interface são uma generalização de cada tipo de elemento de interface, há uma regra de transformação para cada um desses tipos. Então a regra “DataItem2ArbitraryValues” é acionada. Essa regra transforma o elemento de interface item de entrada de dado (DataItem) em um campo de entrada de dado (ArbitraryValues).

4.2 TRANSFORMAÇÃO DE WIDGETS ABSTRATOS PARA WIDGETS ESPECÍFICOS (PIM-PARA-PSM)

Na segunda etapa, o modelo PIM gerado anteriormente é transformado em diferentes modelos PSM, um para cada tecnologia. Para fazer isso, é necessário descrever os componentes de interface com o usuário de cada *framework*, como JSF e ASP.NET, em um metamodelo. As Figuras 20 e 21 mostram a estrutura e código do metamodelo para os componentes JSF.

Então, tem-se que identificar que elemento de cada tecnologia representa os elementos do modelo PIM, assim como CompositeInterfaceElement, SimpleActivator, ArbitraryValues, e escrever regras no código ATL. A Figura 22 mostra algumas dessas regras de mapeamento de alguns Widgets Abstratos para elementos JSF (o conjunto de todas as regras pode ser visualizado no repositório do projeto, https://uid2ui.googlecode.com/svn/trunk/).

Note que embora se precise de um modelo e um arquivo de transformação ATL para cada tecnologia, o desenvolvimento não requer muito esforço. Visto que cada tecnologia específica geralmente possui componentes de interface com o usuário para todos os elementos do modelo PIM, na prática, precisa-se apenas replicar as regras de transformação, substituindo os elementos em cada linguagem específica.

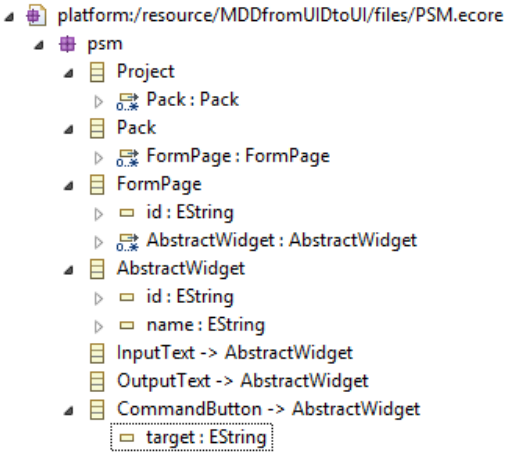

Figura 20 – Metamodelo Ecore dos Componentes JSF (PSM)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Project xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi
  <Pack>
    <FormPage id="initialState">
      <AbstractWidget xsi:type="InputText" name="Destino"/>
      <AbstractWidget xsi:type="InputText" name="Hotel"/>
      <AbstractWidget xsi:type="InputText" name="Data Inicial"/>
      <AbstractWidget xsi:type="InputText" name="Data Final"/>
      <AbstractWidget xsi:type="InputText" name="Número de Quartos"/>
      <AbstractWidget xsi:type="InputText" name="Números de Adultos"/>
      <AbstractWidget xsi:type="InputText" name="Número de Crianças"/>
      <AbstractWidget xsi:type="CommandButton" target="states0"/>
    </FormPage>
    <FormPage id="states0">
      <AbstractWidget xsi:type="OutputText" name="nome"/>
      <AbstractWidget xsi:type="OutputText" name="valor"/>
      <AbstractWidget xsi:type="OutputText" name="endereço"/>
      <AbstractWidget xsi:type="OutputText" name="contato"/>
      <AbstractWidget xsi:type="OutputText" name="avaliação"/>
      <AbstractWidget xsi:type="OutputText" name="descrição"/>
    </FormPage>
  </Pack>
</Project>
```

Figura 21 – Código do modelo JSF (PSM) gerado para a interação Buscar Hotéis

```
module PIM2PSM;
create OUT: PSM from IN: PIM;

rule Project2Project{
    from
        source: PIM!Project
    to
        target: PSM!Project(
            Pack <- source.AbstractInterface
        )
}

rule AbstractInterface2Pack{
    from
        source: PIM!AbstractInterface
    to
        target: PSM!Pack(
            FormPage <- source.CompositeInterfaceElement
        )
}

rule CompositeInterfaceElement2FormPage{
    from
        source: PIM!CompositeInterfaceElement
    to
        target: PSM!FormPage(
            id <- source.idTest,
            AbstractWidget <- Set{source.AbstractInterfaceElement}
        )
}

rule ValueCapturer2InputText {
    from
        source: PIM!ValueCapturer
    to
        target: PSM!InputText (
            id <- source.id,
            name <- source.name
        )
}

rule ElementExhibitor2OutputText {
    from
        source: PIM!ElementExhibitor
    to
        target: PSM!OutputText (
            id <- source.id,
            name <- source.name
        )
}

rule SimpleActivator2CommandButton {
    from
        source: PIM!SimpleActivator
    to
        target: PSM!CommandButton (
            id <- source.id,
            name <- source.name,
            target <- source.targetTest
        )
}
```

Figura 22 – Código das regras de transformações ATL do modelo PIM para PSM

4.3 MAPEAMENTO A PARTIR DOS UIDS PARA WIDGETS ABSTRATOS E DESSES PARA WIDGETS ESPECÍFICOS (CIM-PARA-PIM-PARA-PSM)

Para que as transformações dos UIDs sejam feitas para os Widgets Abstratos, foi feito o mapeamento de cada elemento do modelo de origem para o seu respectivo representante no modelo destino.

Existe mais de uma possibilidade de mapeamento de um componente de origem para seu respectivo destino. Porém, para que a transformação ocorresse automaticamente, foi definido um único mapeamento, de forma a atender sem perdas a troca de informação do usuário com o sistema representada no UID.

Da mesma forma que na transformação CIM para PIM, foi feito o mapeamento de cada elemento do modelo de Widgets Abstratos para o modelo de Widgets Específicos.

Seguindo o cenário de exemplo deste trabalho, o mapeamento realizado foi direcionado à tecnologia JSF. Além disso, foi mapeado também para a tecnologia ASP.NET, como demonstração da capacidade de geração para diversas tecnologias, como proposto. O Quadro 2 mostra alguns desses mapeamentos.

Vale destacar que mapeamentos de UIDs para Widgets Abstratos e de Widgets Abstratos para JSF também foram realizados por (DAMIANI; VILAIN, 2012) e que serviram como base para o entendimento e definição dos mapeamentos aqui apresentados.

Quadro 2 – Mapeamento CIM para PIM e para PSM

UIDs	**Widgets Abstratos**	**JSF**	**ASP.NET**
Entradas de Usuário			
Item de Dado	ArbitraryValue	InputText	TextBox
Conjunto de Itens de Dado	CompositeInterfaceElement formado por ArbitraryValues	Conjunto de InputTexts	Conjunto de TextBox
Entrada Enumerada	MultipleChoice	SelectManyListBox	ListBox
Estrutura	CompositeInterfaceElement formado por um ElementExhibitor como título e ArbitraryValues para cada componente diferente da estrutura	PanelGroup formado por InputTexts	Panel formado por TextBoxes

Conjunto de Estruturas	CompositeInterfaceElement de CompositeInterfaceElements formados por ElementExhibitor como título de cada estrutura e ArbitraryValues para cada component diferente da estrutura	Conjunto de PanelGroups formados por InputTexts	Conjunto de Panels formados por TextBoxes
Entrada Opcional	Propriedade nos elementos de Entrada de Usuário	Destaque (ex: asterisco) nos elementos de Entrada de Usuário	Destaque (ex: asterisco) nos elementos de Entrada de Usuário
Saídas de Sistema			
Texto	ElementExhibitor	OutputText	Label
Item de Dado	ElementExhibitor	OutputText	Label
Conjunto de Itens de Dados	CompositeInterfaceElement formado por ElementExhibitor	PanelGroup formado por OutputTexts	Panel formado por Labels
Estrutura	CompositeInterfaceElement formado por um ElementExhibitor como título e ElementExhibitors para cada componente diferente da estrutura	OutputText e PanelGroup formado por OutputTexts	Label e Panel formado por Labels
Set of Structures	CompositeInterfaceElement de CompositeInterfaceElements formado por ElementExhibitor como título de cada estrutura e ElementExhibitors para cada componente diferente da estrutura	PanelGrid	GridView
Estados de Interação			
Estado de Interação	CompositeInterfaceElement	Form	FormView
Estado Inicial de Interação	Propriedade nos Estados de Interação	Form	FormView
Estado de Interação Alternativo	Estado de Interação Normal que depende da Entrada do Usuário	Form	FormView
Sub-estado	CompositeInterfaceElement interno ao CompositeInterfaceElement de outro Estado de Interação	Form	FormView
Transições			
Transições com N elementos de	SimpleActivators dependentes da Entrada do Usuário	CommandLink (nas linhas dos PanelGrids) / CommandButton	LinkButton (nas linhas dos GridViews) /

seleção e opção de seleção X		(em PanelGroups e Forms	Button (em Panels e FormViews
UIDs			
UID	AbstractInterface	Page	Page
Demais			
Chamada de outro UID	SimpleActivators dependentes da Entrada do Usuário	CommandLink (nas linhas dos PanelGrids) / CommandButton (em PanelGroups e Forms	LinkButton (nas linhas dos GridViews) / Button (em Panels e FormViews
Chamada originada de outro UID	CompositeInterfaceElement	Form	FormView
Transição com condição Y	SimpleActivators dependentes da Entrada do Usuário	CommandLink (nas linhas dos PanelGrids) / CommandButton (em PanelGroups e Forms	LinkButton (nas linhas dos GridViews) / Button (em Panels e FormViews
Précondições	Sem mapeamento		
Póscondições	Sem mapeamento		
Notas textuais	Sem mapeamento		

4.4 TRANSFORMAÇÃO DE WIDGETS ESPECÍFICOS PARA CÓDIGO (PSM-PARA-CÓDIGO)

Em relação às transformações, essa é a última etapa. Nessa fase, não é mais utilizada a linguagem ATL pois essa suporta apenas transformações entre modelos. Em vez dela, é utilizado o *framework* Acceleo, que transforma o modelo PSM em código.

O Acceleo trabalha com o padrão de *templates*. Assim um projeto destino, no caso, uma aplicação *web* JSF ou ASP.NET, deve ser criado para ser utilizado como base, e suas partes variáveis devem ser codificadas via Acceleo, para que sejam preenchidas pelos valores dos modelos. Ou seja, para cada projeto destino, deve ser criado um arquivo Acceleo (Figura 23), e para cada arquivo da aplicação um arquivo Acceleo (Figuras 24 e 25) também deve ser criado.

Então, a partir do modelo PSM, é feito um mapeamento entre o modelo e os arquivos de código a serem gerados. Por exemplo, a partir de um elemento “Project”, os arquivos da aplicação serão gerados através da execução dos arquivos de geração de código Acceleo. Outro exemplo são os elementos “AbstractInterface”, que gerarão páginas JSF ou ASP.NET, e o elemento “FormPage” ou “FormPanel”, que irá gerar um formulário dentro das páginas *web*. Por fim, os elementos “CommandButton”,

“InputText” e “Button” ou “TextField” irão gerar os botões e campos dentro dos formulários *web*.

As páginas geradas podem ser visualizadas nas Figuras 26 e 27. A primeira representa a transição com o usuário no seu estado inicial, <1>, e a segunda, se refere ao estado <2>.

