UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

GUSTAVO SCHMID DE JESUS

Variabilidade em Interfaces de Usuário em Linhas de Produto de Software baseadas na Web: um Estudo Exploratório

Trabalho de Graduação.

Prof. Dr. Ingrid Oliveira de Nunes
Orientador

Porto Alegre, janeiro de 2014

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

de Jesus, Gustavo Schmid

Variabilidade em Interfaces de Usuário em Linhas de Produto de Software baseadas na Web: um Estudo Exploratório / Gustavo Schmid de Jesus. – Porto Alegre: Graduação em Ciência da Computação da UFRGS, 2014.

52 f.: il.

Trabalho de Conclusão (bacharelado) – Universidade Federal do Rio Grande do Sul. BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO, Porto Alegre, BR–RS, 2014. Orientador: Ingrid Oliveira de Nunes.

1. Linha de produto de software. 2. Interface de usuário. 3. Automatização. 4. Model based user interface design. I. de Nunes, Ingrid Oliveira. II. Título.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

SPL	Software Product Line
SPLC	Software Product Line Conference
PFE	Product Family Engineering
SPLS	Software Product Line Series
UI	User Interface
MBUID	Model Based User Interface Design
CBGUI	Constrained Based Graphic User Interface
HTML	Hypertext Markup Language
XML	Extensible Markup Language
PG	Presentation Group
PU	Presentation Unit
GUI	Graphical User Interface

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Linhas de Produto de Software (LPS) é uma recente e promissora abordagem para desenvolvimento de famílias de sistemas que compartilham um núcleo comum, promovendo o reuso em larga escala e consequentemente alguns benefícios tais como menor tempo e custo de desenvolvimento. A principal característica de uma LPS é a variabilidade de sistemas que ela pode produzir, considerando as diferentes funcionalidades que um sistema pode possuir. Entretanto, existe uma área que é pouco explorada que é a variabilidade em interfaces de usuário (IU). O grande diferencial da variabilidade, nesse contexto, é que a combinação de funcionalidades cria um número exponencial de configurações. Desta forma, projetar uma interface adequada para todas as configurações é inviável do ponto de vista prático.

Este trabalho consiste de uma pesquisa de um novo algoritmo utilizando como base um estudo exploratório para identificar os principais desafios e limitações das abordagens atuais no contexto de variabilidade de IU's em LPS's. Utilizando como base a implementação de uma LPS baseada em um sistema Web, é explorada a utilização de técnicas atuais para a implementação dessa variabilidade neste caso de uso particular em conjunto de algumas modificações feitas nestas técnicas para otimizar este processo. Com base no estudo desenvolvido, discutem-se quais lições foram aprendidas, os pontos positivos e negativos da técnica desenvolvida, uma conclusão sobre o estudo realizado e possíveis direções para trabalhos futuros que queiram contribuir nesta área de pesquisa.

Palavras-chave: Linha de produto de software, interface de usuário, automatização, model based user interface design.

1 INTRODUÇÃO

Neste trabalho apresenta-se uma pesquisa de um novo algoritmo que busca otimizar o conjunto atual de algoritmos responsáveis por criar a IU de produtos de uma LPS de forma semi-automática. Analisa-se as principais técnicas existentes com o intuito de identificar os principais problemas de cada uma, com isto é sugerido uma nova técnica que busque solucionar tais problemas.

1.1 Contextualização e Motivação

Linha de Produto de Software (LPS) é uma abordagem recente na engenharia de software com sua primeira conferência datando de 1996 (Product Family Engineering - PFE) na Europa e 2000 (Software Product Line Series - SPLS) nos EUA, sendo representado atualmente pela Software Product Line Conference (SPLC) (Software Product Line Conference 2013). LPS pode ser visto como um conjunto de algoritmos e modelos de engenharia de software, utilizados para produzir um sistema capaz de produzir uma família de sistemas, possibilitando a criação de diferentes *Software Products* que possuam características em comum. Devido a essa família de sistemas, sua principal característica é a variabilidade de seus produtos. Essa variabilidade cria a necessidade de gerenciamento das diferentes *features* presentes na LPS, tratando assim da variabilidade de configurações possíveis que forma cada produto único entre seus irmãos.

Devido a necessidade de gerenciar produtos com diferentes conjuntos de *features*, surge um problema que consiste em definir um método de produção de software de modo que seja criado de forma sistemática e semi-automática, o produto escolhido por um conjunto de *features*. Considerando que cada produto é formado por uma parte lógica e outra visual, podemos dividir o problema original em dois, que seriam:

- Derivação da lógica do produto
- Derivação da IU[1] do produto

De acordo com (Hallvard 2002), para construir a parte lógica do produto existem técnicas já comprovadas que produzem bons resultados, como por exemplo *Compilação Condicional*, *Programação Orientada a Feature*, *Padrões de Design* entre outras. Entretanto quando se considera a IU percebe-se que existem técnicas automáticas, mas o resultado produzido por elas não é satisfatório para grande maioria dos aplicativos existentes.

[1] Interface de Usuário

1.2 Estado-da-arte e Limitações

Atualmente, para produzir boas IU's (intuitivas e de fácil uso) a solução atual que produz o melhor resultado é utilizar pessoas especializadas, por exemplo projetistas de software. Essas pessoas possuem o papel de executar as modificações necessárias de cada IU manualmente para cada produto, criando-se assim uma dependência do produto com esse projetista, pois quando for necessário modificar algo na LPS que modifique a interface de algum produto, não existirá maneira de garantir a funcionalidade do produto pois não existe um rastreamento da *feature* com a interface modificada.

Na tentativa de automatizar esse processo de derivação de produtos temos como principal área de pesquisa a Engenharia de Software dirigida a Modelos (MDSE, do inglês Model-Driven Software Engineering) (19), que quando aplicado a LPS trata da Engenharia de Linha de Produto de Software dirigida a Modelo (MDSPLE, do inglês Model-Driven Software Product Line Engineering) (2).

Os algoritmos que utilizam modelos são muito eficientes para derivar a parte lógica dos produtos e até mesmo interfaces de aplicações mais simples como formulários, mas apresentam limitações quando os produtos requerem um nível de customização muito grande aliado de uma boa usabilidade. Podemos citar como exemplo de aplicativo que necessita de muitas customizações um jogo para celular que deve considerar os diferentes tipos de celulares, com suas entradas de dados, tamanho e resolução de telas distintas, entre outros fatores.

1.3 Este trabalho

Neste estudo exploratório é feito um estudo utilizando uma LPS desenvolvida para a plataforma *Web* como uma aplicação de exemplo. Apresenta-se primeiro alguns dos principais problemas do processo de derivação de IU's dos produtos da LPS e após uma técnica que tentará solucionar estes problemas.

Utilizando os conceitos de desenvolvimento baseado em modelos, com foco na técnica MBUID (do inglês, Model Based User Interface Design) será discutido uma possibilidade de integração entre a técnica desenvolvida e o MBUID.

Como principal fonte de motivação deste estudo temos o artigo (Pleuss et al. 2012) que apresenta os diferentes produtos de uma LPS com seus diferentes níveis de modificação manual da IU devido aos diferentes requisitos do mesmo sistema. Esse estudo mostrou que a área de derivação de IU de modo automatizado necessita de mais pesquisa e experimentação.

1.4 Objetivos

Por se tratar de um estudo exploratório, os objetivos focam na pesquisa de novos métodos que busquem solucionar problemas já existentes e instigar novas fronteiras de investigação na área de automatização de IU's para LPS's.

Como primeiro objetivo tem-se a criação e experimentação de um novo método para criação semi-automática de IU's para LPS's. Primeiramente é feito uma discussão dos principais problemas encontrados na produção automática de IU dos métodos utilizados atualmente. Após apresentam-se os problemas que essas técnicas se propõem a resolver seguido da técnica proposta neste trabalho para solucionar alguns desses problemas.

O segundo e final objetivo é mostrar os resultados da técnica utilizada, de modo que

com estes resultados seja possível detectar pontos positivos e negativos do método proposto para que ele possa auxiliar com novas direções para pesquisas e incitar novas investigações no processo de automatização de IU's para LPS's.

1.5 Estrutura

Apresenta-se a seguir a estrutura dos capítulos deste trabalho, sumarizando o que é tratado em cada um.

Capítulo 2 Apresenta-se a fundamentação teórica necessária para o entendimento deste estudo e uma visão resumida dos principais processos e vantagens da utilização de LPS.

Capítulo 3 Apresenta-se a aplicação utilizada como estudo de caso seguido da identificação dos problemas que foram investigados.

Capítulo 4 Apresenta-se a técnica desenvolvida neste estudo, apresentando suas restrições de uso, seus modelos e seu algoritmo.

Capítulo 5 Apresenta-se os desafios identificados no desenvolvimento de IU's e as lições aprendidas no estudo exploratório realizado.

Capítulo 6 Conclue-se o estudo apresentado retomando o contexto do estudo de caso desenvolvido, seguido das contribuições da técnica apresentada e por fim apresenta-se as sugestões de pesquisa para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os principais conceitos e ferramentas utilizados no desenvolvimento do estudo de caso e da técnica desenvolvida para implementar o gerador semi-automático de IU para LPS.

Para um melhor entendimento deste trabalho, é necessário uma introdução dos principais conceitos que são tratados neste estudo. Explica-se o suficiente para o entendimento deste estudo, para maiores informações é recomendado a pesquisa da bibliografia indicada em cada tópico.

2.1 Linha de Produto de Software

De acordo com (Clements e Northrop 2001), LPS é definida por um conjunto de sistemas de software que compartilha um conjunto comum de *features* gerenciáveis que satisfaz uma necessidade específica de um segmento de mercado ou missão e é desenvolvida a partir de um conjunto comum de *features* de maneira prescrita.

Com esta definição em mãos, pode-se verificar que um dos principais requisitos da LPS é criar e gerenciar essa variabilidade de produtos que podem ser produzidos automaticamente com a escolha das *features* desejadas. Para definir-se uma LPS é necessário a implementação de dois processos sobre a família de produtos escolhida. Apresenta-se agora os processos com uma breve explicação sobre cada um.

Engenharia de Domínio Nesse processo é definido e criado os aspectos comuns e variáveis da LPS. Os principais artefatos criados nessa etapa são: (1) um *Modelo de Features* (Kang et al. 1990), (2) um conjunto de *Artefatos do Domínio*, e (3) um *Mapeamento de Features* que relaciona *features* com seus respectivos artefatos de domínio.

Engenharia de Aplicação Nesse processo é feita a derivação dos possíveis produtos da LPS. Este processo é composto por: (1) uma *Configuração do Produto* que é uma lista das *features* selecionadas para o produto desejado, e (2) o processo de *Derivação do Produto* caracterizado por utilizar o mapeamento das *features* com a configuração do produto para derivar o produto final utilizando-se dos artefatos de domínio. Esse processo normalmente é mais trivial se comparado com a Engenharia de Domínio.

De acordo com (Hauptmann et al. 2010), outras vantagens da utilização de LPS é a redução no esforço de desenvolvimento, redução no custo dos produtos e aumento na qualidade do software. Apresenta-se a seguir os fatores que tornam essas vantagens possíveis.

