

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



**UMA ABORDAGEM PARA DESENVOLVIMENTO DE
LINHAS DE PRODUTOS DE *SOFTWARE* ORIENTADA A
CARACTERÍSTICAS E DIRIGIDA POR MODELOS**

RODRIGO REIS PEREIRA

Uberlândia - Minas Gerais

2010

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



RODRIGO REIS PEREIRA

**UMA ABORDAGEM PARA DESENVOLVIMENTO DE
LINHAS DE PRODUTOS DE *SOFTWARE* ORIENTADA A
CARACTERÍSTICAS E DIRIGIDA POR MODELOS**

Dissertação de Mestrado apresentada à Faculdade de Ciência da Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de *Software*.

Orientador:

Prof. Dr. Marcelo Almeida Maia

Uberlândia, Minas Gerais
2010

Dados Internacionais de Catalogação na Publicação (CIP)

- P436a Pereira, Rodrigo Reis, 1981-
 Uma abordagem para desenvolvimento de linhas de produtos de
 software orientada a características e dirigida por modelos / Rodrigo
 Reis Pereira. - 2010.
 125 f. : il.
- Orientador: Marcelo Almeida Maia.
- Dissertação (mestrado) – Universidade Federal de Uberlândia, Pro-
 grama de Pós-Graduação em Ciência da Computação.
 Inclui bibliografia.
1. Engenharia de software - Teses. 2. Software -
 Desenvolvimento - Teses. I. Maia, Marcelo Almeida. II. Universidade
 Federal de Uberlândia. Programa de Pós-Graduação em Ciência da
 Computação. III. Título.

CDU: 681.3.06

Elaborada pelo Sistema de Bibliotecas da UFU / Setor de Catalogação e Classificação

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Ciência da Computação a aceitação da dissertação intitulada “**Uma Abordagem para Desenvolvimento de Linhas de Produtos de *Software* Orientada a Características e Dirigida por Modelos**” por **Rodrigo Reis Pereira** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 15 de Março de 2010

Orientador:

Prof. Dr. Marcelo Almeida Maia
Universidade Federal de Uberlândia

Banca Examinadora:

Prof. Dr. Eduardo Magno Lages Figueiredo
Universidade Federal de Uberlândia

Dr. Daniel Lucrédio
Universidade de São Paulo

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Março de 2010

Autor: **Rodrigo Reis Pereira**
Título: **Uma Abordagem para Desenvolvimento de Linhas de Produtos de *Software* Orientada a Características e Dirigida por Modelos**
Faculdade: **Faculdade de Ciência da Computação**
Grau: **Mestrado**

Fica garantido à Universidade Federal de Uberlândia o direito de circulação e impressão de cópias deste documento para propósitos exclusivamente acadêmicos, desde que o autor seja devidamente informado.

Autor

O AUTOR RESERVA PARA SI QUALQUER OUTRO DIREITO DE PUBLICAÇÃO DESTE DOCUMENTO, NÃO PODENDO O MESMO SER IMPRESSO OU REPRODUZIDO, SEJA NA TOTALIDADE OU EM PARTES, SEM A PERMISSÃO ESCRITA DO AUTOR.

Dedicatória

Para meus pais Mário e Eudir, meus irmãos Rômulo e Ramon e minha filha Nalanda.

Para toda minha família e meus AMIGOS de hoje e sempre.

Agradecimentos

Inicialmente, gostaria de agradecer a meus pais pelo apoio incondicional e todo suporte, fundamental ao sucesso na conclusão deste trabalho. Agradeço ao meu orientador, professor Marcelo Maia, pela amizade, incentivo e pelos preciosos ensinamentos. Agradeço, também, à Faculdade de Computação da Universidade Federal de Uberlândia pela oportunidade de participar do programa de pós-graduação e à FAPEMIG pelo apoio financeiro. Aos meus amigos de Uberlândia, em especial aos colegas de mestrado. A José Matsumura, Frederico Augusto e Gabriel Coutinho.

“Os loucos abrem caminhos que mais tarde são percorridos pelos sábios.”
(Carlo Dossi)

Resumo

Linhas de Produtos de *Software* são um paradigma emergente para o desenvolvimento de *software*, fundamentado no reuso sistemático de ativos modulares, capaz de aumentar ainda mais as perspectivas de eficiência e produtividade das empresas. Os focos principais deste paradigma são: o aumento da qualidade dos produtos, a escalabilidade, a diminuição dos prazos de entrega, a redução dos custos de produção e manutenção, além da customização em massa. O modelo tem como objetivo ampliar a eficiência e eficácia do processo de desenvolvimento, explorando as similaridades e controlando a variabilidade de características dos produtos membros da família. O Desenvolvimento de *Software* Orientado a Características e Dirigido por Modelos (FOMDD - Feature Oriented Model Driven Development) é uma abordagem para linhas de produtos de *software* que une FOP (Feature Oriented Programming - Programação Orientada a Características) e MDD (Model Driven Development - Desenvolvimento Dirigido por Modelos). MDD utiliza modelos para especificação de programas e transformações sobre modelos para síntese de executáveis. FOP é um paradigma para linhas de produtos de *software* no qual programas são construídos através da composição de características. FOMDD propõe que produtos de uma linha de produtos de *software* sejam construídos pela criação de modelos compostos por características e que estes sejam transformados em executáveis. Este trabalho é uma proposta de generalização para o método de FOMDD apresentado por Trujillo e outros, onde a especificação de domínios de aplicação é feita através de modelos de alto nível, cujos elementos são anotados por características, permitindo a seleção específica de elementos para a configuração de produtos. Este trabalho mostra como um conjunto de modelos de análise pode ser transformado em especificações arquiteturais independentes de plataforma, e estas em código fonte, por transformações que introduzem os detalhes específicos de tecnologia, permitindo a geração de especificações concretas de produtos em diferentes tecnologias, a partir de um mesmo projeto arquitetural. Nosso trabalho mostra como modelos não explicitamente fragmentados em função de características podem ser utilizados como entrada para a geração de artefatos de código orientados a características, permitindo a utilização de modelos legados num processo de desenvolvimento orientado a características e dirigido por modelos. Para a avaliação da abordagem é apresentado um estudo de caso para a geração de sistemas interativos, especificados segundo o padrão arquitetural MVC. Esta geração é fundamentada num arcabouço de transformação que consiste na definição de modelos de análise e modelos de projeto, na definição do formato da especificação de código, bem como na definição e aplicação dos procedimentos de transformação que permitem a obtenção do produto final.

Palavras chave: desenvolvimento de *software* orientado a características, modelagem de características, arquiteturas dirigidas por modelos, metaprogramação, linhas de produto de *software*.

Abstract

Software Product Lines are an emerging paradigm for software development, based on systematic reuse of modular assets, enhancing the perspectives of efficiency and productivity of businesses. The main focuses of this approach are: increasing product quality, scalability, reduced delivery times, reduced costs of production and maintenance, in addition to mass customization. The model aims to expand the efficiency and effectiveness of the development process, exploring the similarities and controlling the variability of member products of a family. The Feature Oriented Model Driven Development (FOMDD) is a blend of FOP (Feature Oriented Programming) and MDD (Model Driven Development). MDD uses models to specify programs and model transformations to synthesize executables. FOP is a paradigm for product lines in which software programs are built from the composition of features. FOMDD proposes the construction of products of a software product line by creating models composed of features and by their transformation into executables. This work is a proposal for a generalization of the FOMDD method presented by Trujillo and others, where the specification of application domains is done with high level models, whose elements are annotated with features, allowing the selection of specific elements for product configuration. This work shows how a set of analysis models can be transformed into platform independent architectural specifications, and these in source code, using transformations which introduce technology specific details, allowing the generation of concrete specifications for products in different technologies, from the same architectural design. Our work shows how models not explicitly fragmented in terms of features can be used as input for the generation of feature oriented code artifacts, allowing the use of legacy models in a feature oriented and model-driven development process. For the evaluation of the approach we present a case study for the generation of interactive systems, specified according to the MVC architectural pattern. This generation is based on a transformation framework that consists on the definition of analysis and design models, on the definition of code specification format, as well the definition and implementation of processing procedures that allow obtaining the final product.

Keywords: feature oriented software development, feature modeling, model driven architectures, metaprogramming, software product lines.

Sumário

Lista de Figuras	xxi
Lista de Tabelas	xxiii
Lista de Abreviaturas e Siglas	xxv
1 Introdução	1
1.1 Técnicas para Implementação de Linhas de Produtos de <i>Software</i>	3
1.2 Caracterização do Problema	10
1.3 Objetivos	11
1.4 Visão Geral da Abordagem Proposta	13
2 Uma abordagem para desenvolvimento orientado a características e di- rigido por modelos	17
2.1 Requisitos para os Metamodelos	21
2.1.1 Metamodelos de Análise	21
2.1.2 Metamodelos de Projeto	24
2.2 Transformações	25
2.2.1 Passos de Transformação Iniciais	25
2.2.2 Via A: Composição de características no nível de código fonte . . .	27
2.2.3 Via B: Composição de características no nível de modelos	29
2.3 Considerações Sobre o Processo de Geração	29
2.3.1 Geração de Linhas de Produtos	29
2.3.2 Geração de Produtos	30
2.4 Resumo do Capítulo	31
3 Estudo de Caso: Linhas de Produtos para Lojas Virtuais	33
3.1 Tecnologia Utilizada	33
3.2 Modelos Utilizados	34
3.2.1 Modelos de Análise	34
3.2.2 Modelos de Projeto	41
3.3 Avaliação do Estudo	44

3.4	Limitações da Validação/Avaliação	46
3.5	Resumo do Capítulo	47
4	Discussão e Trabalhos Relacionados	49
4.1	Avaliação das abordagens	51
4.1.1	Compilação Condicional	51
4.1.2	<i>Frameworks</i> de aplicação	51
4.1.3	<i>Programação Orientada a Aspectos</i>	52
4.1.4	Programação Orientada a Características	53
4.1.5	Programação Orientada a Características com <i>Aspectual-mixin-layers</i>	53
4.1.6	Abordagem de <i>Templates</i> Baseados em Sobreposição de Variantes	54
4.1.7	Desenvolvimento Orientado por Características e Dirigido por Modelos	56
4.1.8	Nossa abordagem para Desenvolvimento Orientado por Características e Dirigido por Modelos (FOMDD*)	56
4.1.9	Desenvolvimento Dirigido por Modelos com <i>Early-Aspects</i>	58
4.1.10	PL-AspectualACME	59
4.1.11	MAS-PL	60
4.2	Comparação de Abordagens	61
5	Conclusões e Perspectivas Futuras	63
	Referências Bibliográficas	67
A	Transformações	73
A.1	$T_{SimplificarModeloEstados}$ (ME)	73
A.2	$T_{SimplificarModeloDados}$ (MD)	73
A.3	$T_{GerarTemplateMapeamentoEstadosCaracteristicas}$ (MES)	74
A.4	$T_{GerarTemplateMapeamentoDadosCaracteristicas}$ (MDS)	74
A.5	$T_{GerarTemplateMapeamentoEstadosLayout}$ (MDS)	75
A.6	Preenchimentos Manuais para Mapeamento de Modelos	75
A.7	$T_{MapearCaracteristicasModeloEstados}$ (MES,ACME)	75
A.8	$T_{MapearCaracteristicasModeloDados}$ (MDS, ACMD)	76
A.9	$T_{MapearLayoutModeloEstados}$ (MEAC, ALME)	77
A.10	$T_{AnaliseParaProjeto}$ (MEACL, MDAC)	77
A.10.1	$T_{DadosParaMVCModelo}$ (MEACL)	77
A.10.2	$T_{EstadosParaVisao}$ (MEACL)	78
A.10.3	$T_{EstadosParaControlador}$ (MEACL)	78