```
[comment encoding = UTF-8 /]
[module main('http://www.ufsc.br/natan/mddfromuidtoui')/]
[import JavaServerFacesAcceleo::files::FormPage/]
[import JavaServerFacesAcceleo::files::Bean/]


[template public main(aFormPage : FormPage)]

    [comment @main /]
    [aFormPage.FormPage()/]
    [aFormPage.Bean()/]

[/template]
```

Figura 23 – Arquivo Acceleo de template de controle de geração dos demais arquivos

```
[comment encoding = UTF-8 /]
[module FormPage('http://www.ufsc.br/natan/mddfromuidtoui')/]

[template public FormPage(aFormPage : FormPage)]
    [file ('WebContent/'.concat(aFormPage.id.toUpperCase().concat('.xhtml')), false,

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

<f:loadBundle basename="resources.application" var="msg"/>

<head>
    <title><h:outputText value="#{msg.welcomeTitle}" /></title>
</head>

<body>
<h:form>
    [for (w : AbstractWidget | aFormPage.AbstractWidget)]
        [if (oclIsTypeOf(InputText))]
            <h:inputText id="[w.name.replaceAll(' ', '_').toLowerFirst()/]"
value="#{[aFormPage.id.toLowerFirst().concat('Bean.').concat(w.name.replaceAll(' ',
        [/if]
        [if (oclIsTypeOf(OutputText))]
            <h:outputLabel for="[w.name.replaceAll(' ', '_').toLowerFirst()/]">[w.na
        [/if]
        [if (oclIsTypeOf(CommandButton))]
            <h:commandButton value="Submit" action="[oclAsType(CommandButton).target
        [/if]
    [/for]
```

Figura 24 – Arquivo Acceleo de template de geração das páginas JSF

```
[comment encoding = Cp1251 /]
[module Bean('http://www.ufsc.br/natan/mddfromuidtoui')/]

[template public Bean(aFormPage : FormPage)]
    [file ('src/br/ufsc/natan/mddfromuidtoui/'.concat(aFormPage.id.toUpperFirst().concat

package br.ufsc.natan.mddfromuidtoui;

import java.io.Serializable;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class [aFormPage.id.toUpperFirst().concat('Bean')/] implements Serializable {

    private static final long serialVersionUID = 1L;

    [for (w : AbstractWidget | aFormPage.AbstractWidget)]
        [if (oclIsTypeOf(InputText) or oclIsTypeOf(OutputText))]
            private String [w.name.replaceAll(' ', '_').toLowerFirst()/];

            public String get[w.name.replaceAll(' ', '_').toUpperFirst()/]() {
                return [w.name.replaceAll(' ', '_').toLowerFirst()/];
            }

            public void set[w.name.replaceAll(' ', '_').toUpperFirst()/](String [w.name.
                this.[w.name.replaceAll(' ', '_').toLowerFirst()/] = [w.name.replaceAll(
            }