2.1.1 Reuso

De acordo com (Frakes e Fox 1995), reuso possibilita diminuir os custos de desenvolvimento e ao mesmo tempo aumentar a qualidade do software, além disso evitamos criar código extra para realizar a mesma tarefa em sistemas diferentes. A qualidade do software aumenta pois o código é utilizado e por consequência executado e testado diversas vezes em sistemas distintos, aumentando assim sua confiabilidade. O reuso também facilita a manutenção do software pois ao se detectar uma falha em uma *feature*, basta resolver o problema uma vez que a solução se replica para todos os produtos.

2.1.2 Manutenção

De acordo com (Stafford 2003), manutenção de software é toda modificação feita depois que o produto está pronto. É uma tarefa recorrente para qualquer sistema visto que os requisitos estão em constante mudança. Com a utilização de LPS e por consequência de reuso de software, não é mais necessário a manutenção separada de cada sistema quando trabalhamos com as funcionalidades obrigatórias, pois todos os sistemas filhos utilizam a mesma base de desenvolvimento. A manutenção de uma funcionalidade afeta todos os filhos da LPS que compartilham aquela funcionalidade, facilitando a manutenção de múltiplos sistemas e diminuindo o tempo e a taxa de erros que naturalmente pode ocorrer cada vez que se modifica um programa.

2.1.3 Custo

Utilizando como base os estudos apresentandos em (Pohl e Linden 2005), LPS além de poupar custos com reuso e facilidade na etapa de manutenção dos sistemas, ocorre uma diminuição de custo quando consideramos que para cada novo sistema, basta selecionar as funcionalidades desejadas que a LPS se ocupa de criar o produto final. Deve-se perceber que o custo de desenvolvimento da plataforma da LPS possui um custo relativamente alto, deste modo um ganho de custo só ocorre por volta do terceiro sistema desenvolvido. Na Figura 2.1 apresenta-se um gráfico relacionando o custo acumulado com o número de sistemas desenvolvidos, comparando sistemas desenvolvidos separadamente com sistemas desenvolvidos utilizando uma LPS.

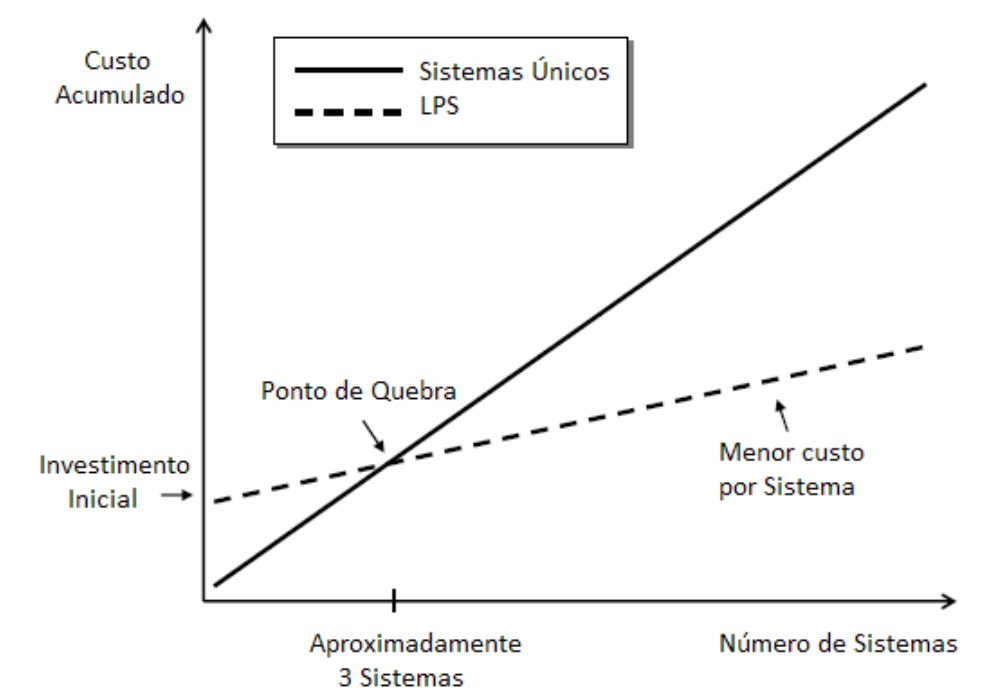

Figura 2.1: Custo acumulado X número de sistemas, fonte: (Pohl e Linden 2005)

Apresenta-se agora alguns trabalhos relacionados que discutem possíveis soluções para a criação de IU de forma automática para LPS.

2.2 Trabalhos Relacionados

Como motivação deste estudo tem-se (Pleuss et al. 2012), o qual apresenta exemplos de produtos de uma LPS e o nível de customização manual na IU que cada produto possui. No mesmo ano que foi publicado esse estudo, o mesmo autor publicou o artigo (Pleuss et al. 2012) onde ele apresenta uma maneira semi-automatizada de modelar a customização de cada produto mantendo a rastreabilidade de cada elemento visual da IU com a sua respectiva *feature*. A técnica utilizada nesse artigo faz parte da Engenharia de Linha de Produto de Software dirigida a Modelo (MDSPLE, do inglês Model-Driven Software Product Line Engineering).

Desenvolvimento de IU baseado em modelos é um conjunto de conceitos da área de interação humano-computador (IHC) e Engenharia de Software (ES) para modelar IU‘s. MBUID é um algoritmo introduzido pela comunidade de IHC que utiliza diferentes tipos de modelos para focar em conceitos diferentes em níveis de abstração diferentes. Na Figura 2.2 é apresentado o diagrama da solução resumida do algoritmo MBUID.

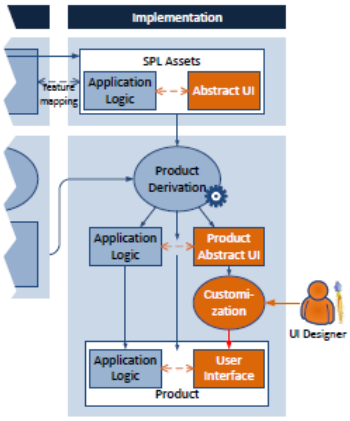

Figura 2.2: Exemplo de solução baseada em modelos (MBUID)

Na Figura 2.3 são apresentados os modelos utilizados no estudo feito por (Pleuss et al. 2012), com as respectivas transformações entre modelos até a construção da IU do produto.

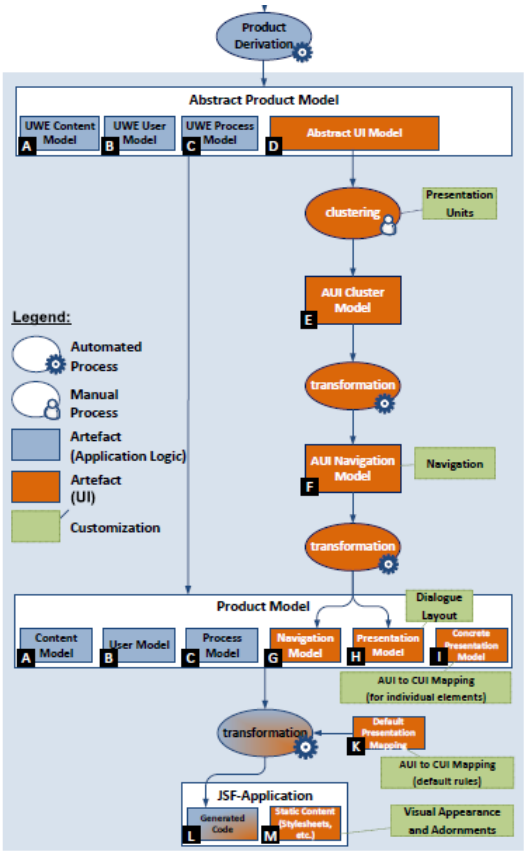

Figura 2.3: Representação dos modelos e de suas transformações da solução baseada em modelos (MBUID), fonte: (Pleuss et al. 2012)

A técnica MBUID consiste na criação de um modelo abstrato da IU do produto que se deseja derivar, a partir deste modelo inicial são feitas transformações para outros modelos e escolhas em cada modelo que possibilitam a customização de alguns atributos de interface. Uma discussão completa com exemplos sobre os modelos e algoritmos contidos no método MBUID está presente em (Hauptmann et al. 2010).

2.3 Tecnologias

A seguir, apresenta-se as principais tecnologias utilizadas no desenvolvimento deste estudo de caso.

2.3.1 S.P.L.O.T

S.P.L.O.T (17) é uma aplicação para a plataforma *web* que facilita a criação e edição de *modelos de features*, além de outras ferramentas como a configuração de produtos baseada em algum modelo do seu repositório. Na Figura 2.4, apresenta-se a IU do sistema S.P.L.O.T com suas principais funcionalidades.

Figura 2.4: Principais funcionalidades da ferramenta S.P.L.O.T

Essa ferramenta foi escolhida por não ter custos para utiliza-lá além de ser de fácil uso.

2.3.2 Ruby e Ruby on Rails

Ruby é uma linguagem interpretada utilizada para programação orientada a objetos, criada por Yukihiro Matusomoto. Ele mesmo a qualifica como sendo "simples, direto ao ponto, extensível e portável". Ruby foi desenvolvida com o objetivo de criar uma linguagem que utilizasse os melhores aspectos de cada uma das mais famosas linguagens. Com uma sintaxe simples e limpa, Ruby foi parcialmente inspirada por *Eiffel* e *Ada* enquanto que o tratamento de exceções possue similaridades com *Java* e *Python*. A Figura 2.5 traz um exemplo de código escrito em Ruby retirado da implementação deste trabalho.

```
#sets the position of a PG
  #copy its appearances in currentPositionLIst to finalPositionList
  def definePgPosition(pg,finalPositionList,currentPositionList)
    currentPositionList.each_with_index do |pos,index|
          if currentPositionList[index] and pos.id == pg.id
            finalPositionList[index] = pg
          end
        end
    return finalPositionList
  end

  #returns true if the position is not occupied
  def positionIsFree?(position,positionList)
    if position > 0
          positionList[position].nil? ? true : false
        else
          false
        end
  end

  #returns true if the PG position has already been defined
  def pgIsDefined?(pg,finalPositionList)
    finalPositionList.each_with_index do |pos,index|
          if finalPositionList[index] == pg
            return true
          end
        end
        return false
  end
```

Figura 2.5: Exemplo de código em Ruby retirado do trabalho

Ruby foi a linguagem de programação escolhida para desenvolver a implementação deste trabalho pois possui um framework para desenvolvimento de aplicações para *web* muito prático e de ampla divulgação atualmente, o Ruby on Rails. Para maiores informações sobre Ruby, a principal referência se encontra em (Matsumoto 2013).

Ruby on Rails é um framework para a linguagem Ruby, tendo seu foco no desenvolvimento de aplicações para a *web*, sendo considerado hoje um dos mais produtivos frameworks do seu segmento (Hibbs e Tate 2006). Ele utiliza o padrão de projeto MVC [1], este por sua vez é um padrão de arquitetura de software muito utilizado para aplicativos da plataforma *web* devido a sua estrutura arquitetural que modela bem o esquema cliente/servidor.