B	Exemplos de Artefatos de Implementação	81
B.1	Artefatos Originais da Aplicação de Referência	81
B.1.1	JSP	81
B.1.2	Javascript	82
B.1.3	Java (Controlador MVC)	85
B.1.4	Java (Modelo MVC)	86
B.2	Artefatos de Geração de Código	86
B.2.1	<i>Template</i> para base JSP (Via A)	86
B.2.2	<i>Template</i> para refinamento JSP (Via A)	88
B.2.3	<i>Template</i> para base Javascript (Via A)	88
B.2.4	<i>Template</i> para refinamento Javascript (Via A)	89
B.2.5	<i>Template</i> para base JAK (Modelo MVC - Via A)	90
B.2.6	<i>Template</i> para refinamento JAK (Modelo MVC - Via A)	91
B.3	Artefatos de Código Gerados	92
B.3.1	Base JSP (Via A)	92
B.3.2	Refinamento JSP (Via A)	92
B.3.3	Composição final JSP (Via A)	93
B.3.4	Base Javascript (Via A)	93
B.3.5	Refinamento Javascript (Via A)	93
B.3.6	Base JAK (Modelo MVC - Via A)	94
B.3.7	Refinamento JAK (Modelo MVC - Via A)	95
B.3.8	Composição final JAK (Modelo MVC - Via A)	95

Lista de Figuras

1.1	Relação entre módulos e características	4
1.2	Exemplo de código utilizado em compilação condicional	5
1.3	<i>Mixin-layers</i> adaptado de [Smaragdakis e Batory 1998]	6
1.4	Composição de diretórios e artefatos de [Batory 2003]	7
1.5	Abordagem FOMDD de [Trujillo et al. 2007]	10
1.6	Visão geral de nossa abordagem FOMDD - caminho de síntese A	14
1.7	Visão geral de nossa abordagem FOMDD - caminho de síntese B	15
2.1	Nossa abordagem FOMDD - caminho de Síntese A	19
2.2	Nossa abordagem FOMDD - caminho de Síntese B	20
2.3	Metamodelo para diagramas de características	22
2.4	Fragmento do metamodelo para diagramas de estados	23
2.5	Metamodelo para diagramas de dados	24
3.1	Fragmento 1 do modelo de características PIM do domínio <i>Lojas Virtuais</i> .	35
3.2	Fragmento 2 do modelo de características PIM do domínio <i>Lojas Virtuais</i> .	36
3.3	Modelo de características PSM do domínio <i>Lojas Virtuais</i>	36
3.4	Fragmento 1 do modelo de características PIM de produto	37
3.5	Fragmento 2 do modelo de características PIM de produto	37
3.6	Modelo de características PSM de produto	38
3.7	Fragmento do modelo de estados do domínio de aplicação <i>Lojas Virtuais</i> .	38
3.8	Exemplo de tela de aplicação	38
3.9	Fragmento do modelo de dados do domínio de aplicação <i>Lojas Virtuais</i> .	39
3.10	Exemplo de especificação de <i>layout</i>	40
3.11	Fragmento de <i>template</i> de mapeamento estados x características	40
3.12	Fragmento de <i>template</i> de mapeamento dados x características	41
3.13	Fragmento de <i>template</i> de mapeamento estados x <i>layout</i>	41
3.14	Fragmento de representação arquitetural abstrata MVC - <i>Modelo</i>	42
3.15	Fragmento de representação arquitetural abstrata MVC - <i>Visão</i>	43
3.16	Fragmento de representação arquitetural abstrata MVC - <i>Controle</i>	43
4.1	Aspectual mixin-layers	54

4.2 Sobreposição de variantes de [Czarnecki 2005] 55

Lista de Tabelas

3.1	Métricas de Linhas de Código	46
4.1	Propriedades da Compilação Condicional	51
4.2	Propriedades de <i>Frameworks</i>	52
4.3	Propriedades da Programação Orientada a Aspectos	53
4.4	Propriedades da Programação Orientada a Características	54
4.5	Propriedades da Programação Orientada a Características com <i>Aspectual-mixin-layers</i>	55
4.6	Propriedades da Superimposição de Variantes	56
4.7	Propriedades do Desenvolvimento Orientado por Características e Dirigido por Modelos	57
4.8	Propriedades da nossa Abordagem para Desenvolvimento Orientado por Características e Dirigido por Modelos	58
4.9	Propriedades do Desenvolvimento Dirigido por Modelos com <i>Early-Aspects</i>	59
4.10	Propriedades de PL-AspectualACME	60
4.11	Propriedades de MAS-PL	61
4.12	Tabela Comparativa	62
4.13	Tabela Comparativa FOMDD e FOMDD*	62

Lista de Abreviaturas e Siglas

AAD	Artefatos Arquiteturais do Domínio
AAP	Artefatos Arquiteturas do Produto
ACD	Artefatos de Código do Domínio
ACMD	Anotações de Características do Modelo de Dados do Domínio
ACME	Anotações de Características do Modelo de Estados do Domínio
ACP	Artefatos de Código do Produto
ADPIM	Modelo de Arquitetura Independente de Plataforma do Domínio
ALME	Anotações de <i>Layout</i> do Modelo de Estados do Domínio
APPIM	Modelo de Arquitetura Independente de Plataforma do Domínio
DPIM	Modelo de Características Independentes de Plataforma do Domínio
DPSM	Modelo de Características Específicas de Plataforma do Domínio
MAD	Modelo de Análise do Domínio
MAP	Modelo de Análise do Produto
MD	Modelo de Dados do Domínio
MDAC	Modelo de Dados Anotado por Características
MDC	Modelo de Mapeamento Dados x Características do Domínio
MDS	Modelo de Dados do Domínio Simplificado
ME	Modelo de Estados do Domínio
MEAC	Modelo de Estados Anotado por Características
MEACL	Modelo de Estados Anotado por Características e <i>Layout</i>
MEC	Modelo de Mapeamento Estados x Características do Domínio
MEL	Modelo de Mapeamento Estados x <i>Layout</i> do Domínio
MES	Modelo de Estados do Domínio Simplificado
ML	Modelo de <i>Layout</i> do Domínio
MM	Modelos de Mapeamento
MPD	Modelo de Projeto do Domínio
TMDC	<i>Template</i> de Mapeamento de Dados x Características
TMEC	<i>Template</i> de Mapeamento de Estados x Características
TMEL	<i>Template</i> de Mapeamento de Estados x <i>Layout</i>
PDOC	Projeto de Código Orientado a Característica do Domínio

PP	Projeto de Código do Produto
PPIM	Modelo de Características Independentes de Plataforma do Produtos
PPSM	Modelo de Características Específicas de Plataforma do Produto

Capítulo 1

Introdução

Um dos principais objetivos da Engenharia de *Software* é gerenciar e controlar a complexidade dos sistemas. Neste contexto, considerando o baixo nível de rastreabilidade entre os artefatos utilizados ao longo dos processos tradicionais de desenvolvimento, a crescente complexidade dos sistemas e, consequentemente, a maior dificuldade na compreensão das estruturas de implementação, cada vez mais, técnicas avançadas de desenvolvimento se fazem necessárias para auxílio aos profissionais envolvidos tanto nas tarefas de modelagem quanto nas tarefas de codificação [Batory 2003].

A forma como bens de consumo tem sido produzidos vem mudando significativamente com o decorrer do tempo. Anteriormente, mercadorias eram feitas artesanalmente para clientes individuais. Aos poucos, o número de pessoas que poderiam possuir vários tipos de produtos aumentou. No domínio dos automóveis isto levou à invenção das linhas de produção, o que permitiu a produção em larga escala para um mercado de massa, com menor custo do que a criação de produtos individuais de forma artesanal. A grosso modo, ambos os tipos de produtos, individuais e aqueles produzidos em massa podem ser identificados no domínio do desenvolvimento de *software*, ou seja, *software* personalizado e *software* padronizado. No entanto, cada uma destas formas de produção apresenta alguns inconvenientes. Produtos de *software* individuais são caros, enquanto produtos de *software* padronizados apresentam diversificação limitada [Pohl et al. 2005].

Nos primórdios da computação a produção de *software* começou com programas pequenos para a implementação de algoritmos. Contudo, desde então, o domínio de problemas que podem ser suportados por *software* vem crescendo constantemente. Da mesma forma, os sistemas sendo construídos também vem se tornando cada vez mais complexos. Neste contexto, a reutilização de *software* se tornou cada vez mais importante em uma variedade de aspectos para a Engenharia de *Software*. O reconhecimento do fato de que vários sistemas de *software* contêm porções de código similares (ou mesmo idênticas), anteriormente desenvolvidas a partir do zero ou mesmo através de técnicas rudimentares de reuso como “copiar e colar”, levou a esforços para a reutilização de componentes.

Componentes possuem uma interface, encapsulam detalhes internos e, em geral, são

documentados separadamente. O reuso de *software* através de componentes tem grande impacto na estrutura dos sistemas e na forma como eles são construídos, o que consequentemente, propicia uma série de vantagens. É possível distribuir o desenvolvimento de diferentes componentes entre diferentes equipes, de forma a permitir desenvolvimento paralelo. Também, a manutenção dos sistemas torna-se mais simples quando as interfaces dos componentes estão claramente definidas, pois as mudanças podem ser feitas localmente sem causar interferências globais. Além disso, quando as inter-relações entre componentes estão claramente documentadas, a troca e incorporação de novos componentes em um sistema torna-se mais fácil.