        [/if]
    [/for]
}

    [/file]
[/template]
```

Figura 25 – Arquivo Acceleo de template de geração dos Beans JSF

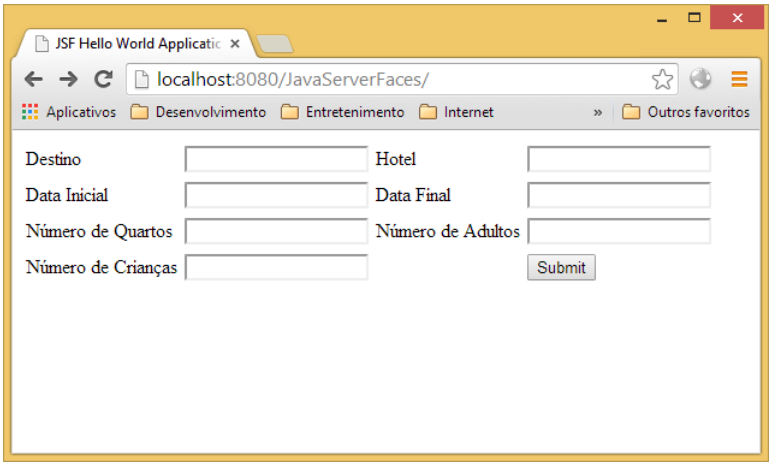

Figura 26 – Interface gerada respectiva ao estado inicial do UID Buscar Hotéis

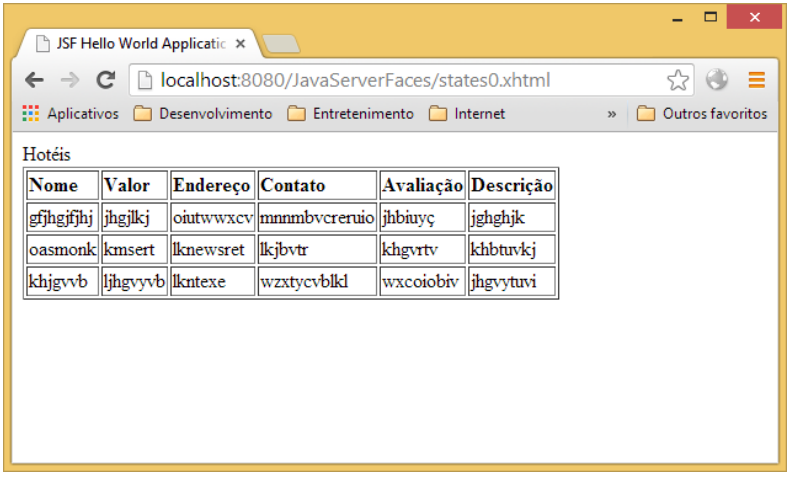

Figura 27 – Interface gerada respectiva ao estado “0” do UID Buscar Hotéis

4.5 GERAÇÃO DE DADOS

Nesta etapa, o modelo foi completamente transformado em código, e a aplicação gerada contendo as interfaces funcionais pode ser executada,

com usabilidade e navegação entre as telas, porém sem transporte de dados.

Como no UID é definido apenas as trocas de informações, ou seja, é definido apenas que o sistema irá exibir uma lista de hotéis para o usuário. Entretanto, em nenhum momento são definidos quais hotéis o sistema possui. Dessa forma, a aplicação gerada através da transformação de modelos não possui dados, conforme apresentado na Figura 28.

Portanto, foi implementada a geração automática de dados. Com a finalidade de facilitar os testes de interface e garantir apenas a funcionalidade do transporte de dados e não a exemplificação de cenários reais, os dados são criados numa lógica de utilizar o nome do elemento do UID seguido de um número sequencial e numa quantidade de 10 registros por conceito, conforme apresentado na Figura 28.

Em se tratando de cadastros, talvez não fosse necessária a geração de dados, uma vez que esses podem ser criados pela própria utilização do usuário. Porém, mesmo nesse contexto, eles continuam sendo úteis, no sentido de que nada precisa ser feito para que o sistema possa ser usado. Além disso, em contextos diferentes de cadastros, onde o sistema possui os dados e o usuário os visualiza e trabalha sobre eles, essa etapa se torna fundamental. Vale ressaltar que esta geração deve ser retirada no momento da utilização do código fonte gerado como código da aplicação real, pois estes dados são dados fictícios necessários apenas para testes e para a análise junto ao cliente.

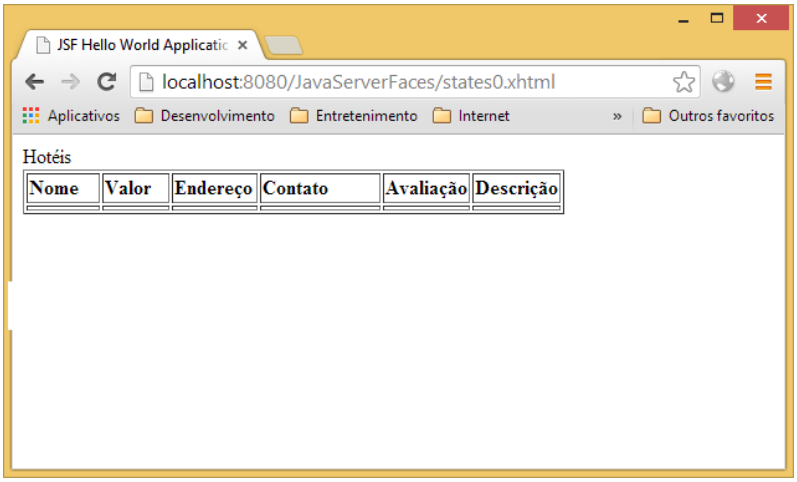

Figura 28 – Interface sem dados gerada respectiva ao estado “0” do UID Buscar Hotéis

4.6 LAYOUT GENÉRICO

Por fim, os UIDs não tratam da interface gráfica com o usuário. Eles definem somente o modelo dos dados e não como os dados se dispõem na interface gráfica. Por isto, após o mapeamento dos UIDs para uma interface gráfica, torna-se necessária a organização dos componentes de interface.

Foi implementada, então, uma disposição padrão de layout através de CSS. Com base nos componentes dos UIDs, são criadas classes CSS para cada um deles. Por ordem de criação, eles são dispostos sequencialmente na tela com um padrão no qual elementos de campos são colocados em duas colunas e elementos de *grid* ocupam as duas colunas.

4.7 UTILIZAÇÃO

A ferramenta proposta foi desenvolvida em Java e é formada por um conjunto de projetos, conforme apresenta a Figura 29.

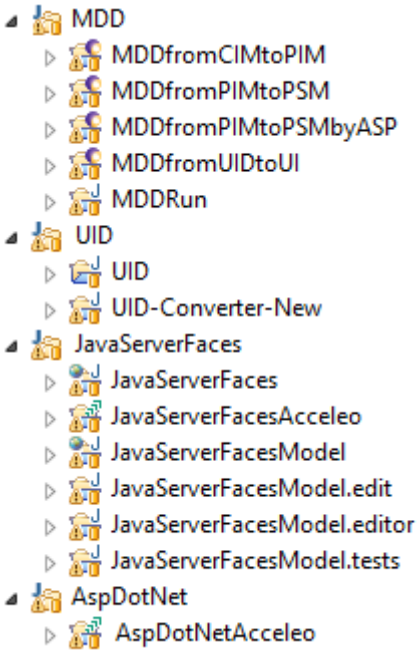

Figura 29 – Estrutura de projetos da aplicação

Os projetos da seção MDD estão relacionados ao processo de transformação dos modelos (*model-driven development*, MDD).

A seção UID contém os projetos de modelagem e conversão dos UIDs. Ou seja, o projeto UID permite que o usuário crie seus UIDs e os salve, e o projeto UID-Converter-New faz a conversão do código do UID no padrão de código utilizado pelo *framework* de transformação de modelos utilizado, como explicado na seção 2.2.5 Eclipse Modeling Framework.

Por fim, as seções JavaServerFaces e AspDotNet possuem os projetos modelo utilizados como base para a transformação Acceleo e os projetos destino, aqueles que possuem as interfaces geradas.

O primeiro projeto a ser considerado é o MDDRun, que contém a classe que inicia a execução do processo. Além disso, esse projeto contém a tela de seleção dos UIDs a serem executados, como mostra a Figura 30.

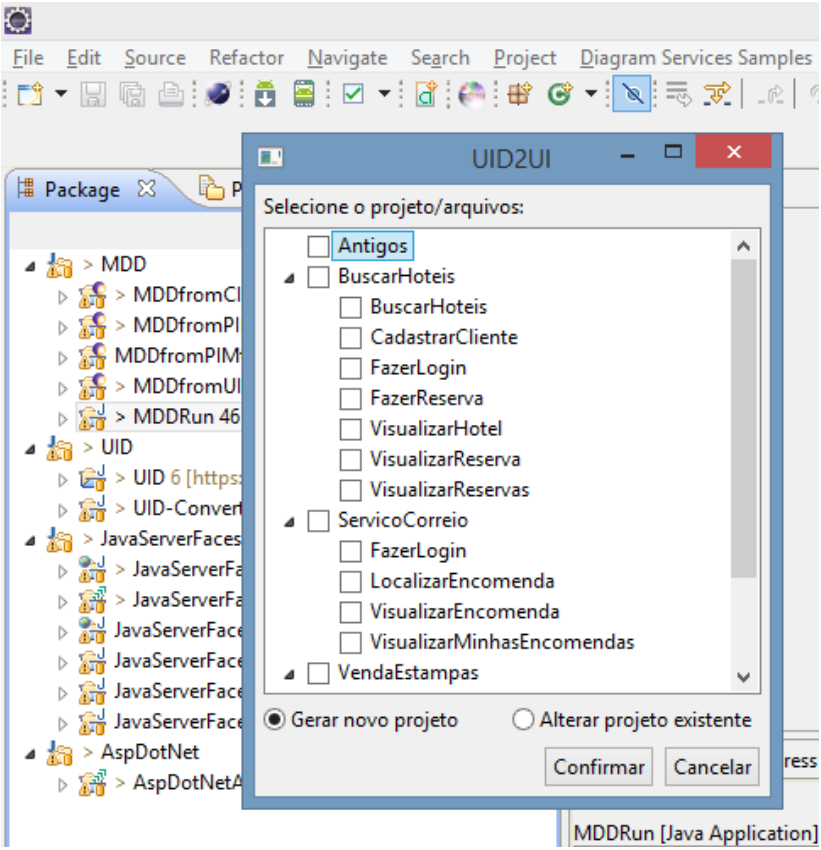

Figura 30 – Tela de seleção de UIDs

Para executar a classe inicial através da IDE Eclipse, com o lançador da classe configurado, é possível executá-lo através do menu de lançadores, conforme pode ser visto na Figura 31.

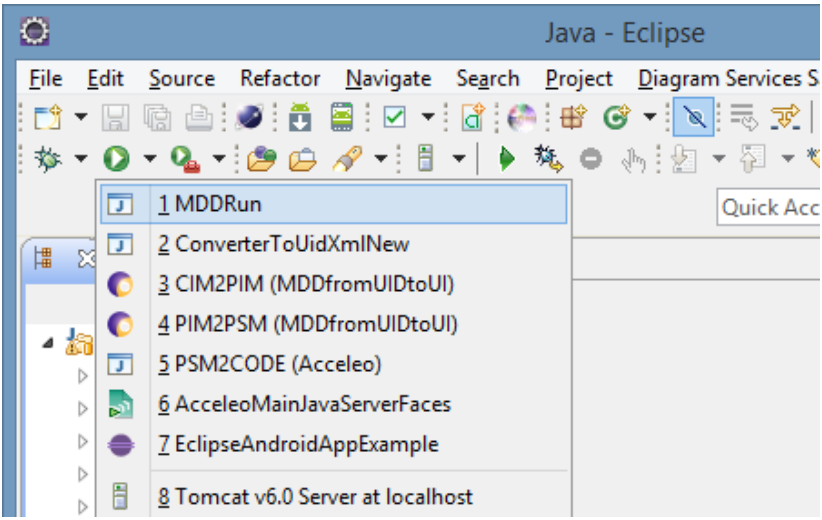

Figura 31 – Botão de execução da aplicação

Antes de se executar a transformação, é necessário criar um UID. Para isso, foi utilizado a ferramenta gráfica para UID desenvolvida por SCHOEPPING (2007). Esta ferramenta é um conjunto de plugins na IDE Eclipse. Assim, no projeto UID, basta criar um UID através da opção do menu de criação de arquivos da IDE, conforme mostrado na Figura 32.

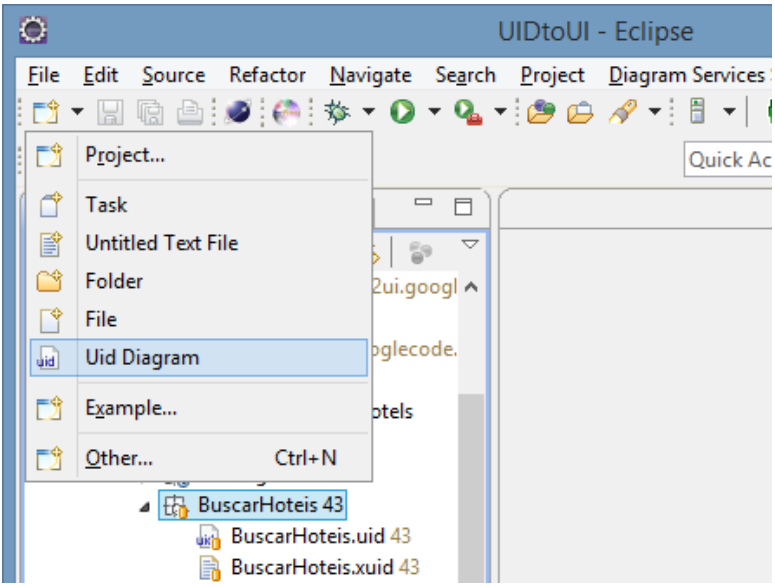

Figura 32 – Botão de criação de UID

Então, na tela principal, é possível criar e editar o UID, conforme apresentado na Figura 33. Na paleta da direita, estão os elementos do UID que podem ser utilizados para formar o UID. Na paleta inferior, estão as propriedades do elemento selecionado do UID da área principal e, na esquerda, estão os arquivos UIDs dentro das pastas do projeto UID.

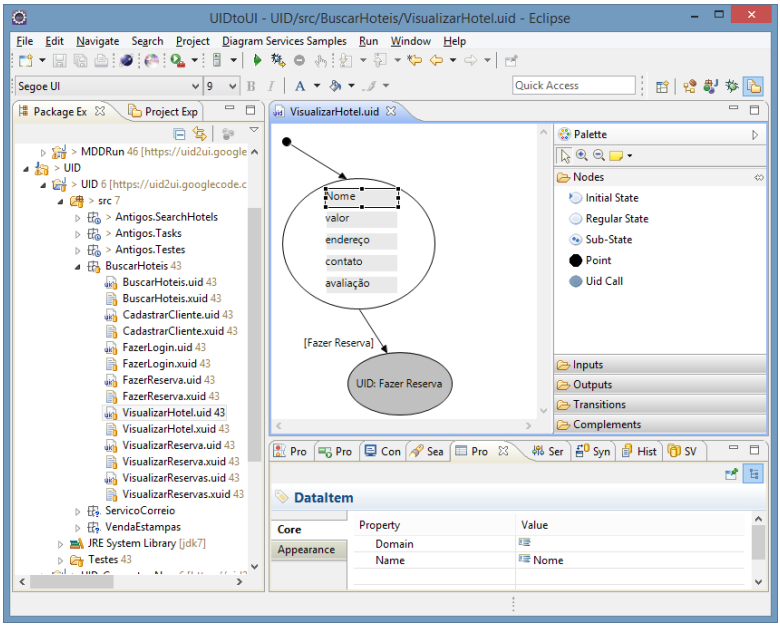

Figura 33 – Tela de criação de UID

Criado um ou mais UIDs que formam uma aplicação, conforme as necessidades do cliente, e executado o processo de transformação conforme explicado anteriormente, o projeto da aplicação final é gerado na pasta JavaServerFaces e na pasta AspDotNet para a geração para as tecnologias JSF e ASP.NET. A Figura 34 mostra essas pastas e uma das classes do projeto JSF.

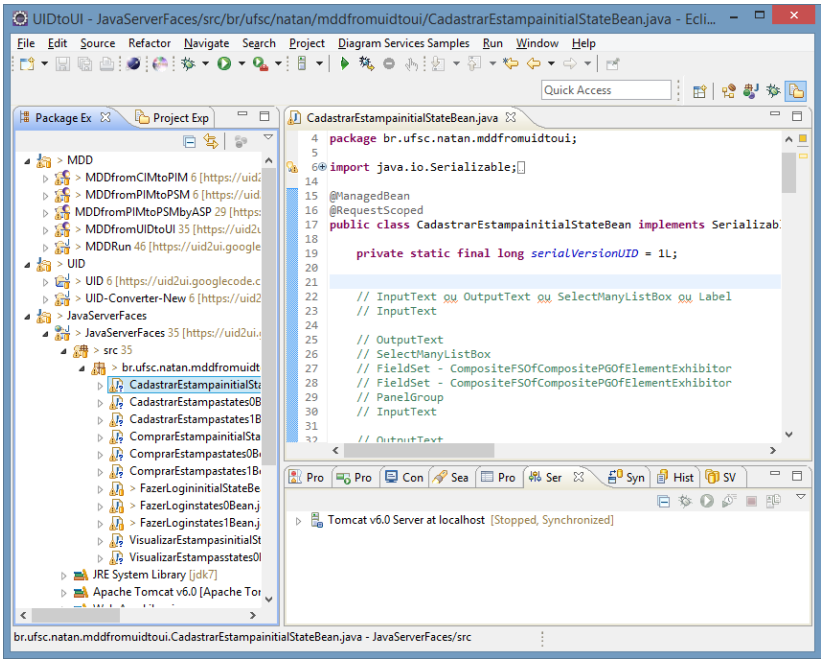

Figura 34 – Classe gerada automaticamente no pacote destino da aplicação