Os principais recursos do RoR [2] são:

Meta-programação Esta técnica serve para criar programas que por sua vez criam outros programas, por exemplo, o que ocorre em uma LPS.

[1] Model View Controller
[2] Ruby on Rails

Active Record Funciona como uma camada de abstração ao acesso ao BD[3], permitindo que com uma só linguagem seja possível a descrição do BD para plataformas distintas.

Convenção sobre configuração Como o próprio nome já diz, utilizando-se de convenções perde-se menos tempo configurando o ambiente e mais tempo criando código, bastando para isso apenas conhecer os padrões do RoR.

2.3.3 XML

"eXtensive Markup Language", ou XML é uma linguagem de programação extensiva, ou seja, pode-se definir novos elementos utilizando apenas marcadores do tipo «"e »". Ela é muito utilizada para descrever arquivos que contenham uma grande quantidade de informação e que estejam organizados da maneira necessária e especificada pelo usuário. O uso mais comum do XML é definir um formato de arquivo (DTD, do ingles Data Type Definition) e após cria-se um programa que faça a leitura destes arquivos. XML é utilizado principalmente com a funcionalidade de ser uma forma de intercambio de informação padrão, desta maneira todos podem utilizar uma mesma linguagem para trocar informação.

Para maiores informacoes sobre XML e tecnologias afins, recomendo a referência (4). Na Figura 2.6 apresento um arquivo XML, neste caso se trata de um arquivo que armazena os produtos de um café da manha, com seus respectivos preços e valores calóricos.

[3]Banco de Dados

```
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <calories>900</calories>
  </food>
  <food>
    <name>Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    <calories>900</calories>
  </food>
  <food>
    <name>French Toast</name>
    <price>$4.50</price>
    <calories>600</calories>
  </food>
  <food>
    <name>Homestyle Breakfast</name>
    <price>$6.95</price>
    <calories>950</calories>
  </food>
</breakfast_menu>
```

Figura 2.6: Exemplo de código em XML, fonte: (XML code example 2013)

Escolheu-se XML pois é um padrão bem conhecido de linguagem de descrição de arquivos e apresenta várias ferramentas que auxiliam na leitura e criação de arquivos com sua notação.

2.3.4 HTML

HTML é a sigla para *Hypertext Markup Language*, que traduzido para o português significa "Linguagem de marcação para Hipertexto". Hipertexto é o nome utilizado para designar páginas da *web* que possuam grande quantidade de caracteres e dados em geral.

HTML é a principal linguagem utilizada para descrever páginas e portanto IU para sistemas da plataforma *web*, por isso foi escolhida para implementar a IU neste trabalho. Outro exemplo bem conhecido e muito utilizado para atualizar páginas em HTML é a linguagem *Javascript*. *Javascript* executa no lado do cliente modificando a visualização da interface sem que uma requisição extra ao servidor da página seja necessária. A seguir, na Figura 2.7, um exemplo de código escrito em HTML, neste caso é uma página da *web* com um título e alguns títulos e parágrafos.

```
<HTML>
<HEAD>
<TITLE>Titulo da pagina</TITLE>
</HEAD>
<BODY BGCOLOR="FFFFFF">
<CENTER><IMG SRC="clouds.jpg" ALIGN="BOTTOM"> </CENTER>
<a href="http://somegreatsite.com">link</a> link para outro site
<H1>Isso e um cabecalho</H1>
<H2>Isso e um cabecalho medio</H2>
Mande-me email para
<a href="mailto:support@exemplo.com"> support@exemplo.com </a>
<P> Isso e um novo paragrafo!
<P> <B> Isso e um novo paragrafo, em negrito! </b>
</BODY>
</HTML>
```

Figura 2.7: Exemplo de código em html, fonte: (HTML code example 2013)

2.4 Considerações Finais

Neste capítulo se apresentou os principais conceitos e tecnologias que trabalhados no desenvolvimento deste estudo. Apresentou-se resumidamente o que constitue uma LPS, seus dois principais processos, seus artefatos de software e as principais vantagens de seu uso. Viu-se também um trabalho relacionado que trata sobre desenvolvimento baseado em modelos (Hauptmann et al. 2010) que utiliza a técnica MBUID, além de ferramentas e tecnologias relacionadas a esse trabalho.

No próximo capítulo é apresentado o estudo de caso composto por uma LPS, utilizando exemplos de produtos para mostrar os problemas que este estudo se propõe a solucionar.

3 ESTUDO DE CASO

Neste capítulo se apresenta o estudo de caso que é composto por uma LPS que será implementada. Finaliza-se o capítulo com os desafios de IU investigados neste estudo.

3.1 Escopo

Devido a natureza deste trabalho, deve-se escolher uma aplicação que sirva de estudo para teste da técnica proposta. A LPS escolhida é denominada *Webmail* e é composta por um cliente de emails para a plataforma *web*. Escolheu-se essa aplicação por dois motivos principais, que são:

1 - Grande número de usuários Clientes de email são ferramentas muito comuns atualmente, pois a necessidade de um correio virtual é cada dia maior, por isto o número de usuários está em constante aumento. Além disso, praticamente não existe restrição para o tipo de usuário dessas ferramentas, tem-se então desde crianças a idosos utilizando-a. Esses dois fatores dão um grande suporte para futuras avaliações da técnica proposta no capítulo 4 pois existe o potencial de opiniões e requisitos de diversos tipos de usuários.

2 - Grande número de *features* Por ser uma ferramenta utilizada com frequência, é comum a integração de outras ferramentas no cliente de email, por exemplo sistemas de mensagem instantânea, calendários, repositório de arquivos, entre outros. Essa característica possibilita um alto nível de variabilidade dos produtos possíveis o que aumenta o desafio da técnica e facilita encontrar problemas relacionados a IU.

Na Figura 3.1 é apresentado um exemplo de um cliente de email para plataforma *web*, o aplicativo *Gmail* popularmente conhecido com diversas *features* integradas.

Para realizar este estudo escolheu-se a plataforma *web* pois a motivação se originou nos estudos realizados com uma LPS de uma ferramenta de gerência de universidades que possui diversas modificações na sua IU devido a requisitos distintos de cada universidade. Além disso, o autor deste trabalho possui experiência com a linguagem Ruby e com o framework RoR[1].

Como visto na seção 2.2, este estudo tem como foco o processo de transformação de modelos, desde um nível abstrato até chegar a um modelo mais concreto que possibilita a criação da IU implementada na linguagem escolhida, esse processo faz parte da Engenharia de Aplicação. Viu-se na seção 2.1, que para a construção de LPS é necessário

[1] Ruby on Rails

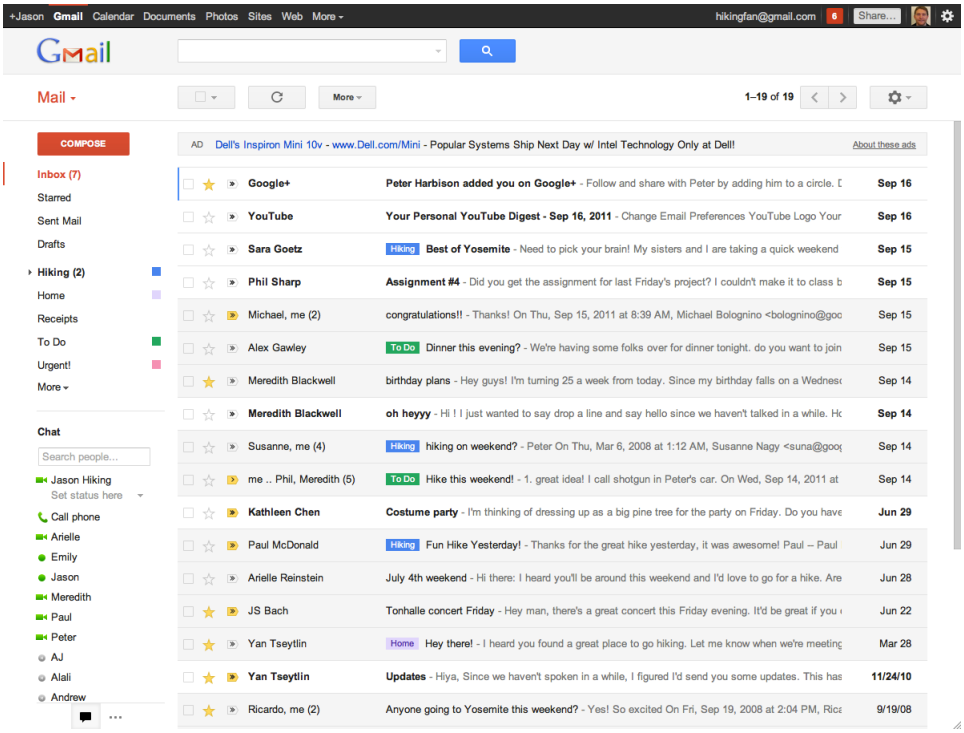

Figura 3.1: IU do cliente de email *Gmail*

o desenvolvimento de dois principais processos, a Engenharia de Domínio e a Engenharia de Aplicação. Na Figura 3.2 é apresentado um resumo do processo de derivação de produtos numa LPS.

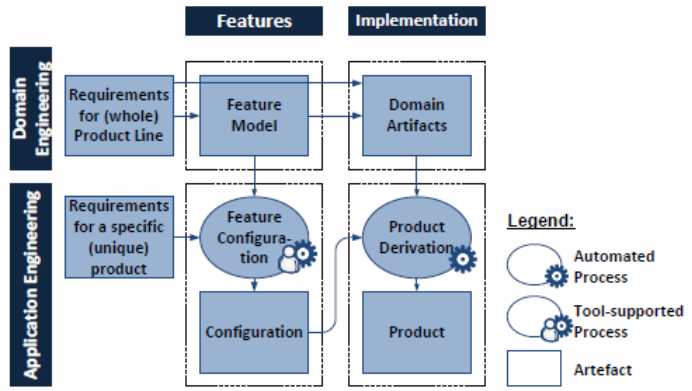

Figura 3.2: Interação entre Engenharia de Domínio e Engenharia de Aplicação na derivação de produtos, fonte: (Hauptmann et al. 2010)

A Figura 3.2 deve ser lida da esquerda para direita e de cima para baixo. Ela representa os dois processos na coluna da esquerda e seus elementos no topo. O próximo passo é apresentar o modelo de *features* da LPS *Webmail* pois é com ele que se inicia a Engenharia de Aplicação necessária na criação do produto de software de uma LPS.

3.1.1 Modelo de *features*

O *modelo de features* é o que representa o modelo de variabilidade de uma LPS. Com ele é possível definir o conjunto de *features* do produto que se deseja criar, além disso ele é o primeiro modelo utilizado na derivação de um produto, seja para o processo de criação da parte lógica quanto da parte visual do produto final. Na Figura 3.3 é apresentado o *modelo de features* da LPS *Webmail*.

Figura 3.3: *Modelo de features* da LPS *Webmail*, fonte: (17)

Na Figura 3.3 pode-se verificar todo o conjunto de *features* dos possíveis produtos da LPS. Para uma melhor compreensão sobre o diagrama apresentado e entendimento das regras e notações presentes neste modelo, recomenda-se a leitura de (Kang et al. 1990). De forma resumida, pode-se dizer que as esferas escuras representam *features* obrigatórias e as esferas claras *features* opcionais.