As iniciativas de reuso na construção de *software* através do uso de componentes fundamentados principalmente no desenvolvimento orientado a objetos tiveram impulso na década de 80. O sucesso destas atividades proporcionou o surgimento de abordagens de reuso sistemático nas diversas etapas do processo de desenvolvimento de *software* incluindo subprodutos de trabalho como documentos, especificações e modelos, capazes de aumentar ainda mais as perspectivas de eficiência e produtividade das empresas. Os focos principais deste tipo de abordagem são o aumento da qualidade dos produtos, a escalabilidade, a diminuição dos prazos de entrega, a redução dos custos de produção e manutenção, além da customização em massa.

A evolução destas ideias levou a formulação do modelo de Linhas de Produtos de *Software*, que apresenta um deslocamento no foco do paradigma tradicional de desenvolvimento. Neste modelo, as organizações que anteriormente abordavam o desenvolvimento de produtos individuais passaram a concentrar seus esforços na criação e manutenção de uma coleção de produtos similares pertencentes a uma família. O modelo tem como objetivo ampliar ao máximo a eficiência e eficácia do processo de desenvolvimento, explorando as similaridades e controlando a variabilidade de características dos produtos membros da família.

Artefatos de *software* são elementos de alta complexidade e, consequentemente, a construção destes implica em uma série de desafios relacionados aos processos de engenharia que possibilitam a obtenção dos mesmos. Em geral, os esforços de aperfeiçoamento destes processos tiveram um foco maior na engenharia de produtos individuais para produção em massa, pois assim como a industrialização de processos de manufatura tradicionais possibilitou aumento de produtividade e qualidade com redução de custos de produção. Também a automação do processo de desenvolvimento através da composição de componentes reutilizáveis de *software* pode possibilitar a obtenção de resultados semelhantes.

Entretanto, na indústria tradicional, a produção em massa teve que passar por ajustes para satisfazer as necessidades individuais dos clientes através da customização em massa [Davis 1987]. Este conceito se firmou com a caracterização de linhas de produtos, as quais correspondem a distintos produtos mas que compartilham uma mesma plataforma e possuem características similares. Um exemplo típico é uma linha de produtos

automotivos que compartilham uma mesma plataforma, mas que permitem um grande número de configurações de carroceria, motor, acabamento, opcionais, etc.

A indústria de *software* tem tido um movimento neste mesmo sentido. Uma Linha de Produtos de *Software* é fundamentada em um conjunto de produtos que possuem funcionalidades em comum, mas que no entanto apresentam outras diferenciadas. Os mecanismos para implementação de linhas de produtos de *software* são mais complexos que de produtos individuais e se enquadram dentro da área de *Reuso de Software*. Uma característica importante de linhas de produtos de *software* é a utilização de técnica de reuso sistemático ao contrário da reutilização oportunista de *software*. A construção de produtos específicos ocorre através da composição de ativos reutilizáveis com artefatos que expressam a variabilidade de características dos produtos de uma linha. De fato, a engenharia de linhas de produtos de *software* tem se mostrado uma metodologia adequada para o desenvolvimento de uma diversidade de produtos a custos mais baixos, com menor tempo de desenvolvimento e maior qualidade [Pohl et al. 2005].

1.1 Técnicas para Implementação de Linhas de Produtos de *Software*

A importância de se construir *software* modularmente já é reconhecida há tempos na Engenharia de *Software*. Entre outros benefícios, modularidade torna mais simples entender, evoluir, usar e prover escala aos sistemas [Parnas 1972]. Linhas de Produtos de *Software* oferecem um paradigma cujo foco de desenvolvimento considera a construção de um conjunto de produtos similares por meio de módulos reutilizáveis, mas com base na variabilidade de características. Segundo Batory e outros [Batory et al. 2003], características são unidades de incremento (refinamento) da funcionalidade de programas pelas quais diferentes produtos podem ser distinguidos e definidos em uma linha de produtos de *software*.

Características nos permitem expressar as semelhanças e as diferenças entre instâncias de um conceito. Elas são fundamentais para a formulação de descrições concisas de conceitos com alto índice de variação entre suas instâncias e podem ser implementadas de diferentes formas [Czarnecki e Eisenecker 2000]. Várias metodologias com diferentes capacidades e limitações foram propostas para implementação de linhas de produtos de *software* baseadas em características, entre elas: camadas [Batory e O'Malley 1992], módulos de características [Kang et al. 1990] [Prehofer 1997] [Prehofer 2001], metaclasses [Forman 1998], colaborações [Reenskaug et al. 1992] [Smaragdakis e Batory 1999], temas¹ [Harrison e Ossher 1993], aspectos [Kiczales e Mezini 2005], e interesses² [Tarr

¹Do inglês, *subjects*

²Do inglês, *concerns*

et al. 1999].

Em geral, a realização de uma característica está espalhada em diferentes módulos de uma aplicação [Sobreira e Maia 2008]. Além disso, um módulo real de construção (exemplo: classe, pacote, etc) participa da implementação de uma ou mais características como mostrado na Figura 1.1.

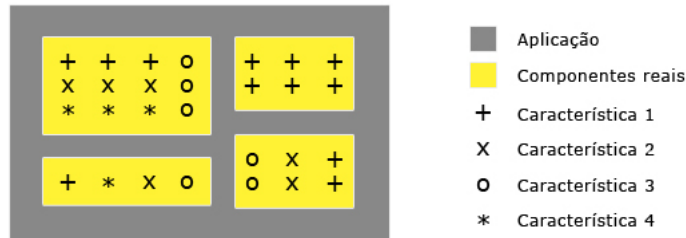


Figura 1.1: Relação entre módulos e características

Linguagens de programação com suporte a orientação a objetos, como C++ e Java, são as principais tecnologias de implementação empregadas pelas empresas atualmente. Os mecanismos básicos de implementação de variabilidade disponíveis em linguagens como estas são baseados em parametrização, herança, *frameworks* e componentes. As metodologias mais avançadas baseadas em orientação a objetos não são puramente baseadas em objetos e classes, mas em componentes, que podem ser considerados um conjunto encapsulado de classes inter-relacionadas [Batory 2000].

A Engenharia de Software Baseada em Componentes [Altunel e Tolun 2007] se refere ao desenvolvimento de sistemas de *software* através de componentes reutilizáveis e o seu principal papel é endereçar o desenvolvimento de sistemas como uma junção entre partes (componentes), o desenvolvimento de partes como entidades reutilizáveis e a manutenção/evolução de sistemas pela customização e substituição de tais partes. Isto requer o uso de tecnologias estabelecidas e suporte ferramental capaz de abranger completamente o ciclo de vida do componente e do sistema incluindo aspectos tecnológicos, organizacionais, etc [Crnkovic 2003].

Não existe um consenso a respeito da definição do conceito de componente. Em um sentido mais amplo, um componente é uma peça reusável de *software* com interface bem definida. Por esta perspectiva podemos considerar classes e/ou conjuntos de classes das linguagens de programação orientadas a objetos como componentes. Contudo, outras definições impõem requisitos adicionais aos componentes: eles devem ser implantáveis de forma independente, abstraídos de suas dependências de outros componentes, abstraídos de *middleware*, etc [Pohl et al. 2007].

Embora seja possível construir linhas de produtos de *software* através de componentes orientados a objetos, esta metodologia apresenta algumas desvantagens: maior tempo e esforço necessários ao desenvolvimento de componentes reusáveis, gerência ineficiente

e não preditiva de requisitos, conflitos entre usabilidade e reusabilidade, alto custo de manutenção de componentes e alta sensibilidade a mudanças [Crnkovic 2003].

A Compilação Condicional [Hu et al. 2000] [Gacek e Anastasopoulos 2001] [Alves et al. 2006] [Adams et al. 2009] é uma abordagem simples e altamente utilizada para a implementação de mecanismo de variabilidade em linguagens como C e C++, e pode ser utilizada na construção de linhas de produtos de *software*. Como mostrado na Figura 1.2, esta técnica utiliza segmentos de código variáveis, opcionais e alternativos que são identificados por uma série de diretivas de pré-processamento.

```
// código independente de plataforma

#ifdef WINDOWS
    Código específico para Windows
#endif

#ifdef MACINTOSH
    Código específico para Macintosh
#endif

#ifdef UNIX
    Código específico para Unix
#endif
```

Figura 1.2: Exemplo de código utilizado em compilação condicional

A Compilação Condicional implementa variabilidade por meio de parametrização e é normalmente utilizada em conjunto com *scripts* de construção. Estes *scripts* definem diferentes configurações para construção de programas por meio da especificação do conjunto de arquivos a serem compilados e valores para os parâmetros de entrada. O principal problema com este tipo de abordagem diz respeito ao alto grau de entrelaçamento entre os módulos. Por não utilizar um mecanismo de separação modular explícito, este entrelaçamento frequentemente acarreta código fonte altamente complexo e de difícil compreensão.

Uma alternativa para implementação de Linhas de Produtos de *Software*, é através da utilização de *frameworks* de aplicação orientados a objetos convencionais. Um *framework* é um conjunto de classes abstratas e concretas que incorporam um projeto abstrato. Uma instância de um *framework* é um conjunto de classes concretas que implementam classes abstratas para prover um sub-sistema executável. *Frameworks* são desenvolvidos para reuso: classes abstratas que encapsulam código comum e classes concretas que encapsulam código específico para instâncias. Entretanto, este delineamento de código reusável versus código específico é problemático. Classes concretas de diferentes instâncias de um *framework* podem conter muito em comum e podem haver variações em classes abstratas, que levam à replicação desnecessária de código [Batory et al. 2000]. *Frameworks* de aplicação permitem reuso de classes abstratas mas não permitem a especificação de coleções de classes concretas que possam ser utilizadas para a construção de uma aplicação. Ou seja, eles permitem o reuso de esqueletos, não de componentes individuais.

Em abordagens orientadas a objetos, o refinamento de uma classe é encapsulado por uma subclasse através de herança. O desafio na construção de linhas de produtos de *software* é proporcionar escala à herança de refinamentos individuais para refinamentos de larga-escala, isto é, a possibilidade de se aplicar herança a módulos de construção compostos.

Uma alternativa eficiente, que considera este princípio para implementação de linhas de produtos de *software*, é a chamada *mixin-layers* [Smaragdakis e Batory 1999]. *Mixin-layers* é uma solução que utiliza como base uma construção orientada a objetos chamada *mixin*, que é uma classe especialmente desenvolvida para ser uma das classes a serem utilizadas num esquema de herança múltipla, por meio de parametrização da superclasse. Para implementar colaborações completas como componentes independentes é necessária a utilização de *mixins* que encapsulam outras *mixins* (i.e., *mixin-layers*), como mostrado na Figura 1.3 a seguir.