Por fim, basta rodar o servidor de aplicação relacionado. Para a aplicação JSF, pode ser utilizado o Tomcat. Na IDE Eclipse, o Tomcat pode ser executado através da aba Server, como também mostra a Figura 34. Para a aplicação ASP.NET, o projeto deve ser aberto pela IDE Visual Studio da Microsoft, onde pode ser executado.

Uma última funcionalidade a ser destacada é a possibilidade de alteração de um projeto sem a perda das alterações realizadas no código gerado. No processo de validação dos requisitos levantados, é comum surgirem novos requisitos, e assim torna-se necessário acrescentar outros campos e transições. Foi, então, desenvolvida uma opção para isso. Essa opção pode ser vista na tela, já apresentada (Figura 30), de seleção dos UIDs a serem gerados, através das opções “Gerar novo projeto” e “Alterar projeto existente”. A segunda opção é responsável por essa funcionalidade. Ela considera os elementos que existiam no UID em sua última geração e apenas gera os códigos relacionados aos elementos novos. Uma restrição à funcionalidade é a não remoção e nem a edição

de elementos já gerados, uma vez que não é possível identificar o que foi alterado no arquivo de código final gerado.

4.8 RESULTADOS

Conforme explicado na seção anterior, as aplicações destino são geradas através da execução da classe principal do projeto de controle. Depois disso, elas podem ser acessadas através de sua inicialização conforme definido por suas tecnologias, ou seja, inicializando o servidor de aplicações, como o Tomcat ou o Microsoft IIS, e acessando o endereço da aplicação *web* no *browser* (Figuras 35 e 36).

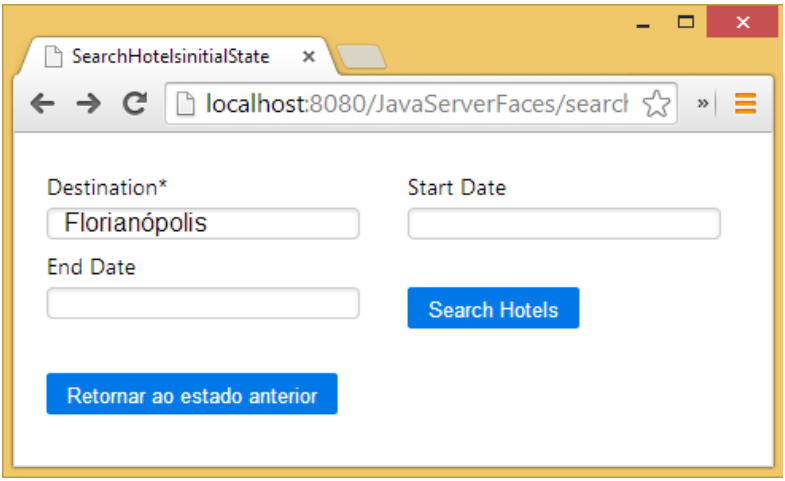

Figura 35 – Interface gerada a partir do estado <1>
do UID Buscar Hotéis

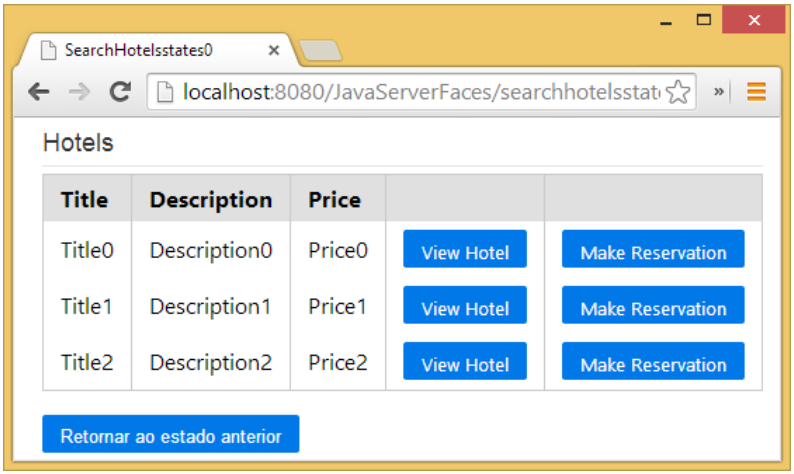

Figura 36 – Interface gerada a partir do estado <2> do UID Buscar Hotéis

As interfaces geradas foram comparadas com interfaces de aplicações reais de busca de hotéis (Figuras 37, 38, 39 e 40). Através dessas interfaces reais, pode ser visto que as interfaces geradas têm estrutura similar, conforme destacado nas imagens das interfaces de aplicações reais citadas.

Outra comparação que se pôde fazer foi quanto à troca de informações, onde as aplicações geradas apresentaram o funcionamento semelhante às aplicações reais nas tarefas de Buscar Hotéis, Visualização de Hotéis, Visualização de um Hotel e Realização de Reserva. E nessa mesma comparação pôde-se observar que a troca de informações entre o usuário e o sistema também foram semelhantes, o que pode ser visto nas áreas destacadas nas Figuras 37 a 40.

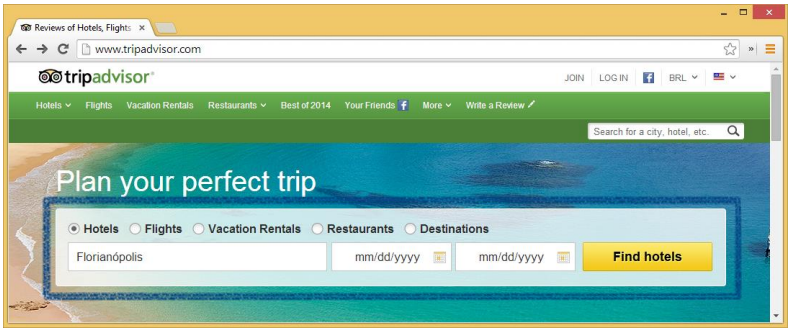

Figura 37 – Interface Tripadvisor[2] de comparação com o estado <1> do UID Buscar Hotéis

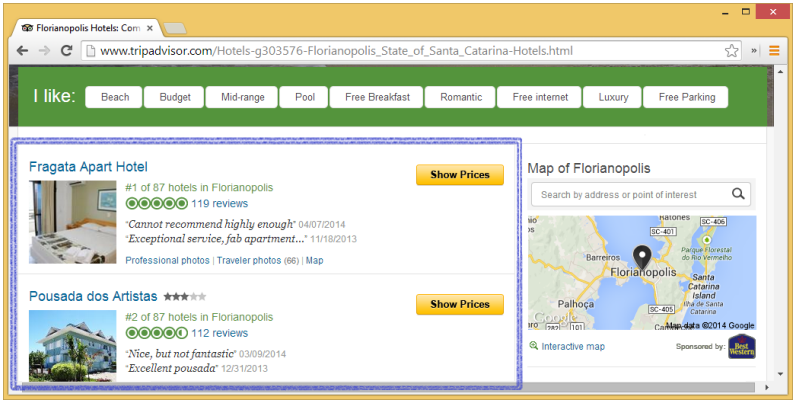

Figura 38 – Interface Tripadvisor[3] de comparação com o estado <2> do UID Buscar Hotéis

[2] http://www.tripadvisor.com
[3] http://www.tripadvisor.com/Hotels-g303576-Florianopolis_State_of_Santa_Catarina-Hotels.html.

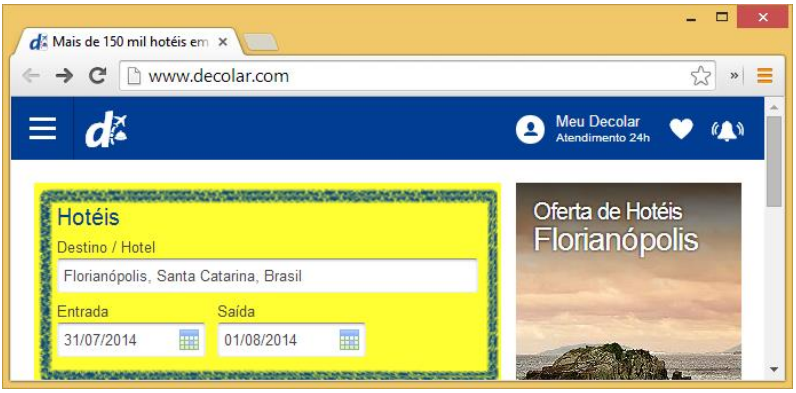

Figura 39 – Interface Decolar[4] de comparação com o estado <1> do UID Buscar Hotéis

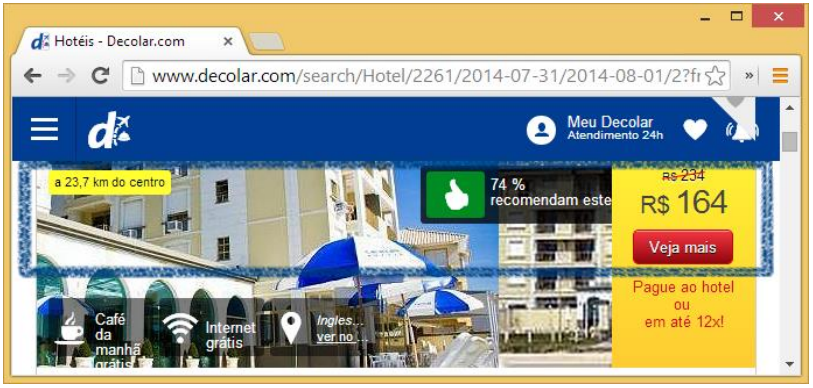

Figura 40 – Interface Decolar[5] de comparação com o estado <2> do UID Buscar Hotéis

[4] http://www.decolar.com.

[5] http://www.decolar.com/search/Hotel/2261/2014-07-31/2014-08-01/2?from=SB

5 ESTUDOS DE CASO

Além da comparação das interfaces geradas com interfaces de aplicações reais, a proposta foi avaliada através de dois estudos de caso, os quais foram desenvolvidos seguindo um fluxo básico de desenvolvimento iniciado pelo levantamento de requisitos, passando pela análise dos requisitos levantados e finalizando com a implementação do código do *software*.

O fluxo de desenvolvimento foi aplicado de duas formas diferentes. A primeira forma utilizada foi baseada na técnica de levantamento de requisitos a partir de Histórias de Usuário (HISTÓRIAS DO USUÁRIO XP, 2014). Os requisitos foram escritos pelo analista de *software* na forma de histórias, em contato direto com o cliente. Em seguida, foram revalidados com o mesmo cliente e, por último, foram utilizados para a implementação manual do código da aplicação. Depois de implementada a aplicação, divergências encontradas também permitiram a revalidação dos requisitos.

A segunda forma de aplicação do fluxo de desenvolvimento empregou o levantamento de requisitos através da utilização do UID pelo analista de *software*. Esses requisitos foram validados pelo cliente e, em seguida, foi gerado automaticamente o código pela ferramenta proposta.

O analista de *software* de ambos os fluxos de desenvolvimento foi o autor do presente trabalho e os clientes foram dois profissionais das áreas dos casos estudados.

Os casos estudados foram (i) um sistema para o serviço de correio e (ii) um site de apresentação e venda de estampas. Por fim, foram analisados os casos em relação à velocidade de desenvolvimento, entendimento e validação dos requisitos.

5.1 LOCALIZAÇÃO DE ENCOMENDAS EM UM SISTEMA PARA SERVIÇO DE CORREIO

Este caso diz respeito a funcionalidade de localização de encomendas em um sistema de correios. Quando um usuário que enviou uma encomenda ou fez uma compra que será enviada pelos correios deseja verificar a localização de sua encomenda, o status da entrega, o peso da mesma entre outras informações relacionadas à encomenda e sua localização.

5.1.1 Desenvolvimento manual baseado em histórias de usuário

Este caso tem como descrição a seguinte história:

> *"Eu, como dono dos correios, gostaria que o nosso sistema permitisse que nossos clientes encontrassem suas encomendas.*
>
> *Para isso creio que cada usuário precise ter um login e senha (para empresas que vendem na internet em grande quantidade), para ver todas as suas encomendas.*
>