Apresenta-se a seguir a IU de alguns produtos dessa LPS para motivar os problemas tratados neste trabalho. Primeiro mostra-se um produto com todas as *features* do *Webmail*, seguido do produto com apenas as *features* obrigatórias e mais dois exemplos de produtos com variações no conjunto de *features* selecionadas.

3.2 Desafios na IU

Para mostrar os problemas que são tratados neste estudo, apresenta-se primeiro a IU do produto que contêm todas as *features* da LPS seguido do produto com menor número de *features*, ou seja, apenas as obrigatórias. A partir do primeiro produto apresentado

(completo) apresenta-se a IU de dois produtos diferentes com um número menor de *features*.

O algoritmo utilizado para posicionamento das unidades de apresentação (PU, do inglês Presentation Unit) na derivação da IU do produto é uma simples remoção dos módulos da tela, ou seja, cada *feature* que possui uma PU na IU possui sua posição fixa (a mesma posição para todos os produtos da LPS). É apresentado na Figura 3.4 a IU do produto com todas *features* do *Webmail*.

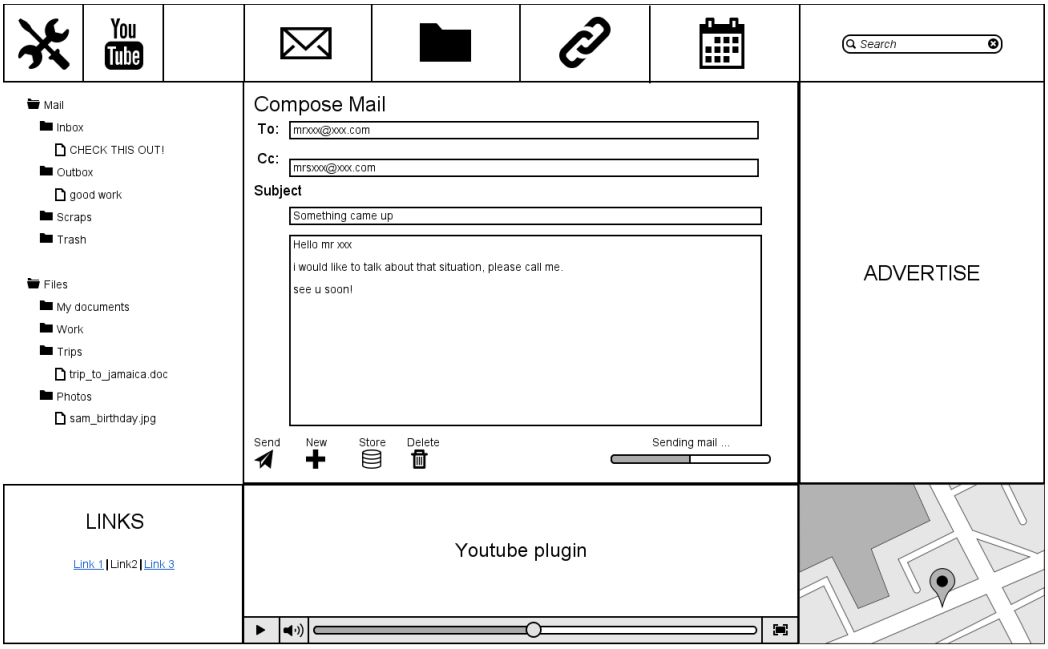

Figura 3.4: IU do produto da LPS *Webmail* com todas *features*

A IU apresentada na Figura 3.4 não apresenta nenhum problema pois esta é a IU planejada do produto, ou seja, a IU ideal. A seguir, na Figura 3.5 apresenta-se a IU do produto com apenas as *features* obrigatórias.

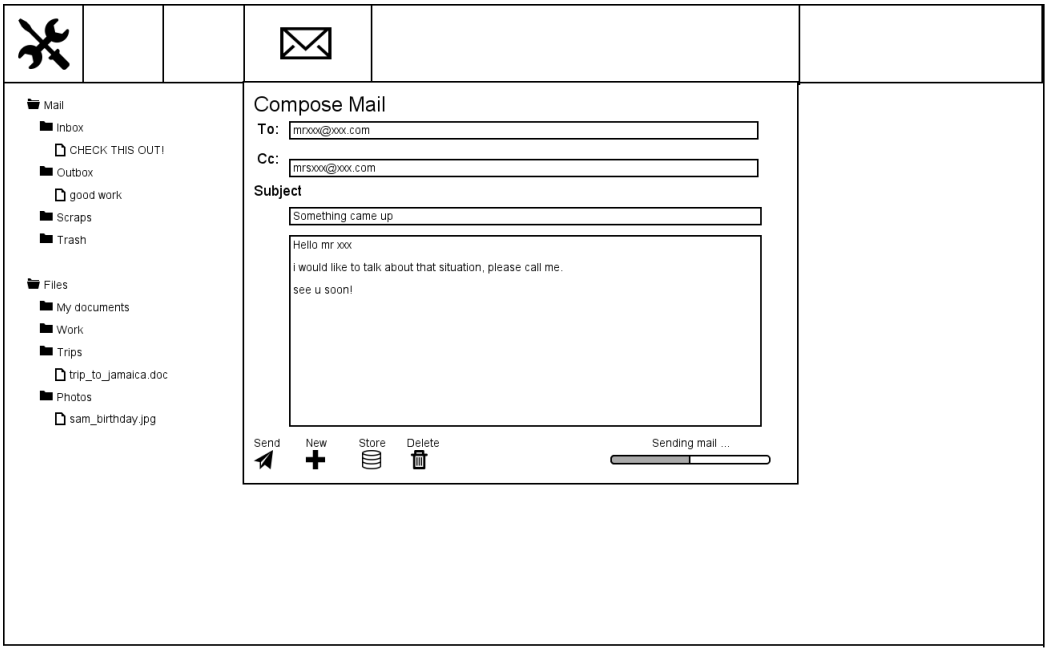

Figura 3.5: IU do produto da LPS *Webmail* com apenas as *features* obrigatórias

Apresenta-se a seguir, na Figura 3.6 e na Figura 3.7 alguns problemas que aparecem quando retira-se algumas *features*.

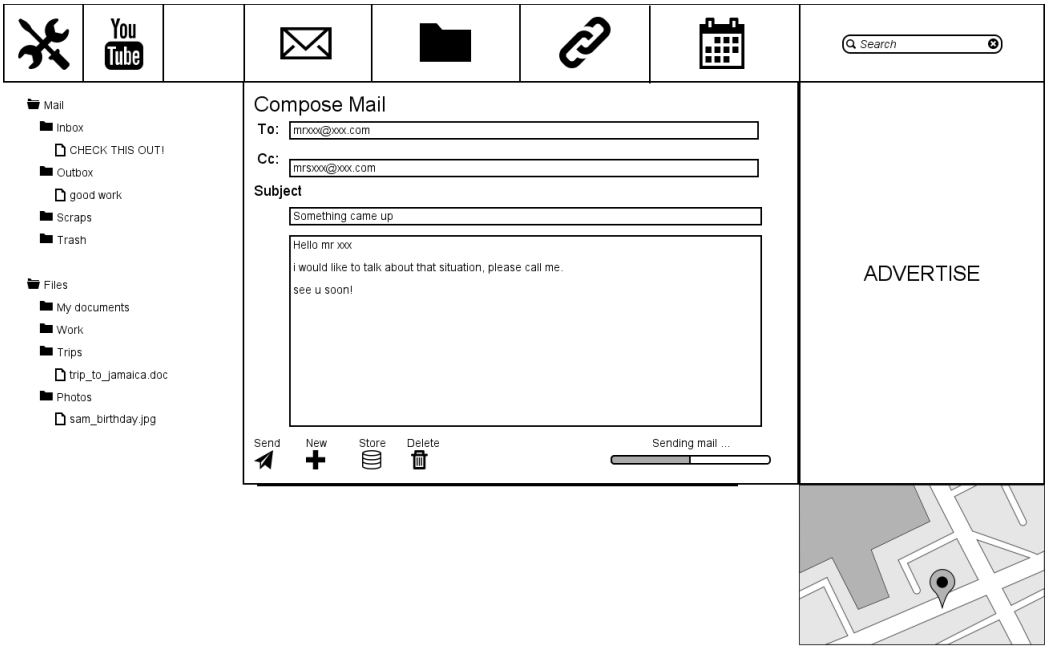

Figura 3.6: Exemplo de IU sem plugin youtube e calendário

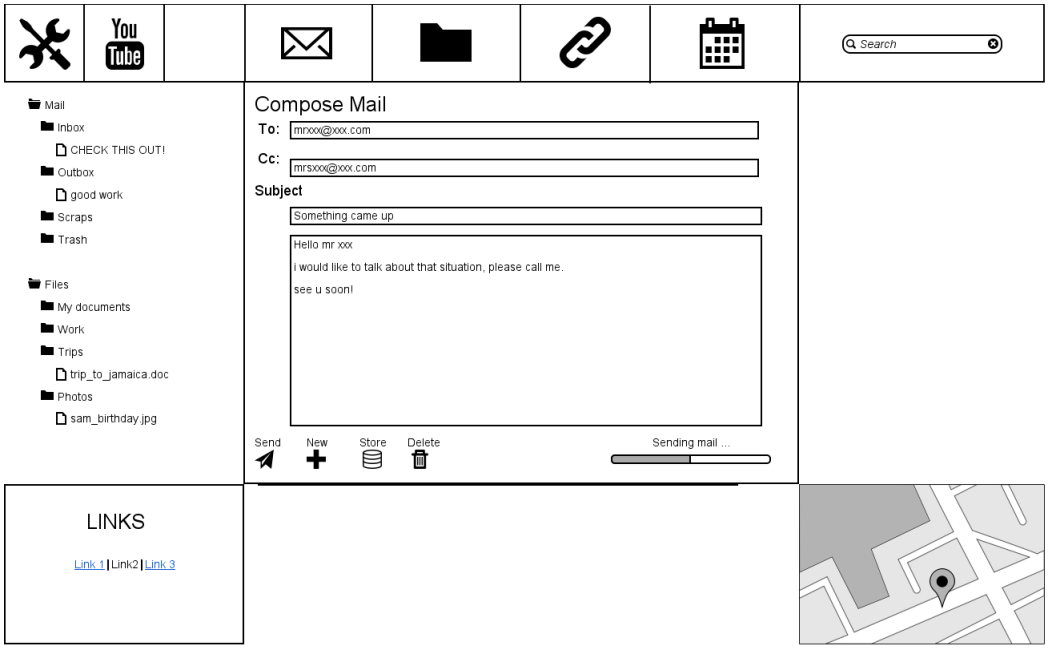

Figura 3.7: Exemplo de IU sem links e propaganda

Com a análise da Figura 3.6 e da Figura 3.7 é possível concluir que definir a posição das PU‘s fixa para todos os produtos da LPS não é um método eficiente de criar a IU do produto. Isto pois quando uma certa *feature* não é selecionada, no seu lugar não aparece coisa alguma. Deste modo diversos espaços em branco surgem na IU o que prejudica o seu uso e introduzem pelo menos dois problemas, que são:

Problema 1: Desperdício de espaço Ocorre um grande desperdício de espaço de tela quando existem espaços em branco. Uma solução para este problema é reorganizar os PU‘s da tela de forma que aquilo que é de maior importância (prioridade) possa ocupar uma área maior na IU. Desta maneira é possível estender alguns módulos para evitar os espaços em branco dos PU‘s não selecionados.