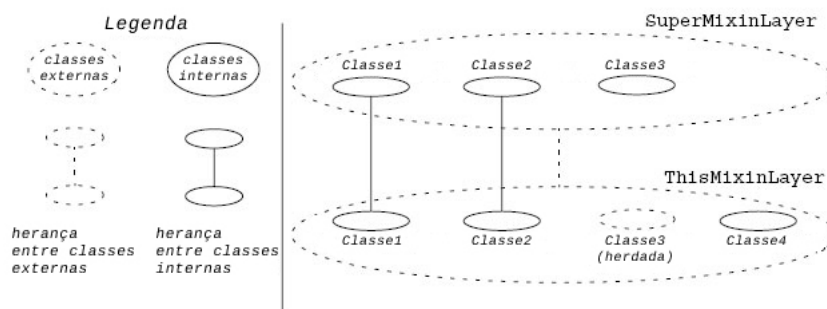


Figura 1.3: *Mixin-layers* adaptado de [Smaragdakis e Batory 1998]

Mixin-layers se enquadram de maneira eficaz no refinamento gradual de aplicações por se tratarem de componentes reusáveis e intercambiáveis. Porém, esta forma em geral é de difícil especificação nas linguagens de programação convencionais e apresenta algumas falhas. Por exemplo, o código que implementa um característica está espalhado em vários trechos, o que leva a um alto índice de entrelaçamento, dificultando assim a manutenção. Além disso, a técnica de implementação é baseada em herança parametrizada o que resulta na necessidade de utilização de *templates* C++ ou de especificações de *templates* fora de padrão para linguagens como Java. Tais linguagens não possuem um mecanismo explícito para implementação de herança múltipla.

A Programação Orientada a Aspectos [Kiczales e Mezini 2005] também pode ser usada na construção de linhas de produto de *software*. Ela oferece um paradigma que visa obter melhor separação de interesses e, assim, facilitar o desenvolvimento de aplicações proporcionando maior apoio para a modularização nos níveis de projeto e implementação. Em [Pulvermüller et al. 2000], é proposta uma abordagem alternativa, onde classes são

projetadas e codificadas separadamente do código transversal aos objetos, que é encapsulado em construções independentes conhecidas como aspectos. Um elemento de união, chamado *weaver*, é responsável pela junção entre classes e aspectos em tempo de compilação ou execução.

A programação orientada a aspectos pode ajudar a evitar o entrelaçamento de código, aumentar a reusabilidade e facilitar a manutenção [Pulvermüller et al. 2000]. Um dos principais problemas com abordagens baseadas em orientação a aspectos diz respeito à depuração. Enquanto no nível sintático os elementos de código de um sistema aparecem em módulos separados, eles trabalham em conjunto em tempo de execução, o que resulta em dificuldades na compreensão global do sistema por meio das partes constituintes.

Uma alternativa que atende satisfatoriamente aos propósitos de *Linhas de Produtos de Software*, fundamentada em Modelagem de Características [Czarnecki e Eisenecker 2000] e Refinamentos Sucessivos [Dijkstra 1997], é chamada Programação Orientada a Características [Batory et al. 2003], [Prehofer 1997]. Ela considera características como entidades de primeira classe do projeto e implementação, tornando mais simples adicionar e remover características em aplicações [Batory 2003]. Em [Batory et al. 2003], é apresentado o ambiente AHEAD, que utiliza um mecanismo que captura *mixin-layers* explicitamente, onde um programa pode conter diferentes tipos de representações, como código fonte, documentos UML, *makefiles* e cada representação é escrita em linguagem própria, como Java, *bytecode*, XML, etc. Quando uma característica é adicionada a um programa, quaisquer ou todas as representações do programa podem ser atualizadas (i.e., extensões aplicadas não apenas a código fonte). Em AHEAD, um módulo é uma hierarquia de composição de artefatos relacionados e é implementado através de estruturas de diretórios. A Figura 1.4, identifica a obtenção de uma característica C, resultante da composição de duas outras características (A e B), representadas por pastas do sistema de arquivos, onde a combinação é feita através de um componente de composição específico para cada tipo de artefato. Assim, uma característica pode encapsular, por exemplo, vários arquivos Java, HTML, XML, etc.

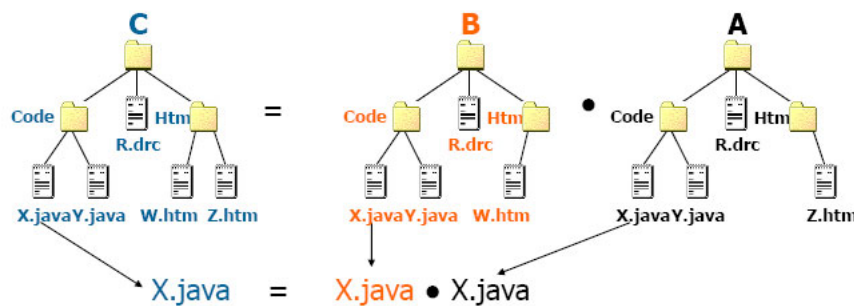


Figura 1.4: Composição de diretórios e artefatos de [Batory 2003]

Em AHEAD, programas são valores e extensões são funções. Por exemplo, os seguintes valores representam programas com diferentes características:

```
f      // programa com característica f
g      // programa com característica g
```

Uma extensão de programa é uma função que toma um programa como entrada e produz um programa refinado como saída:

```
i • x      // adiciona a característica i ao programa x
j • x      // adiciona a característica j ao programa x
```

Uma aplicação com múltiplas características é uma *equação* que é uma expressão nomeada. Diferentes equações definem uma família de produtos, como por exemplo:

```
app1 = i • f      // app1 é constituída pelas características i e f
app2 = j • g      // app2 é constituída pelas características j e g
app3 = i • j • f  // app3 é constituída pelas características i, j e f
```

Diferentes técnicas podem ser utilizadas para a realização de variabilidade de artefatos. Entretanto, algumas delas não se adequam perfeitamente à síntese automática de programas por necessitarem de intervenção humana durante o processo de geração que caracteriza a composição (ex.: configuração de parâmetros para compilação condicional). AHEAD suporta programação composicional organizando os artefatos em uma estrutura geral que permite a geração de produtos por meio de composição de partes independentes. Este tipo de composição horizontal de artefatos, também dita transformação endógena, é fundamental para a realização de variabilidade em linhas de produtos de *software*.

Embora a Programação Orientada a Características se enquadre perfeitamente ao desenvolvimento de linhas de produtos de *software*, esta é ainda uma solução incipiente e de baixo nível de abstração. Em [Trujillo et al. 2007] (Figura 1.5) é mostrada uma forma de se obter a realização de características em alto nível de abstração que utiliza uma adaptação do paradigma de construção de *software* dirigido por modelos (MDA³ [Kleppe et al. 2003]) para o contexto de Programação Orientada a Características. Assim, as técnicas de realização de variabilidade lidam não somente com código mas também diretamente com modelos. Isto permite a composição dos ativos reutilizáveis por meio de um conjunto de modelos ao invés de um conjunto de artefatos de implementação.

O Desenvolvimento Dirigido por Modelos é fundamentado em transformações entre modelos para aumentar o nível de abstração e de reuso. A especificação de programas em MDA é feita através de um ou mais modelos independentes de plataforma (PIMs⁴) que definem um programa alvo. Transformações convertem modelos independentes de

³Do inglês, *Model Driven Architecture*

⁴Do inglês, *Platform Independent Models*

plataforma em modelos específicos de plataforma (PSMs⁵), onde definições específicas de tecnologia são introduzidas. Tais derivações, também conhecidas como transformações exógenas, mapeiam modelos expressos em diferentes linguagens e deslocam o foco de desenvolvimento de *software* da implementação de código para a modelagem.

FOMDD⁶ é um paradigma de programação que une Programação Orientada a Características e Desenvolvimento Dirigido por Modelos, no qual modelos podem ser refinados pela composição de características (i.e., transformações endógenas que mapeiam modelos expressos em uma mesma linguagem), e podem ser derivados de outros modelos (i.e., transformações exógenas que mapeiam modelos expressos em diferentes linguagens). FOMDD envolve modelagem e composição de características bem como a transformação destes modelos com o objetivo final de obtenção do código executável. O estudo de caso apresentado em [Trujillo et al. 2007], mostra por exemplo, uma forma de se obter *portlets* (componentes de portais *web*) [Díaz et al. 2007] através da utilização de diagramas de estados a serem transformados em modelos intermediários refináveis que antecedem o processo de geração de código. Naquele estudo, o processo de desenvolvimento inicia com a especificação de um diagrama de estados que representa o fluxo de computação do componente (sendo cada estado a representação abstrata de uma característica). Este diagrama é posteriormente transformado em modelos intermediários, em conformidade com uma linguagem de domínio específico para *portlets*, definindo a estrutura básica de implementação de cada um dos estados, segundo arquitetura de código previamente definida. Estes modelos intermediários são refinados pela combinação com modelos de configuração específicos de plataforma escritos manualmente, completando a especificação e possibilitando a geração de código funcionalmente completo. No estudo de caso apresentado, a geração de código para *portlets* pode ser executada em duas diferentes vias (Figura 1.5 - vias A e B) e em ambas estas vias é caracterizada por 6 passos de transformação. Na primeira delas (A): (1) geração do esqueleto arquitetural do domínio a partir do modelo de estados, (2) refinamento manual do esqueleto arquitetural do domínio em linguagem específica para *portlets*, (3) geração de código orientado a características para o domínio de aplicação, (4) refinamento manual do código orientado a características do domínio (5) composição de características de produto no nível de código (6) transformações finais para obtenção de artefatos compiláveis de produto (ex.: `jak2java`⁷). Na segunda via (B): (1) composição de características de produto no nível de modelagem, (2) geração do esqueleto arquitetural a partir do modelo de estados do produto, (3) refinamento manual do esqueleto arquitetural de produto em linguagem específica para *portlets*, (4) geração do código de produto, (5) refinamentos manuais do código de produto (6) transformação em artefatos compiláveis de produto.