> *E clientes esporádicos ou clientes dos nossos clientes (comprador do produto em uma empresa online) com apenas CPF ou código de rastreio devem poder ver a localização do pedido e o prazo de entrega, peso, dimensões e tipo de encomenda (Sedex, Sedex 10, Sedex Hoje, PAC e Carta Registrada)"*

Em relação à velocidade de desenvolvimento, foram necessários em torno de 15 minutos para escrever a história inicialmente e mais 10 minutos para validá-la com o cliente. Foram ainda necessárias por volta de 8 horas para fazer a implementação de uma aplicação básica em JSF contemplando a funcionalidade.

5.1.2 Desenvolvimento utilizando a ferramenta de geração automática de interfaces proposta

Os UIDs que descrevem os requisitos desse caso estudado são apresentados pelas Figuras 41 a 44. O primeiro UID (Figura 41) refere-se ao estado inicial da aplicação, quando o usuário pode entrar com as informações de pesquisa da sua encomenda, que são o código de rastreio ou o CPF; fazer *login* para visualizar suas encomendas, através da chamada do outro UID, FazerLogin; e visualizar suas encomendas, através da chamada do outro UID, VisualizaMinhasEncomendas.

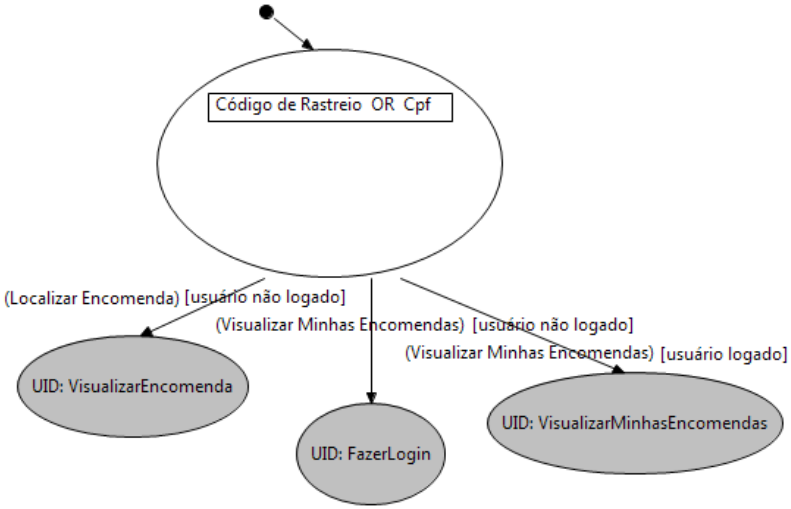

Figura 41 – UID de consulta de localização de encomendas.

O segundo UID (Figura 42), trata da visualização da encomenda. Ele é composto pela saída do sistema da estrutura de dado Encomenda, que possui os dados de uma encomenda, como nome, código de rastreio, localização, prazo de entrega e peso.

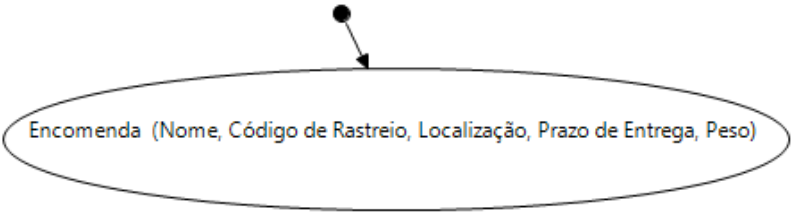

Figura 42 – UID de visualização localização de uma encomenda.

Outro UID deste caso é o de Login (Figura 43). Este UID descreve um estado inicial onde o usuário informa um *e-mail* e uma senha e pode logar com sucesso ou ter uma falha, sendo informado por mensagens exibidas pelo sistema. No caso de um *login* válido, o usuário não pode retornar ao estado anterior, pois a transição utilizada é unidirecional. Então o fluxo da aplicação segue normalmente do ponto de chamada deste UID.

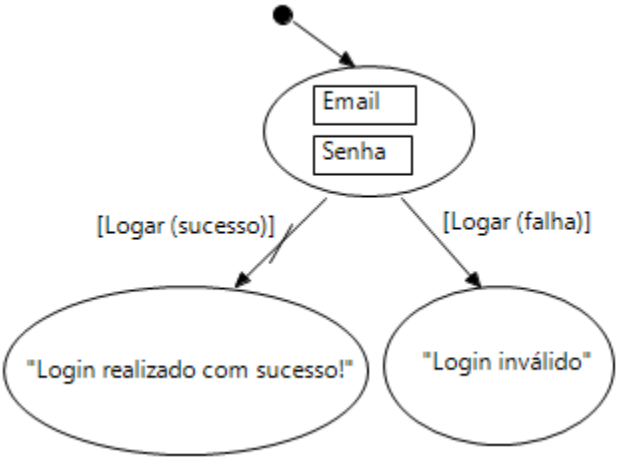

Figura 43 – UID de controle de *login*.

O último UID deste estudo é o de visualização de encomendas do usuário (Figura 44). Como pode ser visto no UID inicial, é necessário que o usuário esteja logado para que este UID seja acessado. Nesse estado da aplicação, o sistema exibe uma lista de encomendas, e o usuário pode visualizar cada encomenda individualmente, o que é feito acessando-se o mesmo UID, que foi chamado ao se localizar uma encomenda.

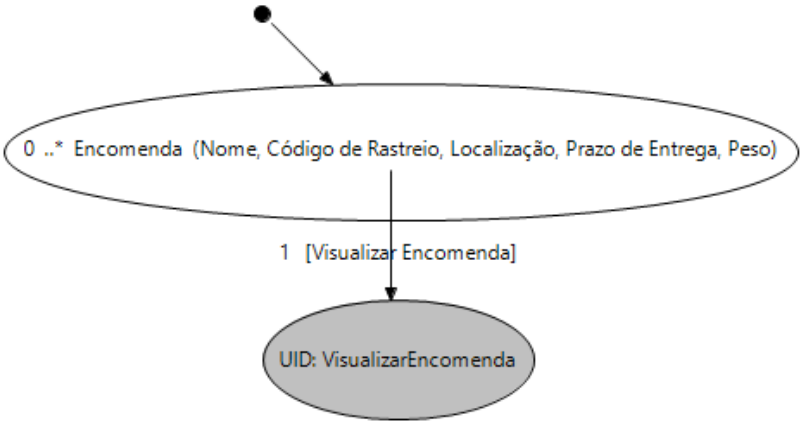

Figura 44 – UID de visualização de encomendas de usuário logado.

Para o desenvolvimento dos UIDs, foram necessários em torno de 25 minutos inicialmente e mais 20 para validá-los e corrigi-los. Enquanto que para o desenvolvimento, não se teve nenhum custo, uma vez que as interfaces foram geradas automaticamente e não exigem nenhuma alteração para seu funcionamento.

5.1.3 Interfaces desenvolvidas

As Figuras 45 a 52 representam as interfaces implementadas manualmente. Estas imagens representam também as interfaces geradas automaticamente, uma vez que, a implementação manual focou-se em gerar um código com as mesmas interfaces, para melhorar a comparação de velocidade de desenvolvimento.

O primeiro estado da aplicação é a busca de localização de encomendas (Figura 45). A partir deste estado, é possível localizar uma encomenda, caso o usuário esteja logado, fazer *login*, caso o usuário não esteja logado, ou visualizar as encomendas do usuário logado.

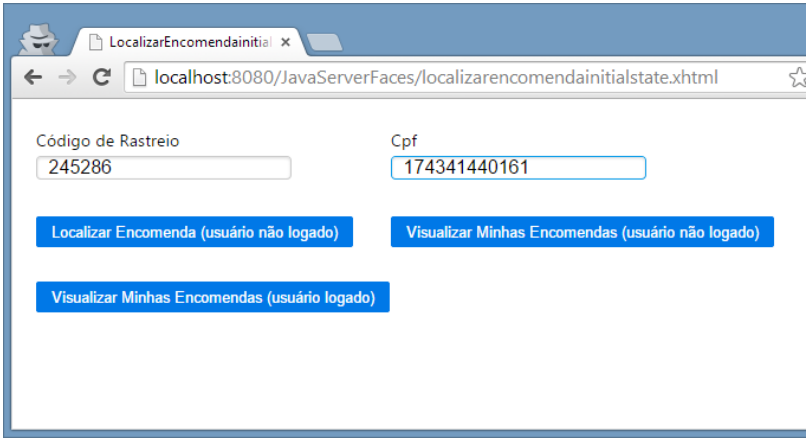

Figura 45 – Interface busca de localização de encomendas

A partir do estado inicial da aplicação, é possível localizar uma encomenda, conforme mostra a Figura 46.

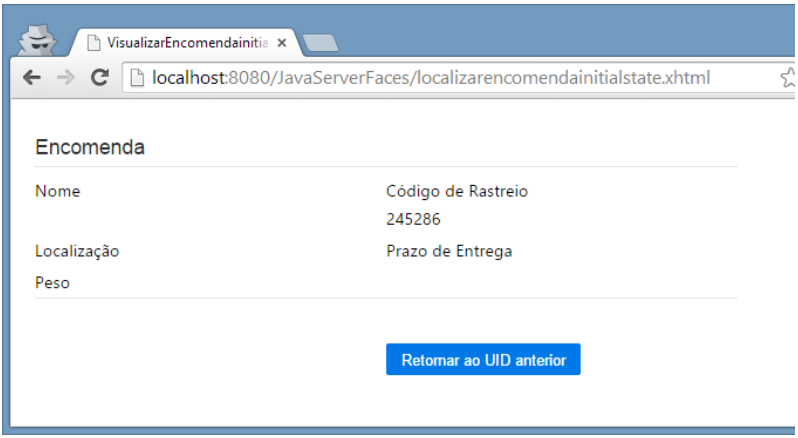

Figura 46 – Interface de localização de encomenda. Encomenda não encontrada.

Outro fluxo possível a partir do estado inicial é o *login* necessário à visualização das encomendas do usuário. A Figura 47 apresenta este estado quando o usuário não tiver preenchido os campos obrigatórios.

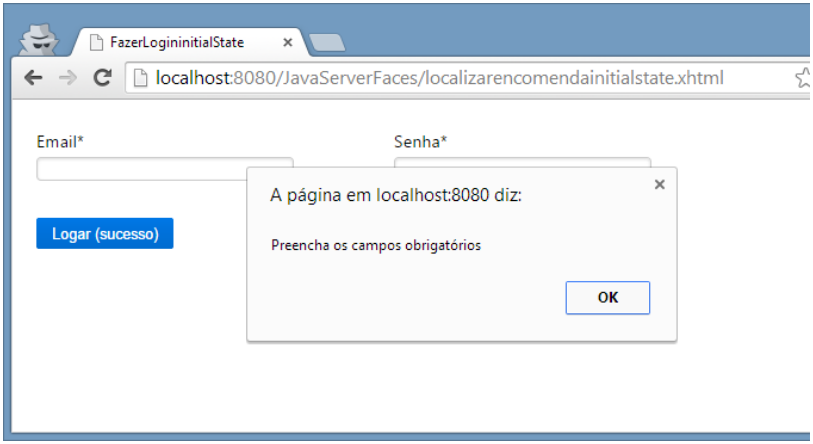

Figura 47 – Interface de *login* para acesso a visualização de encomendas de usuário logado. Campos obrigatórios não preenchidos.

A Figura 48 representa a interface de *login* com os campos preenchidos, com as opções de fluxo de *login* com sucesso e falha.

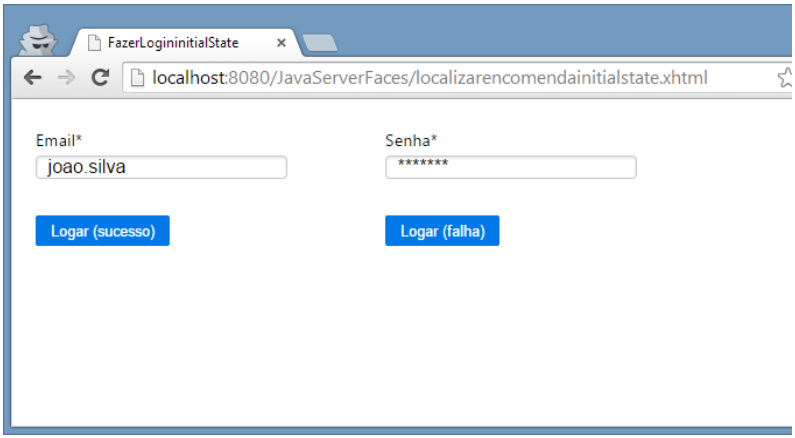

Figura 48 – Interface de *login* para acesso à visualização de encomendas de usuário logado. Campos preenchidos.

Caso o *login* não seja realizado com sucesso por alguma regra de negócio, por exemplo, o sistema irá exibir uma mensagem para o usuário (Figura 49). A partir desse estado, o usuário pode voltar ao estado de *login* ou ao estado anterior, a tela inicial de localização de encomendas.

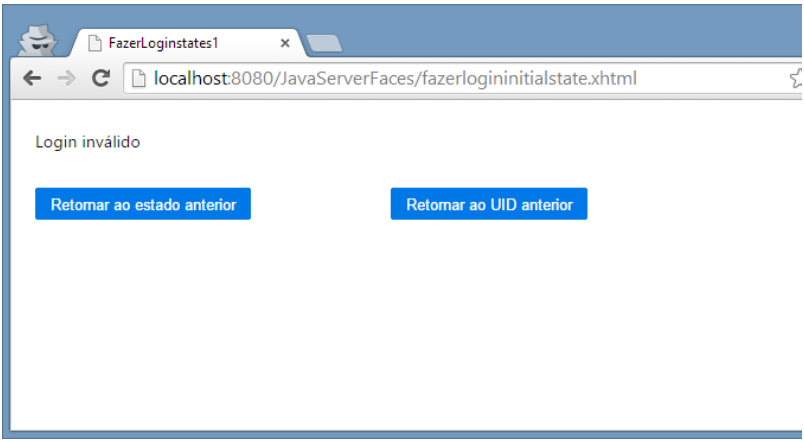

Figura 49 – Interface de *login* para acesso à visualização de encomendas de usuário logado. *Login* inválido.

Caso o *login* seja realizado com sucesso, o sistema exibe uma mensagem de sucesso e o usuário pode seguir com fluxo normal, retornando ao estado de localização de encomendas (Figura 50).

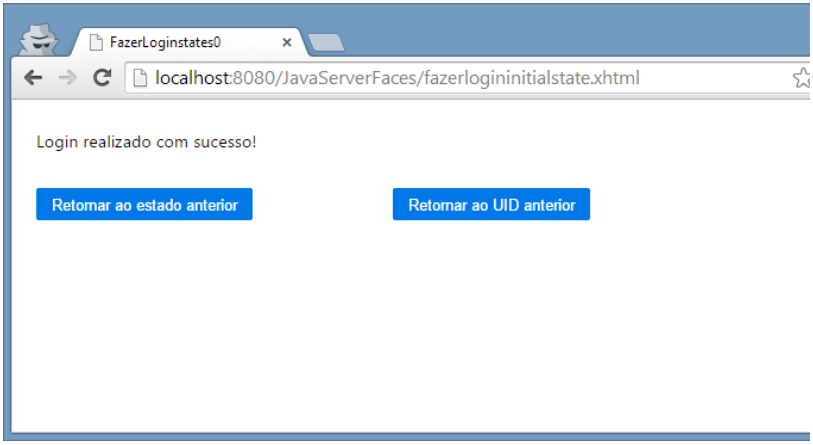

Figura 50 – Interface de *login* para acesso à visualização de encomendas de usuário logado. *Login* válido.

A Figura 51 representa o estado de visualização das encomendas do usuário logado. A partir deste estado é possível visualizar cada encomenda individualmente.

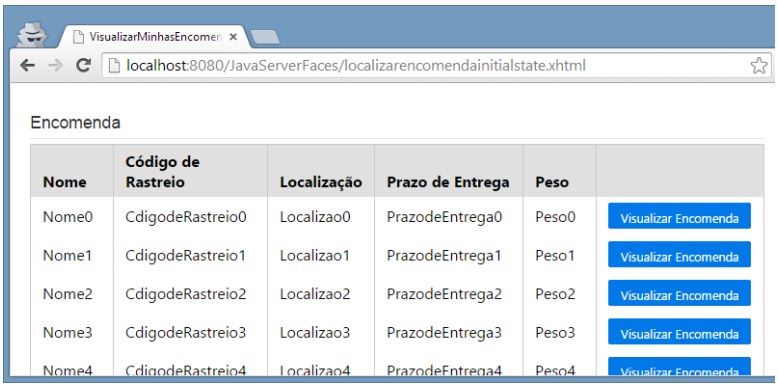

Figura 51 – Interface de visualização de encomendas de usuário logado.

O último estado é o de visualização de encomendas individuais, onde o usuário pode visualizar as informações da encomenda e voltar ao