Problema 2: Baixa usabilidade Os espaços em branco criados pelas *features* faltantes tornam a IU pouco ortogonal e podem confudir o usuário. A IU perde um pouco de sua comunicabilidade e se torna não ergonômica, o que desencoraja o uso do produto principalmente se existem outros aplicativos que resolvem o mesmo problema mas apresentam uma IU mais amigável e com seus elementos bem dispostos.

3.3 Considerações Finais

Neste capítulo apresentou-se o escopo deste estudo com o detalhamento da LPS que será utilizada como estudo de caso da técnica proposta, analisando a IU considerada ideal e alguns problemas que surgem quando utiliza-se um algoritmo simples de posicionamento de PU‘s para derivar a IU dos produtos.

No próximo capítulo é apresentado em detalhes a técnica desenvolvida, com uma explicação de suas restrições de uso, seus modelos e seus algoritmos. Também apresenta-se um resumo breve da técnica MBUID e como o método proposto deriva e se adapta a ela.

4 SOLUÇÃO

Neste capítulo é apresentado a técnica desenvolvida que sugere uma possível solucão para os problemas apresentados na seção 3.2. Este capítulo inicia-se com uma breve descrição dos problemas de IU que estão relacionados a técnica, seguido de uma descrição detalhada do método proposto.

4.1 Breve descrição dos problemas

Antes de apresentar a técnica, é importante identificar quais são os principais problemas que ela se propõe a resolver e que as técnicas utilizadas atualmente não resolvem.

Problema 1: Espaços em branco Esse problema se refere aos espaços em branco que surgem quando a posição de cada PU é fixa para a LPS, deste modo quando uma *feature* que possui uma PU na tela não é selecionada para o produto, no seu lugar não aparece nada.

Problema 2: Posição dos PU's Esse problema é um requisito necessário para a customização de PU's de modo que seja possível definir a posição individual de cada PU dentro de um conjunto de posições possíveis, de acordo com o *layout* da aplicação.

Considerando os problemas apresentados, mostra-se em detalhes a solução desenvolvida para resolver estes problemas.

4.2 Técnica proposta - *PG-Constraint*

O nome *PG-Constraint* pode ser dividido em duas partes, PG e Constraint. PG é a sigla do inglês *Presentation Group* que traduzido para o português significa "Grupo de Apresentação". Neste trabalho um grupo de apresentação é um *cluster* que precisa ter uma representação na IU do produto, sendo ele representado por um PU. *Constraint* é uma palavra do inglês que significa restrição, são as restrições que modelam as customizações do método.

Para definir a técnica em poucas palavras, pode-se dizer que ela se resume na utilização de uma tabela para representar cada PG da IU do produto que se está derivando, onde para cada PG é possível definir um conjunto de restrições que definem alguns atributos desejados na IU final.

4.2.1 Origem

A técnica *PG-Constraint* possui suas raízes no método MBUID. Pode-se ver na Figura 2.3 que o processo MBUID é composto por várias etapas, cada etapa com um propósito e modelos distintos. A derivação de um produto inicia com um modelo abstrato de IU (AUI, do inglês Abstract User Interface), seguida de um processo de clusterização que possui o objetivo de dividir o *modelo de features* da LPS em PU's que transforma o AUI num *modelo de cluster*. Deste *modelo de cluster* é executado um processo automatizado para criar um modelo de navegação que passa por outro processo automatizado afim de criar dois modelos intermediários, um modelo de navegação e um modelo de apresentação.

O modelo de navegação serve para gerenciar a navegação entre as telas do produto final, por isto não é contemplado neste estudo. O modelo de apresentação contém as PU's porém num nível ainda abstrato, desta forma é necessário a transformação deste modelo para um modelo concreto de apresentação (CUI, do inglês Concrete User Interface). Depois de criado este modelo concreto é executado a última transformação para a linguagem de IU desejada.

4.2.2 Objetivo

A técnica *PG-Constraint* foi proposta com o objetivo de fornecer um nível de customização de IU não contemplado na técnica MBUID que foi verificada e implementada em (Hauptmann et al. 2010). A técnica foi criada com o intuito de que uma futura integração com o método MBUID seja possível, de modo que um novo nível de customização esteja disponível, mantendo-se os outros benefícios.

A técnica visa resolver os problemas de posicionamento de PG's e aproveitamento de espaço de tela pois verificou-se que a técnica MBUID é muito eficiente para trabalhar com a estrutura e disposição de itens dentro de um dado PG, entretanto ela não possui uma maneira intuitiva e direta de escolher o posicionamento e atributos relacionados ao posicionamento entre os PG's da IU. A seguir, apresenta-se as principais restrições da técnica *PG-Constraint*.

4.2.3 Restrições

Devido a natureza do trabalho (estudo exploratório), a técnica desenvolvida possui algumas restrições pois ela trata de um estudo de caso específico, logo existem algumas restrições que tornam a técnica um pouco menos genérica do que o desejado. A seguir apresenta-se algumas das restrições consideradas.

Limite de PG's Para simplificar a técnica/problema, definiu-se que não é possível selecionar mais PG's para a IU do que o número de posições possíveis para os módulos. Essa restrição pode ser eliminada se for aceito que um ou mais PG's selecionados podem não aparecer na IU final.

Layout das posições Por restrição de *layout* temos o *Grid Layout* onde cada posição possui bem definido seus vizinhos na vertical e horizontal. A utilização de *layouts* diferentes é uma das sugestões para trabalhos futuros.

Extensibilidade Este item possui duas restrições relacionadas, a primeira se refere a restrição de cada PG só poder ser extendido para uma direção, vertical ou horizontal. A segunda diz respeito as restrições inerentes ao *layout* utilizado, por exemplo, um

PG na posição no canto esquerdo da tela que deseja se extender por toda a tela horizontalmente não poderá estar presente apenas nos extremos mas de forma contínua, ou seja, presente também nas posições intermediárias.

A seguir é apresentado os modelos desenvolvidos que possibilitam a implementação e o entendimento completo da técnica *PG-Constraint*.

4.2.4 Modelagem da solução

Para explicar a origem da modelagem desenvolvida é necessário primeiro explicar os passos iniciais da técnica MBUID e indicar onde se inserem os modelos desenvolvidos da técnica *PG-Constraint*. Os principais passos do MBUID estão representados na Figura 2.3, a seguir explica-se brevemente alguns deles.

Passo 1: Conhecer o *feature model* Para derivar qualquer produto de qualquer LPS é necessário primeiramente um modelo com as *features* que a LPS disponibiliza. Pode-se ver o modelo de *features* na Figura 3.3.

Passo 2: Criação do modelo AUI Nesta etapa é desenvolvido uma espécie de árvore que representa os possíveis itens de IU que as *features* do *modelo de features* representam. Pode-se ver um exemplo de modelo AUI na Figura 4.1.

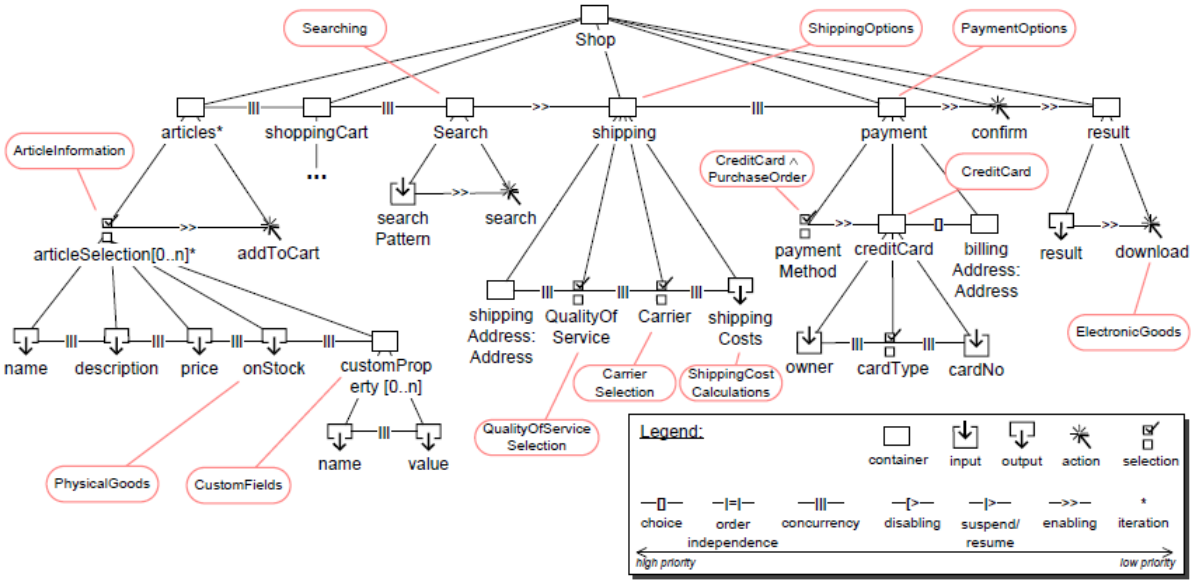

Figura 4.1: Exemplo de modelo AUI do método MBUID, fonte: (Pleuss et al. 2012)

Passo 3: Criação do modelo de clusters/PG's O modelo de clusters é criado a partir do modelo AUI, nele é possível representar os PG's que farão parte do produto final, de modo que aqui é possível customizar a estrutura interna de cada PG. Um exemplo de *modelo de cluster* construído sobre o modelo AUI é apresentado na Figura 4.2.

Com o *modelo de clusters* já definido, o próximo passo do MBUID é desenvolver o modelo navegacional abstrato para depois uní-lo com o *modelo de clusters* para produzir o modelo navegacional e o modelo de apresentação. É na etapa de criação do modelo navegacional abstrato que a técnica *PG-Constraint* se distancia da MBUID, pois ela trata

apenas da disposição de PU's na IU, diferente do MBUID que trata da navegação e da composição interna de cada PU.

Com o *modelo de clusters* é possível mapear cada PU selecionado para uma tabela de modo que seja possível definir algumas restrições para o posicionamento dos PU's. Antes de apresentar essa tabela é necessário mostrar um desenho que apresente as posições de tela que cada PU pode ocupar, pois isto será um fator importante no algoritmo de customização. A seguir, a tabela de posições utilizada neste estudo de caso.

4.2.4.1 Tabela de Posições

A tabela de posições deste estudo não está representada em forma de tabela mas ela pode ser representada por uma. A principal função dessa tabela é servir como um guia responsável por identificar todas as possíveis posições que cada PG pode ocupar. Utilizando essa tabela, o *modelo de clusters* e as restrições desejadas é possível a criação automatizada da tabela *PG-Constraint*. Pode-se ver na na Tabela 4.1 as possíveis posições para PG's deste estudo de caso, a LPS *Webmail*.