⁵Do inglês, *Platform Specific Models*

⁶Do inglês, *Feature Oriented Model Driven Development*

⁷<http://userweb.cs.utexas.edu/users/schwartz/ATS/fopdocs/j2j.html>

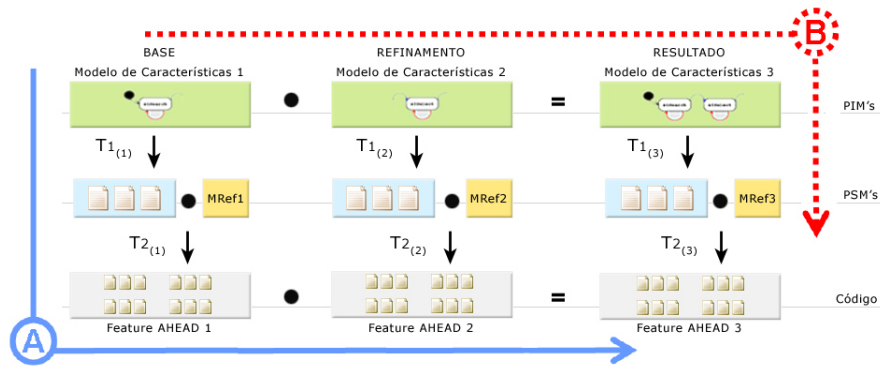


Figura 1.5: Abordagem FOMDD de [Trujillo et al. 2007]

1.2 Caracterização do Problema

O trabalho apresentado por [Trujillo et al. 2007], mostrou uma maneira adequada para realização de características por meio de transformações de modelos. Este trabalho mostra como a composição de ativos reutilizáveis pode ser obtida de forma automatizada e em alto nível de abstração.

Entretanto, a abordagem ainda apresenta algumas deficiências. Dentre as especificações de entrada utilizadas no processo, apenas uma delas (diagramas de estados) é independente de plataforma. As demais, que expressam a lógica de negócio e de visão, são específicas de plataforma (i.e., *portlets*), o que inicialmente inviabiliza a criação de produtos pertencentes a outras plataformas. Além disso, o modelo de diagramas de estados utilizado é insuficiente para a geração integral dos elementos de implementação que definem as características. Este fato exige que refinamentos manuais específicos de plataforma sejam aplicados nos passos intermediários de geração.

O tema principal desta pesquisa está relacionado ao desenvolvimento de famílias de produtos de *software*, através do uso intensivo de modelos. Mais especificamente, este trabalho diz respeito à generalização dos modelos utilizados na técnica de FOMDD apresentada por [Trujillo et al. 2007]. Entre as principais questões relacionadas ao trabalho estão as seguintes:

- Como representar através de modelos, os elementos essenciais de construção de uma linha de produtos de *software*?

Em [Trujillo et al. 2007], a representação de características é feita através de um modelo de estados que representa o fluxo de computação em um *portlet*. Os detalhes restantes de implementação, como persistência de dados e organização visual de elementos (*layout*), são introduzidos por meio de transformações específicas de plataforma. Idealmente, todas as entradas no processo deveriam ser derivadas de modelos independentes de plataforma [Trujillo et al. 2007].

- Como representar a arquitetura de famílias de sistemas de *software* através de mo-

delos?

O estudo de caso apresentado em [Trujillo et al. 2007] é específico para a geração de *portlets*. Assim, as transformações que definem os passos de geração são específicas para a transformação dos modelos de alto nível em elementos de implementação desta plataforma. No entanto, a utilização de um modelo intermediário de arquitetura pode prover mais robustez no processo, permitindo que a geração de sistemas seja feita para diferentes plataformas, a partir de um mesmo conjunto consistente de modelos de entrada. Por exemplo, programas especificados segundo o padrão arquitetural MVC⁸, poderiam ser gerados em diferentes plataformas como JavaSE⁹, JavaEE¹⁰, PHP¹¹, etc.

- Qual o impacto da ordem de aplicação da composição horizontal (composição de características) em relação ao refinamento vertical (refinamento para o nível de implementação)?

No estudo de caso de [Trujillo et al. 2007], são apresentados dois possíveis caminhos para a síntese de produtos. O primeiro deles (Figura 1.5, caminho A), utiliza transformações sucessivas entre modelos para obtenção de código orientado a características. Este código ao final do processo, é combinado através dos mecanismos de composição do ambiente AHEAD, para configuração do produto. A segunda alternativa para síntese (Figura 1.5, caminho B), utiliza a composição de características por meio de modelos de alto nível, definindo em alto nível de abstração a configuração do produto alvo. Dessa maneira, a tradução dos modelos abstratos em código fonte não precisa necessariamente ser feita para linguagens com suporte a composição horizontal de artefatos no nível de código.

- Como se dá a evolução de sistemas gerados através da abordagem apresentada?

Esta questão está relacionada ao refinamento de domínios de aplicações pela adição de novas características e aos impactos causados pelas modificações sobre os modelos de entrada, bem como aos processos necessários para a geração e incorporação de uma nova característica ao conjunto de ativos reusáveis.

1.3 Objetivos

O objetivo deste trabalho é investigar os benefícios da utilização de uma técnica de automação de parte do processo de desenvolvimento de *software* baseada em características e modelos. O uso desta técnica pode ser capaz de aumentar a robustez no processo de geração de famílias de produtos através da automação de tarefas tediosas no desenvolvi-

⁸Do inglês, *Model-View-Controller*

⁹<http://java.sun.com/javase/>

¹⁰<http://java.sun.com/javaee/>

¹¹<http://www.php.net/>

mento de *software*. Mais especificamente, este trabalho pretende generalizar o método de FOMDD apresentado em [Trujillo et al. 2007] para a especificação de linhas de produtos de *software*, através da definição de novos mecanismos, descritos a seguir:

- Definição de modelos de análise para representação dos elementos básicos de construção de linhas de produtos de *software* baseados na interação com o usuário por meio de interfaces gráficas.

Considerando o objetivo de reduzir o grau de intervenção humana nas etapas intermediárias de transformação para geração de produtos, além de eliminar a necessidade de se aplicar refinamentos específicos de plataforma durante estas etapas, é necessária a consolidação de um modelo de análise consistente, expresso por meio de modelos independentes de plataforma, a partir dos quais seja possível produzir elementos de implementação em diferentes plataformas alvo.

Estes modelos devem ser descritos através de UML¹², por se tratar de uma linguagem de modelagem de propósito geral, extensível e passível de utilização na especificação de sistemas pertencentes a diferentes domínios de aplicação, capaz de definir a estrutura, comportamento, arquitetura, processos de negócio e estruturas de dados que caracterizam as aplicações. No entanto, a UML padrão é limitada, e alcançar o nível semântico necessário para a especificação de linhas de produtos, só é possível através da definição de *profiles* específicos.

- Definição de modelos de análise complementares para a especificação da organização visual (*layout*) de elementos nas interfaces gráficas das aplicações.

O modelo de estados utilizado deve contemplar estados (telas), subestados (possíveis configurações de elementos nas telas) e elementos que os compõem (*widgets*), contudo, não deveria necessariamente considerar a organização visual destes elementos nas telas. Assim, este modelo pode definir a estrutura para organização visual de elementos, por meio da identificação de áreas (i.e., *containers*) a serem preenchidas por elementos de interação.

- Definição de modelos de análise complementares para a especificação do relacionamento entre as características e os elementos do modelo de estados e do modelo de dados, bem como o relacionamento entre elementos de interação (do modelo de estados) e *layout*.

A estratégia adotada em nossa abordagem utiliza alguns modelos não orientados a características como entrada para o processo de geração (i.e., modelo de dados e estados). A especificação destes modelos deveria ser a mais abrangente possível e inicialmente não deve possuir modularização baseada em características. No entanto, como a definição dos diferentes produtos de uma linha é obtida em função de

¹²<http://www.uml.org/>

características, é necessária a associação entre os elementos constituintes do modelo de características e dos modelos complementares. Para isso, devem ser definidos modelos de mapeamento, que correlacionam entidades dos modelos (ex.: classes, campos, métodos, estados, transições, etc) às características que definem a linha de produtos.

Ainda, para uma completa especificação da interface gráfica com o usuário, um terceiro modelo de mapeamento é utilizado para associar elementos de interação com o usuário a áreas pré-definidas na estrutura de organização visual.

- Definição de modelos intermediários para representação abstrata de arquitetura. Estes modelos visam possibilitar a geração de produtos para diversas plataformas específicas baseadas em um mesmo padrão arquitetural, bem como facilitar a tradução de modelos orientados a características para linguagens sem suporte a composição horizontal de artefatos.

- Definição completa do arcabouço de transformação necessário para a geração de produtos a partir dos modelos de alto nível.

Consiste na definição dos metamodelos a serem utilizados em diferentes níveis de transformação, bem como na especificação das transformações necessárias ao processo de geração baseado na tradução entre modelos, que ao final deve possibilitar a obtenção de *templates* de código (ou seja, esqueletos semi-funcionais) para as aplicações.

1.4 Visão Geral da Abordagem Proposta

Nossa abordagem utiliza modelos de análise abstratos como entrada para o processo de geração, que ocorre através de transformações sucessivas entre modelos. Sucintamente, estes modelos de análise são convertidos em modelos abstratos de projeto (representação arquitetural) que são finalmente transformados em *templates* de código através de transformações que encapsulam detalhes específicos de plataforma.

O estudo de caso realizado como experimento de avaliação da nossa abordagem (Capítulo 3) visa a geração de sistemas interativos, especificados segundo o padrão arquitetural MVC e cuja implementação é expressa segundo a plataforma JEE. O processo de geração de produtos pode ser executado por duas diferentes vias de geração. Na primeira delas, como na via A da Figura 1.5, a combinação de características é feita por transformações endógenas no nível de código. Esta alternativa requer uma linguagem orientada a características para a combinação das características selecionadas, e o processo de geração que precede sua obtenção consiste em sucessivas transformações exógenas que convertem elementos do modelo de análise do domínio de aplicação em elementos de implementação orientados a características. O processo pode ser resumido em quatro etapas, como identi-

ficado pelas atividades da Figura 1.6: (1) geração do modelo de arquitetura do domínio a partir de modelos de características e modelos complementares do domínio, (2) geração do projeto de código orientado a características do domínio, (3) geração do *template* do produto e (4) refinamentos manuais. O modelo de análise do produto é utilizado na escolha das características específicas expressas em código a serem combinadas para configuração do sistema alvo.