estado anterior (interface de visualização de encomendas de usuário) ou ao UID anterior, aquele que chamou o estado anterior (a interface de localização de encomendas), conforme mostra a Figura 52.

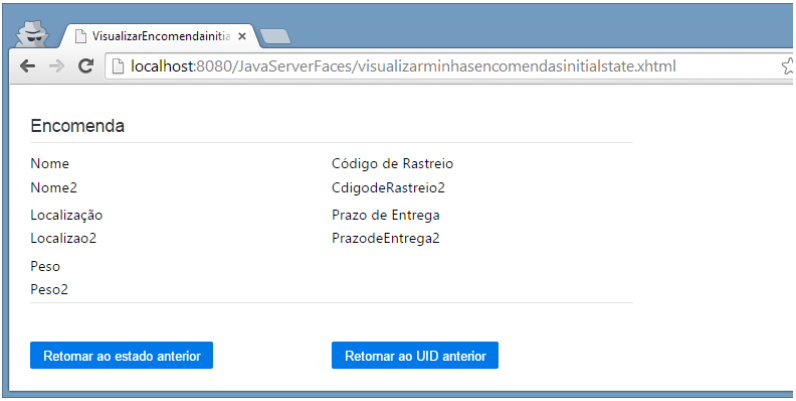

Figura 52 – Interface de localização de encomenda. Encomenda encontrada.

5.2 APRESENTAÇÃO E VENDA DE ESTAMPAS EM UM SITE

Este caso se refere ao cadastro, apresentação e vendas de estampas na internet. Existem duas funcionalidades, o vendedor tem a possibilidade de cadastrar suas estampas e os usuários podem visualizá-las e comprá-las.

5.2.1 Desenvolvimento manual baseado em histórias de usuário

Este caso tem como descrição a seguinte história:

"Eu, como desenvolvedora de estampas, gostaria de poder vender meus trabalhos pela internet. Para isso quero logar e cadastrar meus trabalhos categorizados por projetos, informando o nome da estampa, uma descrição, seu valor e projeto. E meus clientes poderão ver as estampas por projetos e comprá-las informando suas informações bancárias."

Em relação à velocidade de desenvolvimento, foram necessários em torno de 10 minutos para escrever a história e por volta de 6 horas para fazer a implementação de uma aplicação básica em JSF contemplando a funcionalidade.

5.2.2 Desenvolvimento utilizando a ferramenta de geração automática de interfaces proposta

As Figuras 47 a 49 mostram a representação do caso através dos UIDs. A Figura 53 apresenta o primeiro UID desse caso estudado. Nele o sistema apresenta uma lista de estampas e as opções de cadastrar estampas (chamada do UID CadastrarEstampas), fazer *login* (chamada do UID FazerLogin), comprar estampa a partir de uma estampa da lista de estampas (chamada do UID ComprarEstampas) e visualizar estampas do projeto, o que leva ao segundo estado do UID, onde o sistema exibe o projeto e a lista de estampas do projeto.

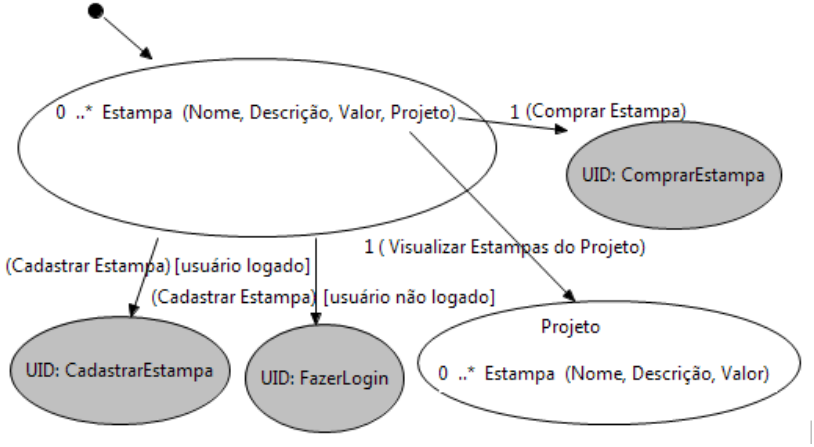

Figura 53 – UID de visualização de estampas.

A Figura 54 apresenta o UID de cadastro de estampas, uma das opções do UID anterior. No estado inicial desse UID, o usuário deve informar os dados de uma estampa a ser cadastrada, nome, descrição, valor e projeto. A partir desse estado, a aplicação pode ter dois fluxos, o cadastro com sucesso ou com falha. Em ambos os casos, o sistema exibe

uma mensagem, informando o usuário sobre o resultado da operação anterior.

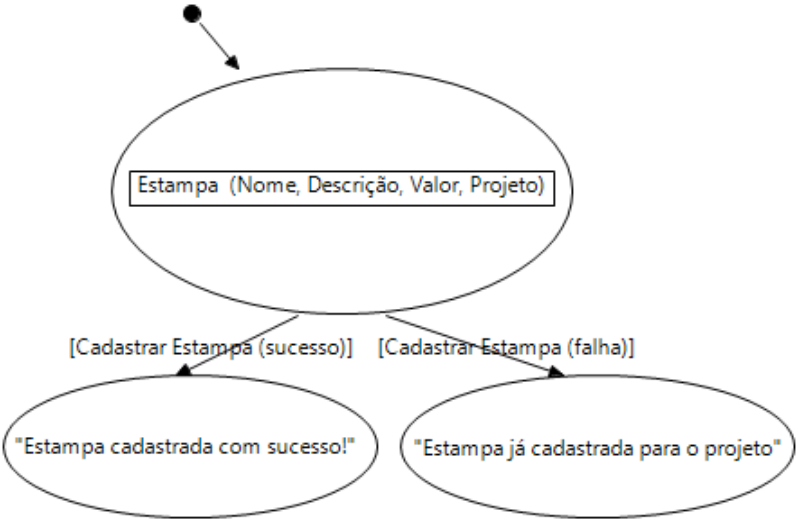

Figura 54 – UID de cadastro de estampa.

O último UID a ser apresentado neste estudo de caso é o de compra de estampa (Figura 55). Ele é chamado pelo UID inicial da aplicação, para uma estampa das estampas da lista. Nele o usuário deve informar os dados do cartão de crédito e, caso o cadastro ocorra com sucesso, o sistema exibe uma mensagem de sucesso. Caso o cadastro falhe, o sistema informa o usuário que o cadastro falhou.

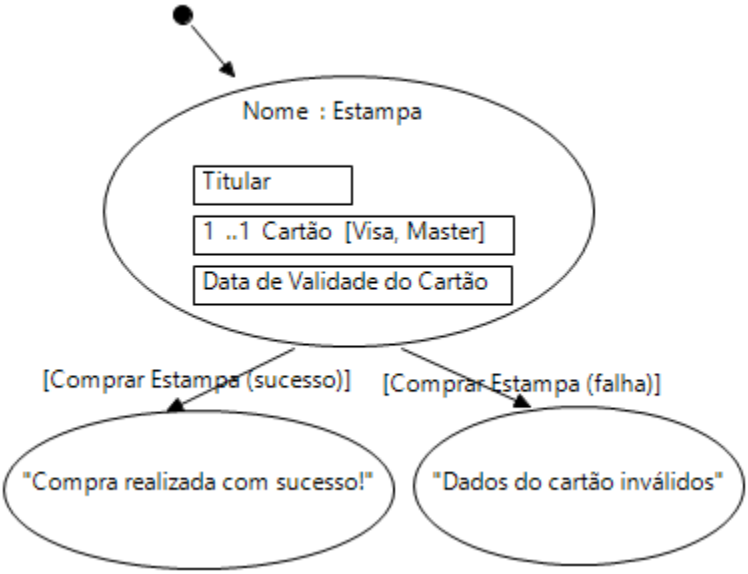

Figura 55 – UID de compra de estampa.

Para o desenvolvimento dos UIDs foram necessários em torno de 25 minutos inicialmente e mais 10 para validá-lo e corrigi-lo. Enquanto que como no caso anterior para o desenvolvimento, não se teve nenhum custo.

5.2.3 Interfaces desenvolvidas

As Figuras 56 a 62 representam as interfaces implementadas manualmente. Da mesma forma que no caso anterior, essas imagens representam também as interfaces geradas automaticamente, uma vez que a implementação manual focou-se em gerar um código com as mesmas interfaces, para melhorar a comparação de velocidade de desenvolvimento. Outro ponto relacionado ao caso anterior é que as interfaces do UID de *login* são as mesmas do caso já visto.

Conforme descrito no UID, o primeiro estado da aplicação é a apresentação das estampas (Figura 56). A partir desse estado, é possível cadastrar uma estampa (caso o usuário esteja logado), fazer *login* (caso o usuário não esteja logado, conforme já apresentado em mesmas interfaces

no primeiro caso estudado), visualizar as estampas de um projeto ou comprar uma estampa.

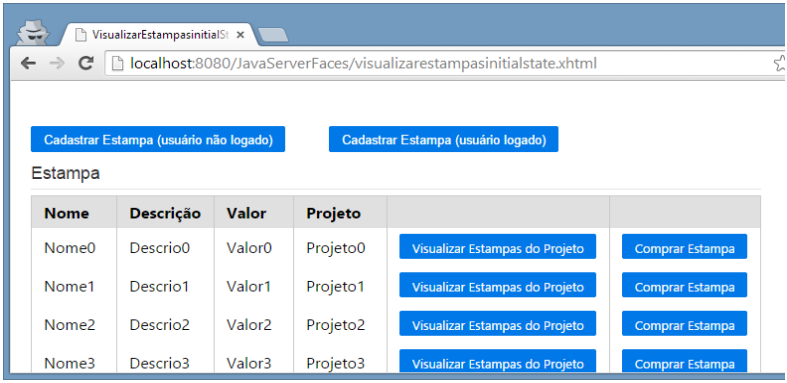

Figura 56 – Interface de visualização de estampas.

A Figura 57 mostra o estado de cadastro de uma estampa. A partir desse estado, é possível cadastrar a estampa com sucesso. Também pode haver falha por uma regra de negócio, ou mesmo o cancelamento do cadastro, voltando-se ao estado anterior.

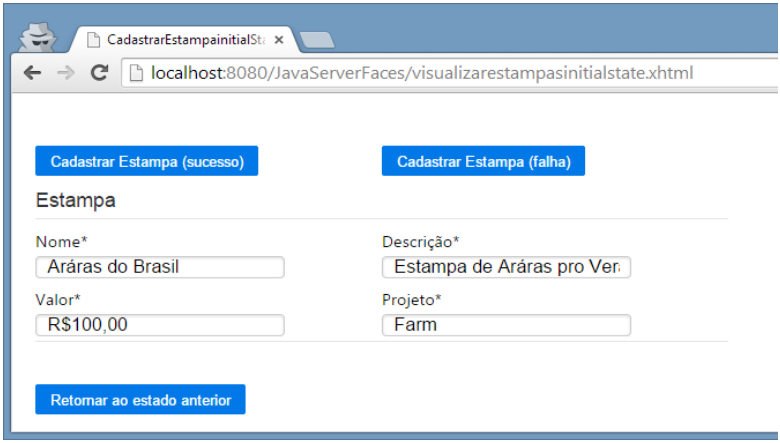

Figura 57 – Interface de cadastro de estampa.

Conforme definido no caso de uso, determinados campos são requeridos e outros são opcionais. Portanto, caso os campos obrigatórios

não sejam preenchidos, uma mensagem de validação é exibida (Figura 58).

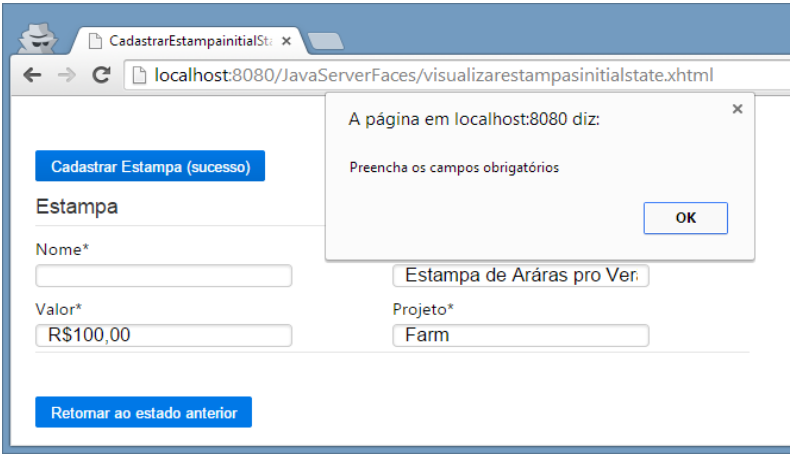

Figura 58 – Interface de cadastro de estampa. Campos obrigatórios não preenchidos.

Caso o cadastro de uma estampa falhe, a aplicação apresenta este estado com uma mensagem de falha ao usuário, conforme descrito no UID (Figura 59). A partir desse estado, o usuário pode seguir o fluxo normal, voltando ao estado do UID anterior (estado de visualização de estampas anterior ao estado de cadastro de estampa), ou pode voltar ao estado anterior do UID atual (cadastro de estampa).

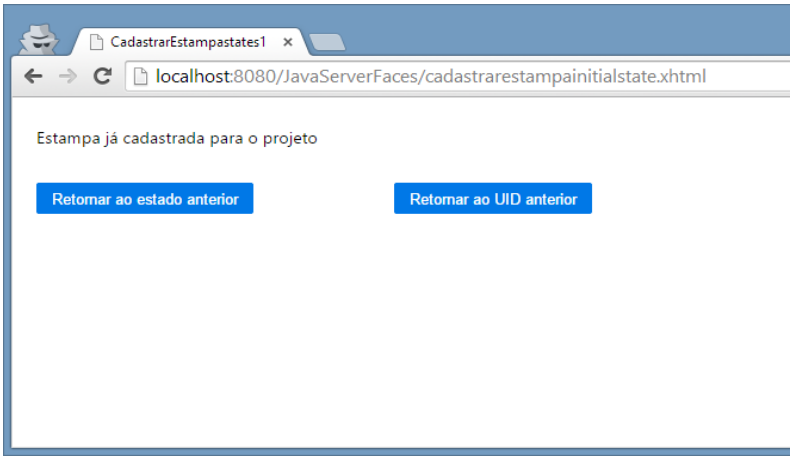

Figura 59 – Interface de cadastro de estampa. Cadastro inválido.

De maneira semelhante ao estado de falha, ao cadastrar uma estampa com sucesso, a aplicação apresenta este estado com uma mensagem de sucesso ao usuário, conforme descrito no UID (Figura 60). Nesse estado, o usuário também pode seguir o fluxo normal, voltando ao UID anterior, ou pode voltar ao cadastro da estampa, voltando ao estado anterior.

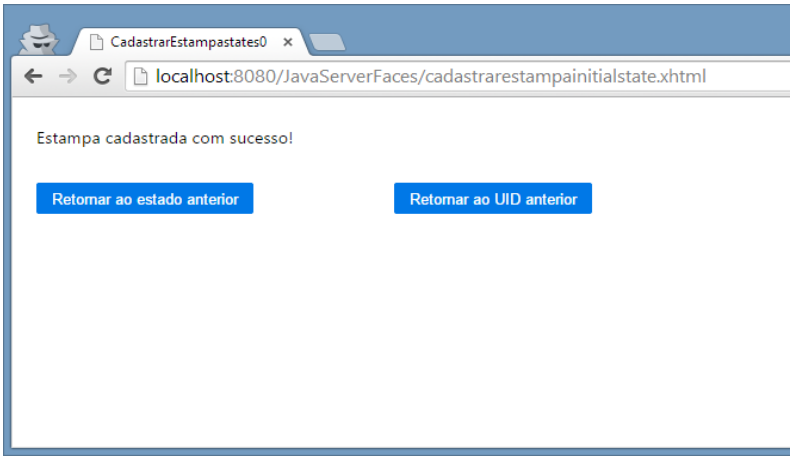

Figura 60 – Interface de cadastro de estampa. Cadastro válido.