Tabela 4.1: Tabela de posições para a LPS *Webmail*

Topo Esquerda TE	Topo Centro TC	Topo Direita TD
Centro Esquerda CE	Centro Centro CC	Centro Direta CD
Baixo Esquerda BE	Baixo Centro BC	Baixo Direita BD

Além de definir as possíveis posições dos PG's, essa tabela possui mais duas funções . A primeira é identificar quais são os vizinhos na vertical e na horizontal de cada posição, a segunda é armazenar as restrições de extensibilidade. Essa funções extras são discutidas com mais detalhe na seção de trabalhos futuros do próximo capítulo. A seguir, apresenta-se a tabela *PG-Constraint* que dá nome a técnica desenvolvida, todas as customizações possíveis estão representadas nela.

4.2.4.2 Tabela PG-Constraint

A tabela *PG-Constraint* é o principal modelo da técnica pois é nela que estão as restrições que definirão as posições finais dos PG's. Ela armazena todos os grupos de apresentação (PG's) selecionados no *modelo de cluster* e que vão compor o produto final. Para entender melhor o seu uso primeiro é preciso definir o que é um PG. Tanto para a *PG-Constraint* quanto para o método MBUID, um PG é um cluster, sendo esse cluster podendo ser composto internamente por um ou mais clusters e/ou fragmentos, este conceito está definido em (Botterweck 2006). Para uma visualização do conceito de cluster e fragmento, apresenta-se a seguir um exemplo de *modelo de clusters* e fragmentos na Figura 4.2.

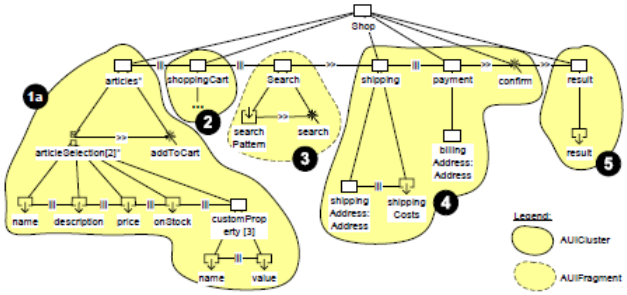

Figura 4.2: Modelo de clusters de uma loja web, fonte: (Pleuss et al. 2012)

A principal função da tabela *PG-Constraint* é definir as restrições de cada PG. Apresenta-se a Tabela 4.2 que representa a IU do produto representado na Figura 3.4, ou seja, o produto com todas as *features* da LPS *Webmail*.

Tabela 4.2: Tabela *PG-Constraint* referente a figura 3.4

Tabela de definição dos PG-Constraints (WEBMAIL COMPLETO)

	Menu - Opcoes	Menu - Funcoes	Search	Main PG	Youtube	Adds	Directory Tree	Maps	Links
Fixo	Sim	Sim	Não	Sim	Não	Não	Não	Não	Não
Extensivel	Não	Sim / Horizontal	Não	Sim, Horizontal	Sim, Horizontal	Sim, Vertical	Sim, Vertical	Sim, Horizontal	Sim, Horizontal
Prioridade	8	9	1	10	5	4	9	3	2
Posição 1	TE	TC	TD	CC	BC	CD	CE	BD	BE
Posição 2	-	-	BD	-	BD	BD	CD	BE	BD
Posição 3	-	-	BE	-	-	BE	-	BC	TD

Explica-se agora os atributos dessa tabela.

Atributo 1: Fixo Esse atributo é utilizado para definir se o PG pode ou não mudar de posição na IU final. Caso ele esteja ativo, o atributo posição2 e posição3 devem ser nulos.

Atributo 2: Extensível O atributo extensível se refere a possibilidade de extensão do PG. Para este atributo são possíveis três estados: desativado, ativo-horizontal e ativo-vertical. A extensão de PG's é utilizada para evitar o desperdício de espaço na tela e dar prioridade a aquelas *features* mais prioritárias.

Atributo 3: Prioridade Apesar de não ser o primeiro atributo da tabela, prioridade certamente é o atributo que possui maior impacto no algoritmo da técnica *PG-Constraint*. A prioridade ajuda a definir a ordem de colocação dos PG's na tela além de definir quem irá ocupar uma dada posição de tela quando uma ou mais PG's possuem prioridades distintas.

Atributo 4: Posição(1,2,3) O atributo posição(1,2,3) representa as posições que se aceita que o PG possa ocupar na IU final. Eles possuem ordem de prioridade da seguinte forma: posicao1 > posicao2 > posicao3. Para todo PG não fixo é necessário a escolha de pelo menos posicao1 e posicao2 pois uma realocação do PG não fixo deve ser possível.

Pode-se perceber que é possível estender essa tabela de modo a adicionar mais atributos para aumentar o nível de customização da técnica, o que acarretaria mudanças no algoritmo da técnica. A seguir o último modelo da técnica *PG-Constraint*.

4.2.4.3 Modelo Concreto - XML

O modelo concreto é o modelo mais próximo da linguagem de implementação da IU, é a partir dele que será criado o código para a linguagem final de IU, neste caso de estudo a linguagem HTML. Para representar este modelo escolheu-se utilizar um arquivo no formato XML pois ele é um formato muito divulgado, de fácil entendimento e com muitas ferramentas existentes para fazer sua leitura. Na Figura 4.3 é apresentado o modelo do DTD (Data Type Definition) do arquivo XML dado a tabela de *PG-Constraint* da Figura 4.2.

```
<!ELEMENT cui (position)>
<!ELEMENT position (cluster)>
<!ELEMENT cluster (priority,extensible,position1,position2,position3)>
<!ELEMENT extensible (horizontal,vertical)>

<!ELEMENT horizontal (#PCDATA)>
<!ELEMENT vertical (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT position1 (#PCDATA)>
<!ELEMENT position2 (#PCDATA)>
<!ELEMENT position3 (#PCDATA)>
```

Figura 4.3: Formato do documento XML

A seguir é apresentado o algoritmo desenvolvido para a técnica *PG-Constraint*.

4.2.5 Algoritmo

Nesta seção mostra-se em detalhes o algoritmo responsável por calcular a posição e extensão dos PG‘s escolhidos e representados na tabela *PG-Constraint*. Se fosse necessário definir o algoritmo em poucas palavras, pode-se dizer que ele consiste em transformar a tabela de posições e a tabela *PG-Constraint* em um modelo final que pode ser representado por um arquivo XML. Com este arquivo é possível o desenvolvimento da IU final na linguagem desejada bastando apenas a utilização de algum leitor de XML que se encontra com facilidade na literatura.

Na Figura 4.4 apresenta-se a função principal do algoritmo e na Figura 4.5 as duas funções auxiliares, a Extende e Realoca.

Considerando as seguintes variáveis:
viz: uma posição da tabela de posições
ent1, ent2 : duas entradas distintas da tabela PG-Constraint
posicoesPG: lista de PG por posição
pos: uma posição da tabela de posições
posLivres: número de posições livres
pgFixoOrdenado: lista de PG fixo ordenado por prioridade decrescente
pgNaoFixoOrdenado: lista de PG não fixo ordenado por prioridade decrescente

1. **Validar tabela PG-Constraint**
   a. Para cada combinação possível (**ent1**,**ent2**)
      i. Verifica se **ent1**.posicao1 == **ent2**.posicao1
         1. se sim , tabela inválida
      ii. Verifica se **ent1**.fixo != **ent2**.fixo E **ent1**.prioridade != **ent2**.prioridade
         1. se sim, se **ent1**.extensivel == **ent2**.extensivel
            a. se sim, se **ent1** é vizinho de **ent2**
               i. se sim, tabela inválida
   b. Para cada **ent1** possível
      i. Verifica se **ent1** não fixo possui pelo menos 2 posicoes
         1. se sim , tabela inválida
2. **Armazenar na variável** posicoesPG **cada entrada da tabela**
   a. Para cada posição **pos**
      i. Para cada entrada **ent1**
         1. se **ent1**.posicao1 == **pos**
            a. posicoesPG[**pos**] = **ent1**
3. **Armazenar na variável** posLivres **o número de posições livres**
   a. Para cada posição **pos**
      i. se **posicoesPG**[**pos**] == nulo
         1. incrementa em 1 **posLivres**
4. **Posicionar e tentar estender todos os PG's fixos**
   a. Para cada elemento **pg** da variável **pgFixoOrdenado**
      i. se **posLivres** > 0
         1. se sim, se **pg** for estensível
            a. **ESTENDE(pg)**
      ii. define a posição do **pg**
5. **Posicionar e tentar estender todos os PG's não fixos**
   a. Para cada elemento **pg** da variável **pgNaoFixoOrdenado**
      i. se **posLivres** > 0
         1. se sim, se **pg** for estensível
            a. **ESTENDE(pg)**
      ii. define a posição do **pg**
6. **FIM**

Figura 4.4: Algoritmo da técnica *PG-Constraint*

4.2.5.1 Validação da tabela PG-Constraint

Para o correto funcionamento da técnica desenvolvida é necessário a definição de algumas regras quanto ao formato do conteúdo presente na tabela *PG-Constraint*, os passos:

Passo 1.A.1: Dois PG's não podem compartilhar posição1 Essa restrição define que dois PG's quaisquers não podem ter definidos como posição1 uma mesma posição. Essa validação não é obrigatória, mas evita um erro comum de sobreposição de PG's e elimina mais um fator de aleatoriedade no resultado final da IU.

Passo 1.A.2: Dois PG's não podem ser iguais (fixo,prioridade) Essa restrição existe para evitar casos onde por exemplo, tem-se dois PG's, fixos e de mesma prioridade que podem ser estendidos. Criando esta restrição não existe dúvida sobre qual PG sofrerá a estensão.

Passo 1.B.1: Cada PG não fixo deve possuir pelo menos duas posições Essa validação não é obrigatória para o funcionamento do algoritmo, entretanto é altamente recomendada pois caso um PG não fixo tenha que ser movida, a posição final do PG poderá estar dentro do conjunto de posições especificado na tabela.

ALGORITMO - ESTENDE(pg)

1. Para cada vizinho **viz** de **pg**
   a. se **viz** == posição livre
      i. **posicoesPG[viz] = pg**
   b. se **viz** não é fixo
      i. **REALOCA(viz)**
      ii. **posicoesPG[viz] = pg**
   c. senão
      i. nao pode estender nessa posição

ALGORITMO - REALOCA(pg)

1. se **posicoesPG[**pg.posicao2**] == nulo ou posicoesPG[**pg.posicao3**] == nulo**
   a. se sim, posiciona **pg** na posicao livre
   b. senão, se **posicoesPG[**pg.posicao2**] e posicoesPG[**pg.posicao3**]** estão ocupadas
      i. se prioridade **pg** > **posicoesPG[**posicao2,3**]**
         1. **REALOCA(posicoesPG[**posicao2,3**])**
         2. **posicoesPG[**posicao2,3**]** = **pg**
      ii. se prioridade **pg** < **posicoesPG[**posicao2,3**]**
         1. posiciona **pg** em alguma posição livre

Figura 4.5: Algoritmos auxiliares da técnica *PG-Constraint*

Apresenta-se agora como é feito o cálculo da posição e estensão dos módulos.