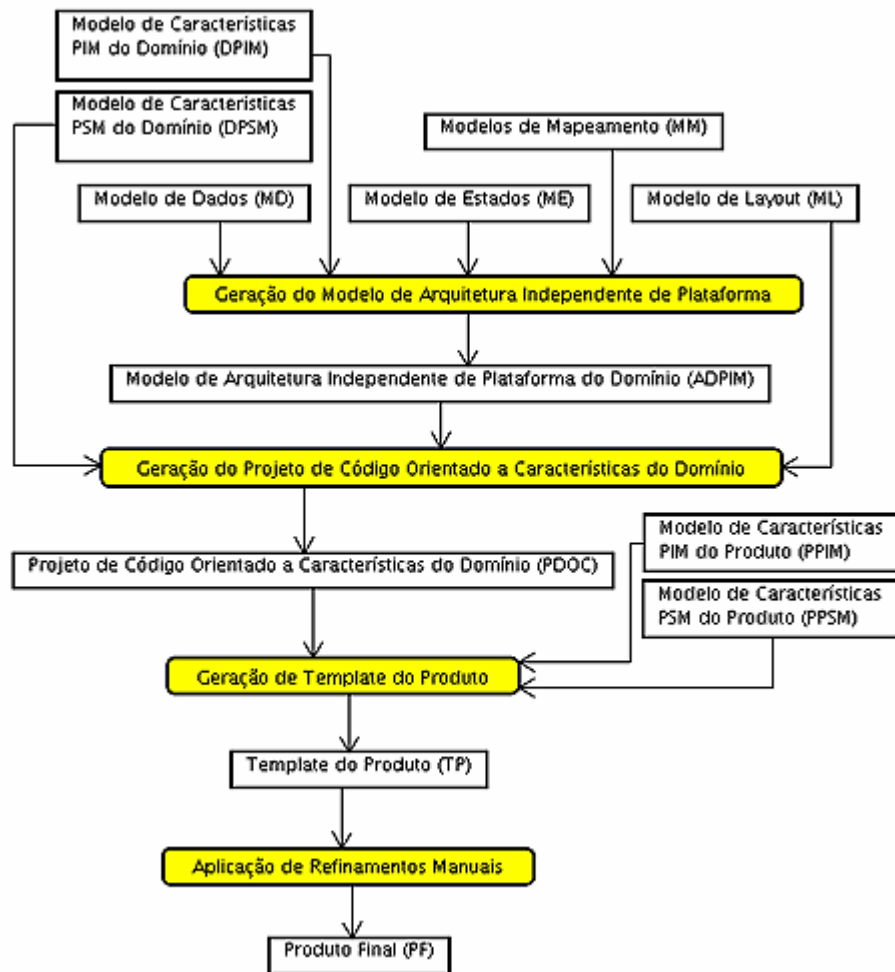


Figura 1.6: Visão geral de nossa abordagem FOMDD - caminho de síntese A

A segunda alternativa de geração de produtos, como na via B da Figura 1.5, faz a combinação de características no nível de modelos, e obtém os artefatos de implementação por meio de transformações exógenas. Nesta opção de geração, modelos de análise do domínio são utilizados juntamente com modelos de análise do produto, para identificação de modelos específicos presentes no domínio a serem combinados em função das características desejáveis no produto. O processo ocorre em seis passos, que podem ser resumidos em três etapas, identificadas pelas atividades presentes na Figura 1.7: (1) geração do modelo de arquitetura do produto, (2) geração do *template* de produto e (3) refinamentos

manuals. Nesta opção de geração a especificação final pode ser feita diretamente para linguagens sem suporte a composição de características.

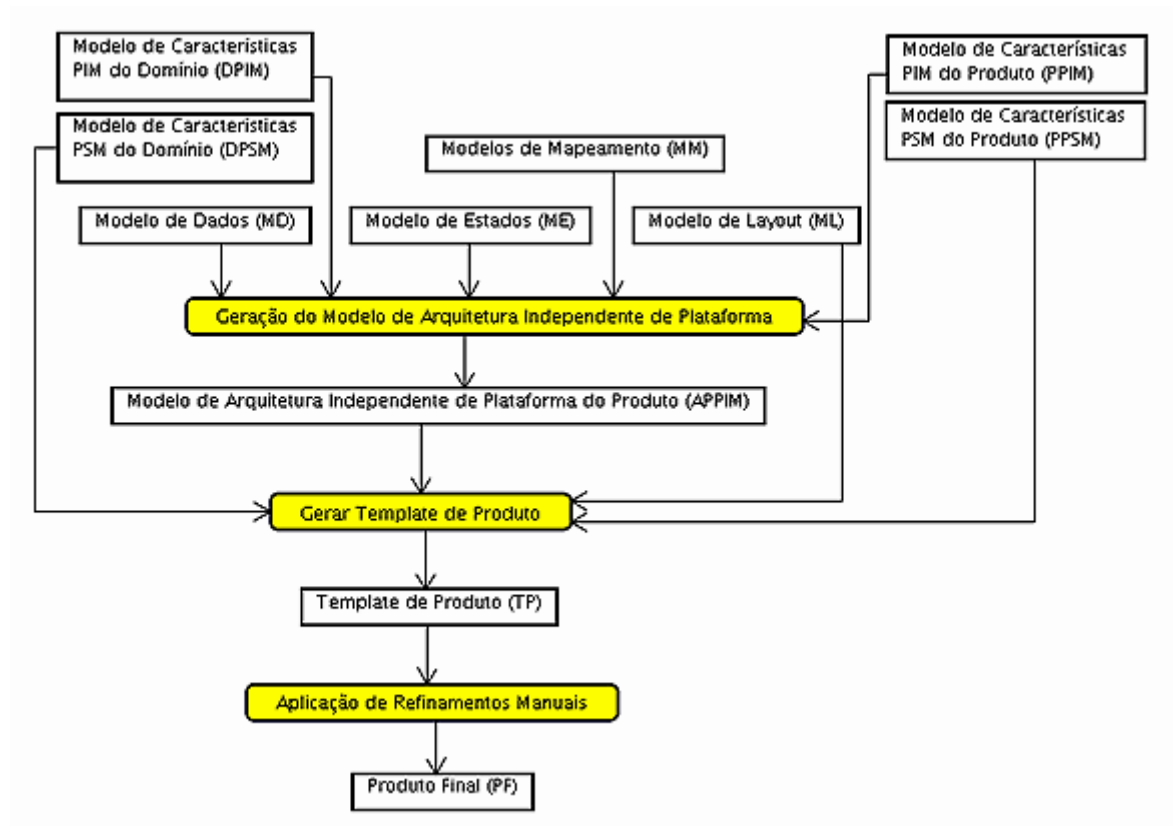


Figura 1.7: Visão geral de nossa abordagem FOMDD - caminho de síntese B

As seções seguintes estão organizadas como segue: o Capítulo 2 apresenta em detalhes a metodologia proposta por nosso trabalho. O Capítulo 3 apresenta uma exemplo de aplicação da metodologia na forma de um estudo de caso, elaborado para a validação de nossa abordagem. O Capítulo 4 apresenta considerações e discussão sobre questões internas da abordagem proposta, bem como trabalhos relacionados. Finalmente, o Capítulo 5 apresenta conclusões e perspectivas futuras para o trabalho.

Capítulo 2

Uma abordagem para desenvolvimento orientado a características e dirigido por modelos

A modelagem de *software* é a atividade de construir modelos que permitam que os aspectos relevantes de sistemas sejam capturados e representados em um nível adequado de abstração. Estes modelos, em geral, são utilizados na identificação das funcionalidades que o *software* deverá prover (análise de requisitos), bem como no planejamento de sua construção. Inicialmente, a modelagem de sistemas teve foco na representação de sistemas individuais, onde uma coleção de modelos tipicamente descreve a estrutura estática, o comportamento dinâmico, interação com usuário, etc. No contexto de linhas de produtos de *software*, a utilização de modelos tem objetivos semelhantes, contudo, estes modelos estão relacionados a um modelo mais geral, chamado Modelo de Características, que identifica o conjunto de características disponíveis em uma linha para a constituição de diferentes produtos. Desta forma, os modelos em linhas de produtos de *software* são compostos por diferentes características. Este tipo de refinamento baseado em características, permite a extensão de um modelo pela adição de elementos, de maneira a permitir a sua customização em função de diferentes necessidades de variabilidade. Assim, ao se trabalhar com uma família de produtos um modelo básico que contém os elementos comuns pode ser reusado [Trujillo et al. 2007].

Um processo para geração orientado a características e dirigido por modelos define um conjunto de modelos orientados a características de entrada, que configura um domínio de aplicação, e que permite a geração de ativos reutilizáveis de *software* através de transformações sucessivas entre modelos. A Figura 1.5 mostra o processo utilizado em [Trujillo et al. 2007].

De forma geral, assim como em [Trujillo et al. 2007], nossa abordagem é uma proposta para construção de linhas de produtos de *software* que define um processo de transforma-

ção por etapas, como em [Trujillo et al. 2007], e que utiliza modelos em três diferentes níveis de abstração: modelos de análise, modelos de projeto e código fonte. Cada um destes níveis é caracterizado por um conjunto de modelos, que encapsula os elementos de representação de programas, descritos em diversos formatos. A transição entre os níveis é caracterizada por um conjunto de transformações que permite a obtenção de novos modelos a partir de modelos de entrada.

A alternativa escolhida para a especificação de características no nível de modelagem neste trabalho, como em [Czarnecki 2005], utiliza modelos não orientados a características em conjunto com modelos orientados a características como entrada para o processo de geração. Como consequência disto, alguns modelos de adicionais foram necessários para expressar o relacionamento entre o modelo de características e os demais modelos de entrada. Nos níveis seguintes, este relacionamento é mantido automaticamente, por meio de transformações MDD.

Nossa abordagem de geração FOMDD, assim como em [Trujillo et al. 2007], permite a geração de produtos por meio de diferentes vias de transformação, que basicamente, são caracterizadas pela ordem em que ocorre a aplicação de transformações horizontais (i.e., a combinação de características). São consideradas duas opções. Na primeira delas, as características são combinadas no nível de modelos, enquanto na segunda as características são combinadas no nível de código.

O processo completo que caracteriza as vias de geração da nossa abordagem é mostrado nas Figuras 2.1 e 2.2. Na primeira das vias de geração o processo é executado em oito etapas, identificadas pelas atividades da Figura 2.1 : (1) simplificação de modelos de análise, (2) geração de *templates* de mapeamento, (3) mapeamento de modelos, (4) geração do modelo de arquitetura do domínio, (5) geração de projeto de código orientado a características do domínio, (6) geração do *template* de produto, (7) transformações finais e (8) refinamentos manuais.

Na segunda das vias o processo ocorre em seis etapas de transformação, identificadas pelas atividades da Figura 2.2: (1) simplificação de modelos de análise, (2) geração de *templates* de mapeamento, (3) mapeamento de modelos, (4) geração do modelo de arquitetura do produto, (5) geração do *template* de produto e (6) refinamentos manuais.

As seções a seguir detalham os modelos e as transformações utilizados no estudo.