A Figura 61 representa o estado de visualização de estampas de um projeto. Este estado era uma das opções de fluxo do estado inicial (apresentação das estampas). Conforme definido no UID, por opção do analista, este estado não permite a compra de estampa, por isso não há nenhum botão para isso.

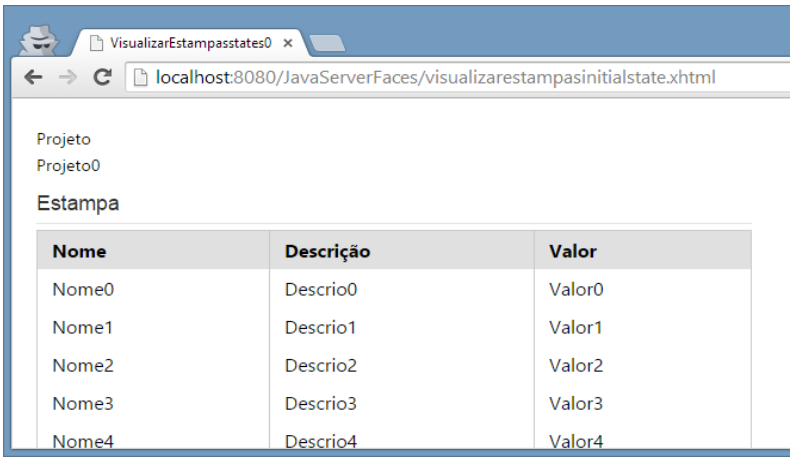

Figura 61 – Interface de visualização de estampa por projeto selecionado.

A última opção de fluxo do estado de visualização de estampa é o estado de compra de estampa, apresentado na Figura 62. Este estado permite a compra com sucesso, a compra em caso de falha e o cancelamento através do retorno ao estado anterior. Conforme descrito no UID, a compra com sucesso e a falha levam a telas com mensagens de sucesso ou falha, da mesma forma que o UID de cadastro de estampas.

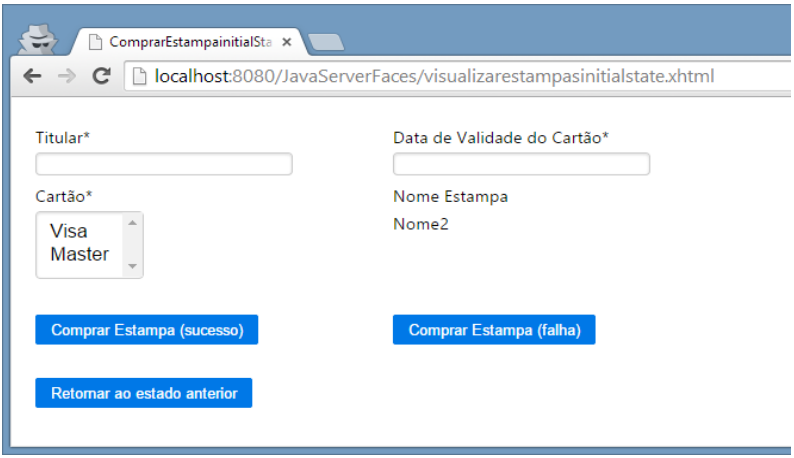

Figura 62 – Interface de compra de estampa.

5.3 RESULTADOS

Através dos casos estudados, foi possível avaliar a proposta em relação a alguns pontos principais, como: levantamento de requisitos, considerando o entendimento; qualidade e capacidade de aproveitamento do código gerado; tempo gasto.

O entendimento através da escrita da história foi considerado o ideal, pois é possível descrever todas as necessidades com quantos detalhes se quiser. Porém a utilização do UID, em ambos os casos, foi considerada de fácil entendimento mesmo para pessoas não relacionadas à área. Além disso, a visualização dos requisitos através dos fluxos descritos pelo UID se aproxima muito da implementação final, facilitando o entendimento.

A qualidade e a capacidade de aproveitamento do código gerado foram consideradas razoáveis. O código gerado não contém lógicas, porém contempla toda a navegação descrita pelos requisitos levantados, expressos através dos UIDs, e permite que sejam utilizados como ponto de partida para o desenvolvimento final do sistema.

Por fim, com relação à velocidade de desenvolvimento, verificou-se que existe um bom ganho de produtividade, uma vez que os tempos gastos para a escrita da história e do UID se assemelham. Por sua vez, o desenvolvimento de uma aplicação básica que contemple os requisitos descritos é muito superior, obviamente, se comparado à

geração automática. Além disso, a ferramenta permite a adição de componentes e alteração do código gerado, sem perda de código já manualmente alterado. Isso é bastante útil na etapa de validação, uma vez que os requisitos vão sendo mais bem entendidos e modificados.

Para finalizar, a Tabela 1 mostra o tempo de cada etapa de cada estudo de caso.

Tabela 1 – Comparativo dos tempos de desenvolvimento

Tipo de desenvolvimento	Tempo etapa de levantamento de requisitos (minutos)	Tempo etapa de validação de requisitos (minutos)	Tempo etapa de implementação (minutos)	Tempo total (minutos)
Localização de encomendas em um sistema para serviço de correio				
Proposta deste trabalho	25	20	0	45
Histórias de usuário	15	10	480	505
Apresentação e venda de estampas em um site				
Proposta deste trabalho	25	10	0	35
Histórias de usuário	10	0	360	370

6 CONCLUSÃO

Este trabalho apresenta uma abordagem para geração automática de interfaces com o usuário através da metodologia de desenvolvimento dirigido a modelos. A geração é realizada a partir do modelo independente de computação (CIM), nesse caso, o Diagrama de interação com o usuário (UID). Ela ocorre através da transformação desse modelo para o modelo independente de plataforma (PIM). Em seguida, o PIM é transformado para o modelo específico de plataforma (PSM), para então ser transformado em código.

A abordagem foi avaliada através do caso de Busca de Hotéis usando as plataformas JSF e ASP.NET. Mostrou-se que a ferramenta é capaz de gerar interfaces funcionais, representando corretamente a troca de informações entre o usuário e o sistema. Também foi possível observar que, em comparação com interfaces de aplicações existentes na internet, as interfaces geradas apresentaram campos e fluxos semelhantes.

Além do caso demonstrado, foram feitos dois estudos de caso, um de "Localização de encomendas em um sistema para serviço de correio" e outro de "Apresentação e venda de estampas em um site". Através desses estudos de caso, foi possível avaliar a proposta como positiva em termos de auxílio no entendimento e auxílio no levantamento de requisitos. A proposta também se mostrou proveitosa em relação ao código gerado, útil como ponto de partida para o desenvolvimento final e também produtivo, conforme foi apresentado na Tabela 1, uma vez que uma boa parte do produto final é gerada automaticamente.

Assim, a proposta mostrou contribuições significantes, como a possibilidade de se usar a geração automática de páginas como protótipo ou como um ponto de partida para a implementação do código. Outra contribuição demonstrada foi o levantamento de requisitos, uma vez que ele permite a rápida validação com o usuário final, além de facilitar o entendimento dos desenvolvedores.

6.1 TRABALHOS FUTUROS

No decorrer da pesquisa foram identificados outros desafios fora do escopo da proposta que poderiam ser adicionados a ela evoluindo-a ainda mais. Entre esses desafios, pode-se mencionar os seguintes para trabalhos futuros:

- Definição de disposição de layout dos componentes de interface no UID, de forma que, possa ser levado através das transformações para o código das interfaces geradas.
- Implementação de regras de transições OCL no fluxo de navegação entre as transições com o objetivo de aumentar a funcionalidade do projeto final, contemplando regras de negócio.

Por fim, pode-se citar que este trabalho gerou uma publicação de um artigo em uma conferência da área:

ZEFERINO, Natan Vinícius; VILAIN, Patrícia. A model-driven approach for generating interfaces from user interaction diagrams. Proceedings - 16th International Conference on Information Integration and Web-based Applications & Services. ACM, 2014. p. 474-478.

REFERÊNCIAS

AMELLER, David. *Considering non-functional requirements in model-driven engineering.* 2009. 82 f. Tese (Màster en Computacion) – Departament de Llenguatges i Sistemes Informàtic. Universitat Politècnica de Catalunya. Barcelona, Espanha. 2009.

ATKINSON, Colin; KUHNE, Thomas. Model-driven development: a metamodeling foundation. *Software, IEEE*, 2003, 20.5: 36-41.

BÉZIVIN, Jean. On the unification power of models. *Software & Systems Modeling*, 2005, 4.2: 171-188.

CZARNECKI, Krzysztof; HELSEN, Simon. Classification of model transformation approaches. *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*. 2003. p. 1-17. Disponível em: <http://www.ptidej.net/courses/ift6251/fall05/presentations/050914/Czarnecki_Helsen.pdf>. Acesso em: 10 set. 2013.

DAMIANI, F.B., VILAIN, P. Automatic generation of web interfaces from user interaction diagrams. *Proceedings of SEKE.* 2012. p. 605-610.

DI RUSCIO, Davide. *Specification of model transformation and weaving in model driven engineering*. 2007. PhD Thesis in Computer Science – Universita degli Studi dell'Aquila (February 2007). Disponível em: <http://www.di.univaq.it/diruscio/phdThesis.php>. Acesso em: 20 jul. 2013.

ECLIPSE. Acceleo. Disponível em: < http://www.eclipse.org/acceleo/>. Acesso em: 13 jul. 2013.

FRAGAL, Vanderson H. *Técnicas de geração de código utilizando LPS e MDE para sistemas embarcados*. 2013. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Maringá, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior. 2013.

FRATERNALI, P.; PAOLINI, P. Model-driven development of Web applications: the AutoWeb system. *ACM Transactions on Information Systems (TOIS)*, 18(4), 323-382, 2000.

GÓMEZ, J.; CACHERO, C.; PASTOR, O. Conceptual Modeling of Device-Independent Web Applications. *IEEE Multimedia*, 8(2), 26-39, 2001.

HISTÓRIAS DO USUÁRIO XP. Disponível em: <http://www.softwarepublico.gov.br/5cqualibr/xowiki/historiasUsuario> . Acesso em Dezembro de 2014.

KLEPPE, Anneke G.; WARMER, Jos B.; BAST, Wim. *MDA explained, the model driven architecture: Practice and promise*. Addison-Wesley Professional, 2003.

KROISS, Christian; KOCH, Nora; KNAPP, Alexander. Uwe4jsf: A model-driven generation approach for web applications. *Web Engineering*. Springer Berlin Heidelberg, 2009. p. 493-496.

LOUHICHI, Soumaya; GRAIET, Mohamed; KMIMECH, Mourad; GAALOUL, Walid; BHIRI, Mohamed Tahar; CARIOU, Eric. ATL Transformation for the Generation of SCA Model. In: *Semantics Knowledge and Grid (SKG), 2011 Seventh International Conference on*. IEEE, 2011. p. 164-167.

MEDEIROS, Ana Luisa Ferreira de. *MARISA-MDD: uma abordagem para transformações entre modelos orientados a aspectos: dos requisitos ao projeto detalhado*. 2008. 112 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal do Rio Grande do Norte, Natal, 2008.

MEIXNER, Gerrit; PATERNÒ, Fabio; VANDERDONCKT, Jean. Past, Present, and Future of Model-Based User Interface Development. i-com Zeitschrift für interaktive und kooperative Medien, 2011, 10.3: 2-11.

MELIÁ, Santiago; GÓMEZ, Jaime; PÉREZ, Sandy; DÍAZ, Oscar. A model-driven development for GWT-based Rich Internet Applications with OOH4RIA. In: *Web Engineering, 2008. ICWE'08. Eighth International Conference on*. IEEE, 2008. p. 13-23.

MELLOR, Stephen J.; CLARK, Anthony N.; FUTAGAMI, Takao. Model-driven development. *IEEE software*, 2003, 14-18.

MOURA, Sabrina Silva de; SCHWABE, Daniel. Interface Development for Hypermedia Applications in the Semantic Web. Joint Conference. *10th Brazilian Symposium on Multimedia and the Web & 2nd Latin American Web Congress*, Ribeirao Preto, SP, Brazil. pp 106-113, IEEE Computer Society, October 2004.

MYERS, Brad A. *Why are human-computer interfaces difficult to design and implement*. No. CMU-CS-93-183. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1993.

MENDIX. Disponível em: <http://www.mendix.com>. Acesso em: 25 jul. 2013.

OBEO. Disponível em: <http://www.obeo.fr/pages/atl-pro/en>. Acesso em: 25 jul 2013.

OMG. MDA – the architecture of choice for a changing world. Disponível em: <http://www.omg.org/mda>. Acesso em: 12 jul. 2013.

ROTHENBERG, J. 1989. A Rand note; N-3027-DARPA p.18 The Nature of Modeling. Santa Monica, California. Disponível em: <www.rand.org/pubs/notes/2007/N3027.pdf>. Acesso em: 22 jul. 2013.

SCALISE, Eugenio G.; FAVRE, Jean-Marie; ZAMBRANO, Nancy. Model-driven reverse engineering and program comprehension: an example. *Ingeniare. Rev. chil. ing.*, Arica, v. 18, n. 1, abr. 2010. Disponível em: <http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-33052010000100009&lng=es&nrm=iso>. Acesso em Janeiro de 2014.

SCHMIDT, Douglas C. Guest editor's introduction: Model-driven engineering. *Computer*, 2006, 39.2: 0025-31.

SCHOEPPING, Guilherme. Projeto e Implementação de uma Ferramenta Gráfica para UID. 2007. 74 f. Trabalho de Conclusão de Curso; (Graduação em Bacharelado em Ciências da Computação) - Universidade Federal de Santa Catarina, Florianópolis, 2007.

SELIC, Bran. The pragmatics of model-driven development. *Software, IEEE*, 2003, 20.5: 19-25.

VILAIN, Patrícia. Modelagem da interação com o usuário em aplicações hipermídia. 2002. Tese de Doutorado. Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2002.

WATERS, John K. Eclipse Futures on Tap at Annual Conference. *VisualStudio Magazine*. 23/9/2009. Disponível em: <http://visualstudiomagazine.com/articles/2009/03/23/eclipse-futures-on-tap-at-annual-conference.aspx>. Acesso em: 13 jul. 2013.

WEBRATIO. Disponível em: <http://www.webratio.com>. Acesso em 20 jul. 2013.