4.2.5.2 Cálculo da posicão e extensão dos PG‘s

Depois que a tabela *PG-Constraint* está validada, o próximo passo é posicionar os PG‘s nas posições representadas pela tabela de posições da Figura 4.1. A seguir apresenta-se os últimos passos do algoritmo:

Passo 2: Guardar em cada posição o respectivo PG ocupado Neste passo é armazenado em uma variável qual PG está contido em cada uma das posições de tela possíveis. O procedimento é ler todas as entradas da tabela *PG-Constraint*, e para cada atributo posição1, relacionar o PG com a posição escolhida.

Passo 3: Contar o número de posições livres Neste passo é feita a contagem de quantas posições não estão ocupadas para definir quantas estensões de PG’s serão possíveis. O procedimento é muito simples, basta ler a variável resultante do passo 2 e para cada posição sem um PG relacionado soma-se um ao número total de posições livres.

Passos 4 e 5: Posicionar PG‘s por ordem de prioridade Os passos 4 e 5 são semelhantes, o principal fator que os diferencia é o tipo de PG que está sendo posicionado, no passo 5 toma-se os PG’s fixos e no passo 6 os não fixos. Primeiramente é selecionado todos os PG’s fixos e ordena-se eles de acordo com suas prioridades. Inicia-se pelo PG de maior prioridade e fixo, se verifica se é possível sua extensão, se a resposta for positiva deve-se tentar estender o PG. Caso a posição necessária para a estensão estiver ocupada, procura-se movimentar quem estiver ocupando se sua prioridade for menor que a PG que está sendo estendida.

A saída deste algoritmo é uma lista das posições com cada posição contendo o seu respectivo PG que o ocupa. Com esta lista é possível criar o arquivo XML que poderá ser utilizado para criar a IU final. A seguir, os resultados da técnica aplicado ao *Webmail*.

4.3 Resultados

Apresenta-se, nesta seção, os resultados da técnica *PG-Constraint* para os quatro exemplos apresentados no capítulo 3, o estudo de caso.

4.3.1 Exemplo 1

O primeiro exemplo é composto pelo produto com todas as *features* da LPS *Webmail*, como aparece na Figura 3.4.

Tabela 4.3: Tabela *PG-Constraint* do Exemplo 1

PG-CONSTRAINT Table

Presentation Group	FIXED	EXTENSIBLE	PRIORITY	position-1	position-2	position-3
Main PG	yes	horizontal	10	CC		
Menu - Options	yes	no	8	TE		
Menu - Functions	yes	horizontal	9	TC		
Directory Tree	no	vertical	9	CE	CD	
Adds	no	vertical	4	CD	BD	BE
Youtube	no	horizontal	5	BC	BD	
Links	no	horizontal	2	BE	BD	TD
Maps	no	horizontal	3	BD	BE	BC
Search	no	no	1	TD	BD	BE

generate HTML

Resultado da tela com o algoritmo *PG-Constraint*:

Example 1

Menu - Options	Menu - Functions	Search
Directory Tree	Main PG	Adds
Links	Youtube	Maps

Figura 4.6: Resultado do Exemplo 1 da técnica *PG-Constraint*

4.3.2 Exemplo 2

O segundo exemplo é composto pelo produto com apenas as *features* obrigatórias da LPS *Webmail*, como aparece na Figura 3.5.

Tabela 4.4: Tabela *PG-Constraint* do Exemplo 2

PG-CONSTRAINT Table

Presentation Group	FIXED	EXTENSIBLE	PRIORITY	position-1	position-2	position-3
Main PG	yes	horizontal	10	CC		
Menu - Options	yes	no	8	TE		
Menu - Functions	yes	horizontal	9	TC		
Directory Tree	no	vertical	9	CE	CD	

generate HTML

Resultado da tela com o algoritmo *PG-Constraint*:

Example 2

Menu - Options	Menu - Functions	Menu - Functions
Main PG	Main PG	Main PG
Directory Tree		

Figura 4.7: Resultado do Exemplo 2 da técnica *PG-Constraint*

4.3.3 Exemplo 3

O terceiro exemplo é composto pelo produto com as *features* da Figura 3.6

Tabela 4.5: Tabela *PG-Constraint* do Exemplo 3

PG-CONSTRAINT Table

Presentation Group	FIXED	EXTENSIBLE	PRIORITY	position-1	position-2	position-3
Search	no	no	1	TD	BD	BE
Maps	no	horizontal	3	BD	BE	BC
Adds	no	vertical	4	CD	BD	BE
Directory Tree	no	vertical	9	CE	BD	
Menu - Options	yes	no	8	TE		
Menu - Functions	yes	horizontal	9	TC		
Main PG	yes	horizontal	10	CC		

generate HTML

Resultado da tela com o algoritmo *PG-Constraint*:

Figura 4.8: Resultado do Exemplo 3 da técnica *PG-Constraint*

Example 3

Menu - Options	Menu - Functions	Search
Main PG	Main PG	Main PG
Maps	Adds	Directory Tree

4.3.4 Exemplo 4

O quarto e último exemplo é composto pelo produto com as *features* da Figura 3.7

Tabela 4.6: Tabela *PG-Constraint* do Exemplo 4

Presentation Group	FIXED	EXTENSIBLE	PRIORITY	position-1	position-2	position-3
Main PG	yes	horizontal	10	CC		
Menu - Functions	yes	horizontal	9	TC		
Menu - Options	yes	no	8	TE		
Directory Tree	no	vertical	9	CE	CD	
Maps	no	horizontal	3	BD	BE	BC
Search	no	no	1	TD	BD	BE
Links	no	horizontal	2	BE	BD	TD

generate HTML

Resultado da tela com o algoritmo *PG-Constraint*:

Example 4

Menu - Options	Menu - Functions	Search
Main PG	Main PG	Main PG
Links	Directory Tree	Maps

Figura 4.9: Resultado do Exemplo 4 da técnica *PG-Constraint*

4.4 Considerações Finais

Neste capítulo apresentou-se a técnica *PG-Constraint* e suas principais características. Mostrou-se em detalhes o algoritmo de posicionamento e extensão dos PG's além dos modelos e tabelas desenvolvidos. Verificou-se também que enquanto a técnica MBUID foca na estrutura interna de cada unidade de apresentação, a técnica *PG-Constraint* se preocupa com a disposição de cada unidade de apresentação na IU como um todo.

No próximo capítulo é feito uma discussão dos resultados da técnica apresentada, identificando pontos positivos e negativos. Seguindo essa discussão são feitas indicações de trabalhos futuros como por exemplo uma possível união desta técnica com a MBUID de modo que seja possível aumentar o poder de customização da técnica MBUID utilizando os conceitos aqui apresentados.

5 DISCUSSÃO

Nesta seção é discutido as lições aprendidas com relação ao estudo de caso desenvolvido e do uso das extensões da técnica MBUID. O foco desta discussão é os pontos positivos e negativos da técnica, com o intuito de apontar quais os resultados foram satisfatórios e quais necessitam de mais pesquisa.

5.1 Abordagens Existentes

Durante a pesquisa para identificar as abordagens existentes para derivação de produtos numa LPS, identificou-se pelo menos dois tipos de soluções possíveis. A primeira solução consiste em criar manualmente a IU de cada produto de modo que todos os requisitos de interface da aplicação sejam satisfeitos, para isto o produto depois de criado deve passar por uma análise e por modificações feitas por um projetista.

A segunda solução é utilizar um método semi-automático que podem ser divididos em duas famílias.

1a Família

A primeira família é composta das técnicas utilizadas para a derivação da parte lógica da aplicação. Alguns exemplos das técnicas: *Compilação Condicional*, *Programação Orientada a Feature*, *Padrões de Design* entre outras.

2a Família

A segunda família é composta dos métodos baseados em modelos para desenvolvimento de IU‘s, com seu principal representante a técnica MBUID.

Apresenta-se a seguir uma discussão dos principais problemas quanto a utilização dos métodos apresentados na seção 5.1.

5.1.1 Criação manual

A criação manual da IU de cada produto possui como principal motivação a possibilidade de customizar praticamente qualquer aspecto de interface, ou seja, todo e qualquer requisito visual pode ser satisfeito quando temos uma pessoa para modificar a IU. Entretanto, existem alguns problemas quando utiliza-se este método para LPS. A seguir, os principais problemas relacionados à utilização em conjunto de LPS.

Ineficiência De acordo com (Myers 1995), mais da metade do código de sistemas de software atuais consistem em código para GUI (do inglês, Graphic User Interface).

Portanto, modificar manualmente a IU de cada produto é um método muito ineficiente pois com o aumento dos produtos de uma LPS, o trabalho aumenta pois para cada modificação na LPS que reflita numa mudança de sua IU é necessário uma mudança manual em cada produto.

Informação desconsiderada Durante a Engenharia de Domínio e a Engenharia de Aplicação muitas informações relevantes são produzidas e colocadas em *modelos de features* e *configurações de features*. Com a criação manual da IU não existe uma maneira sistematizada de considerar essas informações, portanto algumas configurações do domínio da LPS podem ser desconsideradas pelo projetista que executa a modificação da IU.

Propensão a erros Conectar uma IU criada manualmente com a lógica do produto gerada automaticamente produz muitos erros. Além disso, devido a conexão manual as técnicas para testar os eventos da IU não podem ser utilizadas, com isso é necessário maior intervenção humana e aumentam as chances de erros não serem verificados.

Complexidade de manutenção e evolução Uma das principais vantagens da utilização de LPS é a fácil rastreabilidade entre *features* e a implementação (código). Quando modificações na IU são feitas manualmente perde-se este funcionalidade, o que aumenta os custos com manutenção e evolução dos sistemas além de possibilitar a inserção de novos erros no produto.

5.1.2 Criação automatizada - Algoritmos comuns

Os algoritmos automatizados são os principais algoritmos utilizados para fazer a derivação da parte lógica dos produtos. A seguir, alguns exemplos destes algoritmos utilizados com frequência em LPS's.

- Orientação a Objeto e Polimorfismo
- Programação Orientada a *Feature*
- Compilação Condicional
- Programação Orientada a Aspectos

Os algoritmos citados são boas técnicas para derivar a lógica da aplicação, porém não são suficientes para a derivação da IU pois não permitem a customização da tela, desta forma o que elas fazem é basicamente permitir que um dado elemento de tela esteja presente ou não no produto final. Elas não tratam da estrutura interna de cada PG, nem do posicionamento relativo entre eles, por isso não são suficientes.

5.1.3 Criação automatizada - MBUID

Considera-se o método MBUID o maior representante de técnica automática de derivação de IU devido ao fato dele ser citado em muitos trabalhos e além disso há pouca pesquisa nessa área de estudo. O processo do MBUID é relativamente extenso e possui várias etapas onde é possível a customização dos PG's, tratando desde a organização interna de cada cluster quanto da navegação entre telas do sistema, entretanto os problemas citados na seção 3.2 não são tratados no algoritmo MBUID.

5.2 Possíveis Soluções

A técnica *PG-Constraint* procura solucionar os problemas da criação manual de IU ao utilizar elementos da LPS como o *modelo de features* e por possuir um algoritmo que permita automatizar e relacionar elementos da IU com elementos da LPS. Apresenta-se a seguir uma explicação breve da solução para cada problema citado na seção **??**.