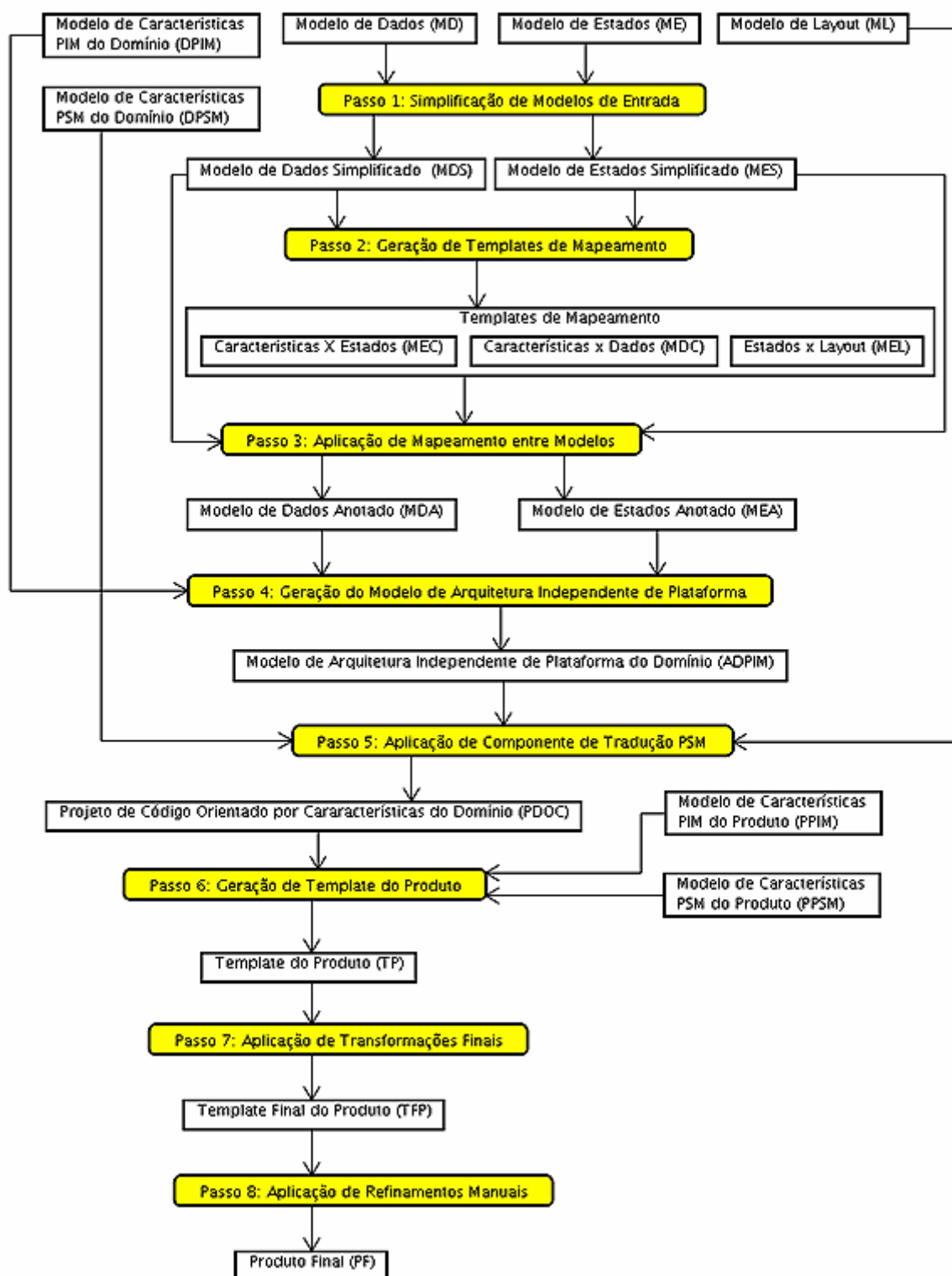


Figura 2.1: Nossa abordagem FOMDD - caminho de Síntese A

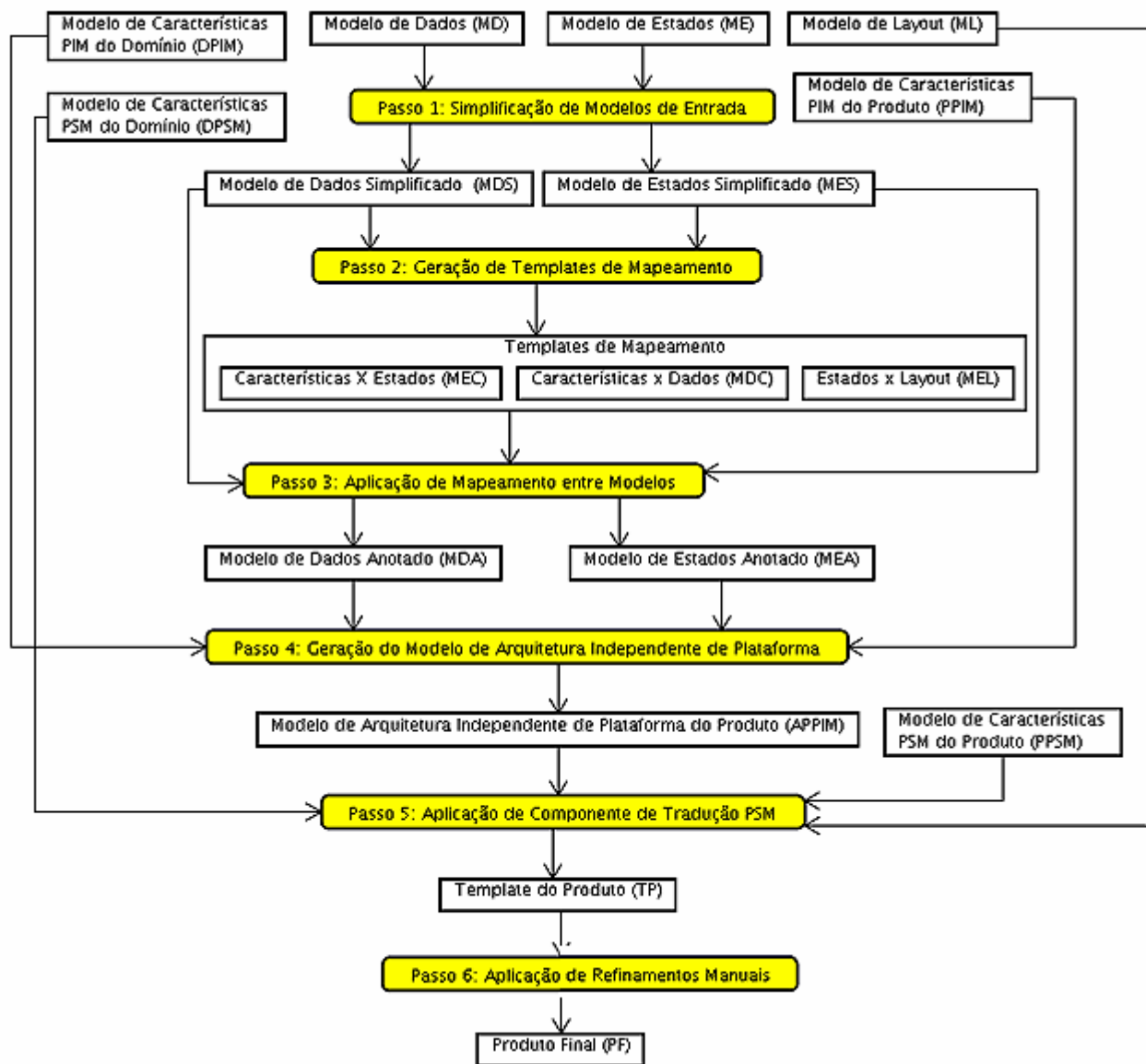


Figura 2.2: Nossa abordagem FOMDD - caminho de Síntese B

2.1 Requisitos para os Metamodelos

Nas seções seguintes mostraremos quais são as propriedades e requisitos que os modelos de análise e projeto devem possuir.

2.1.1 Metamodelos de Análise

Modelos de análise são modelos que refletem estruturas conceituais dos processos de negócio, em vez de implementações reais de *software*. No contexto de linhas de produtos de *software*, estes modelos correspondem à Análise de Domínio, que é o processo sistemático de análise de sistemas relacionados a um mesmo domínio de aplicação. Esta análise tem o objetivo de identificação de partes comuns e partes variáveis. [Czarnecki e Eisenecker 2000]

O conjunto de modelos que caracteriza um modelo de análise pode variar em função do domínio de aplicação em questão, por exemplo, Sistemas Distribuídos, Sistemas Interativos, etc. No caso específico de sistemas interativos, este conjunto deve considerar a interação com o usuário por meio de interfaces gráficas, os dados pertinentes ao domínio de aplicação, bem como as características que constituem o domínio de aplicação e aplicações específicas.

Metamodelo de Características

Um modelo de características é uma representação compacta de todos os produtos de uma linha de produtos de *software*, em termos das funcionalidades que eles apresentam. Modelos de características são utilizados durante todo o ciclo de desenvolvimento de linhas de produtos e são comumente usados como entrada para a produção de outros recursos como documentos, definições de arquitetura ou trechos de código.

Modelos de características foram inicialmente introduzidos pelo método apresentado em [Kang et al. 1990]. Desde então, a modelagem de características tem sido amplamente adotada para a especificação de linhas de produtos de *software*, sendo que uma série de extensões foram propostas à notação original. A notação utilizada neste trabalho considera que apenas as características folhas (i.e., nós terminais da estrutura) encapsulam funcionalidade. Os demais são utilizados apenas como elementos de agrupamento.

Nossa abordagem utiliza quatro diferentes modelos de características: dois modelos de características independentes de plataforma, sendo um para o domínio (DPIM) e outro para o produto (PPIM); e dois modelos de características específicas de plataforma, sendo um para domínio (DPSM) e outro para o produto (PPSM). Os modelos de características do produto, i.e., PPIM e PPSM, contêm a seleção de características para um produto específico escolhido a partir das características do domínio de aplicação presentes em DPIM e DPSM, respectivamente. Os modelos de características independentes de

plataforma (DPIM e PPIM) identificam os módulos funcionais das aplicações enquanto os modelos de características específicos de plataforma (DPSM e PPSM) identificam propriedades tecnológicas do componente de tradução a ser utilizado na fase final de geração automática.

A especificação destes modelos foi feita através de uma notação estendida para diagramas de classes, criada especificamente para diagramas de características, por meio de um perfil¹ UML. Perfis UML proveem um mecanismo de extensão genérico para customização de modelos de domínios particulares, por meio de estereótipos, *tags* e restrições². A Figura 2.3 mostra o metamodelo utilizado para os diagramas de características onde são definidos estereótipos para a extensão da notação de diagramas de classes UML. Estes estereótipos caracterizam os possíveis elementos e relacionamentos presentes em um diagrama de características. Ou seja, eles definem conceitos, características obrigatórias e opcionais, agrupamentos de características (somente uma, uma ou mais, zero ou uma, zero ou várias). O metamodelo da Figura 2.3 define ainda os relacionamentos entre características *excludes* e *requires*, utilizados para expressar características mutuamente exclusivas e dependências entre características, respectivamente. O relacionamento de composição entre características é expresso por associações não estereotipadas.