Criação manual e algoritmos comuns Os problemas das técnicas manuais e das mais comuns de derivação como, por exemplo, *Compilação Condicional* são eliminados pois a técnica é utilizada para cada produto, durante o processo de Engenharia de Aplicação. Desta forma, cada produto é customizável com o método proposto.

Criação automatizada baseada em modelos O problema do método MBUID é a ausência de uma funcionalidade que possibilite o posicionamento relativo dos PG's na IU do produto. O método *PG-Constraint* objetiva esta ausência.

É importante citar que a técnica *PG-Constraint* não se limita a resolver o problema de aproveitamento do espaço e reorganização dos PG's, pois a tabela *PG-Constraint* pode conter mais atributos que podem ser utilizados para outras funções. Caso sejam adicionados atributos é necessário uma modificação no algoritmo. A seguir, uma análise dos pontos positivos e negativos da técnica *PG-Constraint*.

5.2.1 Pontos Positivos

Como pontos positivos da técnica apresentada, pode-se citar pelo menos dois, que são:

1o ponto: Customização A técnica apresenta um bom nível de customização pois ela permite que novos atributos sejam adicionados para modelar o que for necessário de acordo com a aplicação desenvolvida. Além disso, o modo de customização é bem intuitivo e de fácil uso, não sendo necessário um projetista de software para criar a IU depois de implementada a técnica *PG-Constraint*.

2o ponto: Algoritmo adaptável O algoritmo de cálculo das posições e extensão dos PG's é relativamente simples, pequeno e de fácil implementação. Outro fator importante é a possibilidade de adaptação do mesmo a novos atributos que forem necessários, bastando em alguns casos apenas a adição de algumas rotinas de código para adicionar uma nova funcionalidade.

5.2.2 Pontos Negativos

Como pontos negativos da técnica apresentada, pode-se citar pelo menos dois, que são:

1o ponto: Técnica específica A técnica é utilizável apenas para interfaces com o *layout* do tipo grade. Esta restrição diz que cada posição deve possuir bem definidos os vizinhos na horizontal e na vertical, além de definidas algumas restrições para o caso de estender as PG's.

2o ponto: Muitas possibilidades, muitos parâmetros O número de parâmetros necessários e algumas validações na tabela tornam a utilização deste método um pouco confusa e dispendiosa em quesitos de tempo. Além disso existe a possibilidade que exista outra solução para o problema proposto neste estudo não tão específica e que se adapte a sistemas de múltiplas plataformas distintas.

A seguir, sugestões de pesquisa e trabalhos futuros deste trabalho.

5.3 Sugestões de pesquisa

Apresenta-se agora uma lista de tópicos com assuntos relacionados a técnica apresentada.

1o: Integrar a técnica *PG-Constraint* ao MBUID A integração da técnica desenvolvida com o método MBUID possibilitaria uma ampliação no poder de customização do MBUID. Novos estudos que pesquisem essa possibilidade são fortemente sugeridos.

2o: Criar um modelo de representação da tabela de posições Este foi um tópico que não foi discutido neste trabalho, como modelar a tabela de posições e suas restrições. Este tópico é fortemente sugerido para ampliar o alcance da técnica apresentada.

3o: Criar uma maneira de adaptar o método a outros tipos de *layout* Um dos pontos negativos da técnica desenvolvida é o fato dela ser muito específica por possuir muitas restrições de uso como por exemplo o tipo de *layout* de IU. É sugerido como pesquisa a modificação do algoritmo de modo que ele passe a aceitar outros tipos de *layout* de IU.

5.4 Considerações Finais

Neste capítulo apresentou-se os principais métodos de derivação de IU para LPS e os problemas relacionados a cada método. Em seguida moustrou-se de que forma a técnica *PG-Constraint* propôe solucionar esses problemas e por fim uma análise dos pontos positivos e negativos da técnica proposta com algumas sugestões de pesquisas relacionadas.

No próximo capítulo conclui-se este estudo exploratório com uma lembrança do que foi desenvolvido, seguido das contribuições deste estudo para a área e finalmente a sugestão de trabalhos futuros que possam contribuir mais com a solução aqui proposta e a área de derivação de IU para LPS.

6 CONCLUSÃO

Neste trabalho apresentou-se o resultado de um estudo exploratório sobre algoritmos de derivação de produtos de software de LPS. Esse estudo possui como escopo a área de derivação de produtos de LPS, com foco na criação automatizada da interface visual e na customização da mesma. O problema investigado consiste na falta de controle do posicionamento dos PG's, ou seja, de algum método semi-automatizado que possibilite a configuração da posição dos elementos de tela no processo da Engenharia de Aplicação presente nas LPS's. Investigou-se também a possibilidade de extensão de PG's para aproveitar espaços em branco na tela provenientes de *features* não presentes no produto.

Para solucionar esses problemas foi proposto um método que utilizou como estudo de caso um cliente de emails para a plataforma *web*, apresentado no capítulo 3. A técnica desenvolvida denominou-se *PG-Constraint*, ela possui forte relação com a técnica MBUID por utilizar elementos em comum que possibilitaram sua concepção. Ela é composta por dois modelos e um algoritmo.

6.1 Contribuições

Como principal contribuição deste estudo pode-se citar a técnica *PG-Constraint*. Ela propõe uma solução que consiste na possibilidade de customização do posicionamento de unidades de apresentação de tela, de modo que seja possível criar diferentes IU's para produtos com conjuntos de *features* idênticos. Outra contribuição da técnica é um método para aproveitar os espaços em branco, possibilitando a extensão de unidades de apresentação e o reaproveitamento do espaço de *features* não selecionadas.

Além disso, esse estudo permitiu verificar que as abordagens existentes resolvem alguns dos problemas existentes, mas não todos. Foi possível perceber que a criação manual de IU é a melhor maneira de satisfazer qualquer requisitos de interface mas não se adequa bem para LPS. Verificou-se também que existem técnicas como a MBUID que resolvem problemas de interface como a composição dos elementos dos PG's mas não discutem maneiras de modelar o posicionamento dos PG's na IU.

6.2 Trabalhos Futuros

Este trabalho abriu portas para novas pesquisas e trabalhos futuros, a seguir algumas sugestões de pesquisas relacionadas:

Integração com o MBUID A técnica *PG-Constraint* utiliza parte dos modelos utilizados pelo método MBUID para facilitar uma futura integração. Investigar uma maneira

de integrar os resultados finais é o primeiro (mais prioritário) trabalho futuro sugerido.

Modelo de representação - tabela de posições A tabela de posições é um elemento muito importante do algoritmo desenvolvido e neste trabalho foi utilizado um único modelo para todos os produtos da LPS. Sugere-se a pesquisa de como modelar esta tabela de modo que seja possível alterar o *layout* da IU para produtos distintos de uma mesma LPS.

Modelo de restrições - tabela de posições Para o caso de extensão de PG's, existem restrições espaciais dependendo do tipo de *layout* utilizado para criar a IU. É sugerido a investigação de um modelo para definir essas restrições de modo a generalizar o algoritmo apresentado.

Adição de novos atributos/funcionalidades Outra forte sugestão é investigar quais outros atributos podem ser adicionados na tabela *PG-Constraint* de modo a aumentar o poder de customização. Para isto é importante primeiro analisar o algoritmo proposto para verificar possíveis conflitos da adição de novos atributos à técnica apresentada.

REFERÊNCIAS

[Botterweck 2006]BOTTERWECK, G. A model-driven approach to the engineering of multiple user interfaces. In: *Proceedings of the 2006 International Conference on Models in Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2006. (MoDELS'06), p. 106–115. ISBN 978-3-540-69488-5. Available from Internet: <http://dl.acm.org/citation.cfm?id=1762828.1762847>.

[2]BOTTERWECK, G.; O'BRIEN, L.; THIEL, S. Model-driven derivation of product architectures. In: *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: ACM, 2007. (ASE '07), p. 469–472. ISBN 978-1-59593-882-4. Available from Internet: <http://doi.acm.org/10.1145/1321631.132171>.

[Clements e Northrop 2001]CLEMENTS, P. C.; NORTHROP, L. *Software Product Lines: Practices and Patterns*. [S.l.]: Addison-Wesley, 2001. (SEI Series in Software Engineering).

[4]EXTENSIBLE Markup Language (XML). 2013. Available from Internet: <www.w3.org/XML/>.

[Frakes e Fox 1995]FRAKES, W. B.; FOX, C. J. Sixteen questions about software reuse. *Communications of the ACM, Volume 38, Issue 6*, ACM, New York, NY, USA, p. 75–ff, 1995.

[Hallvard 2002]HALLVARD, T. Model-based user interface design. 2002.

[Hauptmann et al. 2010]HAUPTMANN, B. et al. *Supporting derivation and customization of user interfaces in software product lines using the example of web applications*. Tese (Doutorado) — Master's thesis, Technische Universität München, 2010.

[Hibbs e Tate 2006]HIBBS, C.; TATE, B. *Ruby on Rails: Up and Running*. [S.l.]: Alta Books, Rio de Janeiro, BR, 2006.

[HTML code example 2013]HTML code example. 2013. Available from Internet: <http://help.websiteos.com/websiteos/example$_{o}f_{as}imple_{h}tml_{p}age.html$>.

[Kang et al. 1990]KANG, K. C. et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. [S.l.], November 1990.

[Matsumoto 2013]MATSUMOTO, Y. *RUBY: A dynamic, open source programming language*. 2013. Available from Internet: <www.ruby-lang.org>.

[Myers 1995]MYERS, B. A. User interface software tools. *ACM Trans. Comput.-Hum. Interact.*, ACM, New York, NY, USA, v. 2, n. 1, p. 64–103, mar 1995. ISSN 1073-0516. Available from Internet: <http://doi.acm.org/10.1145/200968.200971>.

[Pleuss et al. 2012]PLEUSS, A. et al. User interface engineering for software product lines: the dilemma between automation and usability. *EICS '12 Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, ACM, New York, NY, USA, p. 25–34, 2012.

[Pleuss et al. 2012]PLEUSS, A. et al. A case study on variability in user interfaces. *SPLC '12 Proceedings of the 16th International Software Product Line Conference - Volume 1*, ACM, New York, NY, USA, p. 6–10, 2012.

[Pohl e Linden 2005]POHL, K.; LINDEN, G. Böckle Frank van der. *Software Product-Line Engineering - Foundations, Principles, and Techniques*. [S.l.]: Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2005.

[Software Product Line Conference 2013]SOFTWARE Product Line Conference. 2013. Available from Internet: <http://splc.net/>.

[17]SOFTWARE Product Lines Online Tools (SPLOT). 2013. Available from Internet: <http://www.splot-research.org>.

[Stafford 2003]STAFFORD, J. Software Maintenance As Part of the Software Life Cycle. 2003.

[19]STAHL, T.; VOELTER, M.; CZARNECKI, K. *Model-Driven Software Development: Technology, Engineering, Management*. [S.l.]: John Wiley & Sons, 2006. ISBN 0470025700.

[XML code example 2013]XML code example. 2013. Available from Internet: <http://www.w3schools.com/xml/simple.xml>.