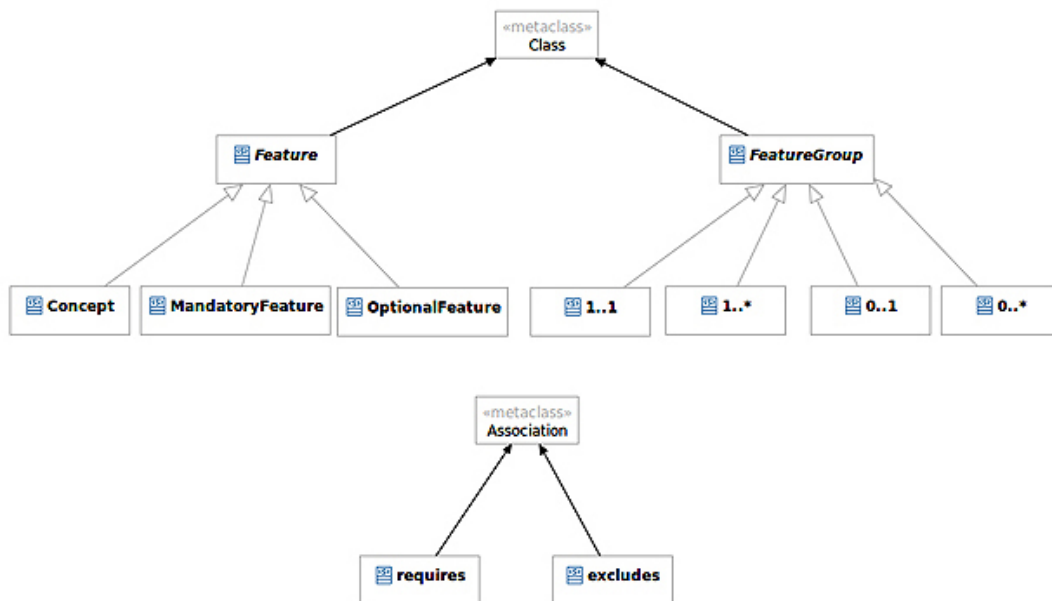


Figura 2.3: Metamodelo para diagramas de características

Metamodelo de Estados

Diagramas de Estados proveem um modelo independente de plataforma para a representação do fluxo de computação em aplicações. Cada aplicação consiste de um conjunto

¹Do inglês, *profile*

²Do inglês, *constraints*

de estados, onde cada estado pode representar uma tela da aplicação. Estados são conectados através de transições cujos elementos para tratamento de eventos (*handlers*) (i) executam alguma ação, (ii) desenham uma tela, ou (iii) ambas as coisas.

A notação para diagramas de estados utilizada em nossa abordagem é uma notação estendida, onde os estados do primeiro nível representam as telas, os estados de segundo nível representam subestados que uma tela pode assumir, e os estados do terceiro nível em diante representam *widgets*, quais serão identificados através de estereótipos e a partir dos quais podem ser disparadas transições. A Figura 2.4 mostra um fragmento do metamodelo utilizado para diagramas de estados. Basicamente, este metamodelo define estereótipos para elementos compostos (como tabelas, formulários, etc) e elementos atômicos (como imagens, rótulos, etc), que podem ser identificados em interfaces gráficas para interação com o usuário.

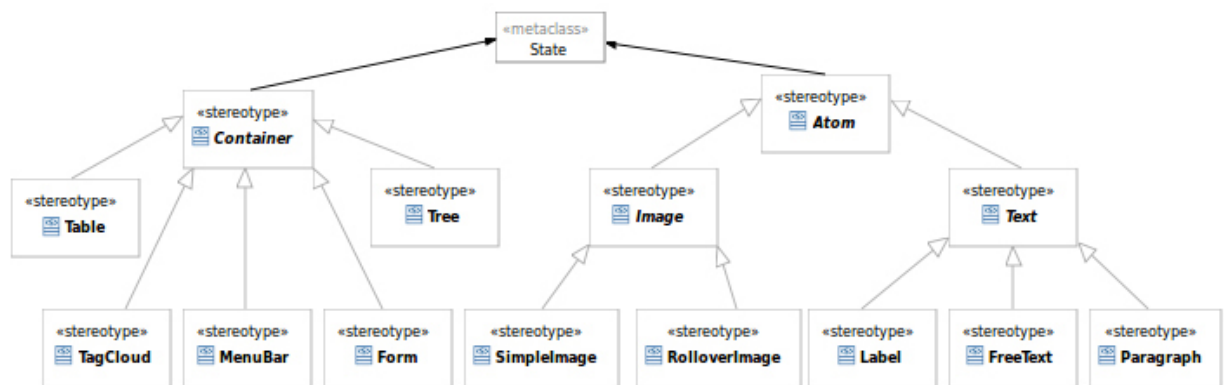


Figura 2.4: Fragmento do metamodelo para diagramas de estados

Metamodelo de Dados

O Modelo de Dados é um modelo abstrato cuja finalidade é descrever, de maneira conceitual, os dados a serem utilizados em um sistema de informação ou mesmo em um domínio específico de sistemas.

A notação utilizada em nossa abordagem é uma extensão de diagramas de classes, expressa por um perfil UML que caracteriza as classes com tabelas e visões. As classes podem conter campos e métodos e podem existir associações entre as classes. Dessa forma, a função do modelo de dados é descrever a representação dos dados persistentes do domínio de aplicação, possibilitando gerar esquemas de bancos de dados. Além disso, o modelo permite a geração das entidades de código para manipulação destes dados. A Figura 2.5 mostra o metamodelo utilizado para diagramas de dados.

Metamodelo de *Layout*

O objetivo do modelo de *layout* é complementar a informação do modelo de estados no que diz respeito a organização visual dos elementos constituintes de cada uma das

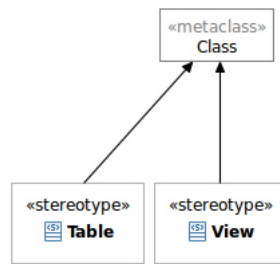


Figura 2.5: Metamodelo para diagramas de dados

telas. Cada uma das telas existentes no domínio de aplicação está associado um *layout* específico, que consiste de uma especificação abstrata para caracterização da estrutura lógica de organização visual. Esta estrutura lógica identifica as posições disponíveis onde os elementos de interação do modelo de estados poderão ser inseridos.

No Capítulo 3, apresentamos uma possível alternativa para o metamodelo de *layout*. Nosso trabalho não utiliza uma formalização deste metamodelo. O objetivo desta não formalização da não fixação de um metamodelo rígido é permitir a utilização de diferentes notações para esta especificação. Um modelo de *layout* deve conter, basicamente, (i) a informação para organização visual dos elementos e (ii) identificadores de áreas disponíveis para alocação dos elementos de interação do modelo de estados.

Metamodelos de Mapeamento

Os modelos de mapeamento são modelos complementares simples, cujo único objetivo é relacionar, de forma declarativa, elementos de diferentes modelos. Estes modelos são constituídos por um conjunto de pares ordenados entre os elementos dos diversos modelos.

Nossa abordagem utiliza três diferentes modelos de mapeamento: entre estados e características, entre dados e características e entre estados e *layout*. Os modelos para mapeamento de características consistem em associações entre os nomes dos elementos constituintes dos modelos (de estados e dados) e os nomes das características de uma linha de produtos. O modelo para mapeamento de elementos de interação e *layout* consiste em associações entre os nomes dos elementos de interação presentes no modelo de estados e os identificadores de áreas do modelo de organização visual.

2.1.2 Metamodelos de Projeto

Modelos de Projeto são modelos relacionados ao desenvolvimento da arquitetura dos sistemas. Segundo [Shaw e Garlan 1996], a arquitetura de *software* envolve a descrição dos elementos a partir dos quais os sistemas são construídos, interações entre estes elementos e padrões (incluindo restrições) que guiam a composição destes elementos. No contexto de linhas de produtos de *software*, os modelos de projeto caracterizam o Projeto de Domínio, cujo objetivo principal é estabelecer uma arquitetura comum para sistemas pertencentes a

um mesmo domínio de aplicação.

Nossa abordagem não formaliza este modelo, já que a representação arquitetural de um mesmo domínio de aplicação pode variar em função de diferentes alternativas para o projeto de código, que encapsulam diferentes padrões de implementação. Além disso, segundo nossa abordagem, a especificação de padrões arquiteturais deve utilizar notações abstratas para permitir a geração de código em diferentes plataformas específicas.

2.2 Transformações

Uma transição entre cada etapa do processo de geração envolve transformações de modelos. Uma transformação no contexto de MDD é definida como o processo de geração de um modelo destino a partir de um ou mais modelos fonte [Kleppe et al. 2003]. Transformações podem ser implementadas através de diversos mecanismos como programas executáveis ou linguagens específicas para definição de transformações como XSLT³, ATL [Jouault et al. 2008], SmartQVT⁴, JET *templates*⁵, etc.

O processo de transformações sucessivas utilizado em nossa abordagem pode ser executado em duas diferentes vias de geração. Inicialmente, descrevemos os passos comuns a estas duas vias (Seção 2.2.1). Em seguida, descrevemos os passos específicos que as caracterizam (Seções 2.2.2 e 2.2.3). Nesta seção, as transformações que caracterizam os passos de transformação da nossa abordagem são descritos de maneira simplificada. O Apêndice A descreve em detalhes as transformações utilizadas no estudo de caso da Seção 3.

2.2.1 Passos de Transformação Iniciais

Esta seção descreve os passos de transformação iniciais, comuns ao processo de geração em ambas as vias de geração.

Passo 1: Simplificação dos Modelos de Dados e Estados

Os modelos de estados e dados utilizados como entrada no processo de geração foram especificados em conformidade com as notações utilizadas pelo *framework* EMF⁶. Grande parte das informações contidas nestes modelos não é de importância fundamental ao processo de geração de código. Dentre estas informações não necessárias, podemos citar: organização visual dos elementos dos diagramas na tela, cores de elementos, dimensões de elementos, inclusão de bibliotecas, etc.

³<http://www.w3.org/TR/xslt>

⁴<http://smartqvt.elibel.tm.fr>

⁵http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html

⁶*Eclipse Modeling Framework* - <http://www.eclipse.org/modeling/emf/>

Este passo consiste em simplificar a notação dos modelos de estados e dados, com objetivo de manter apenas informações relevantes ao processo de geração. Ao final desta fase, obtemos os modelos de estados e dados simplificados. A seguir apresentamos as assinaturas das funções de transformação correspondentes a este passo, onde MD e ME são os modelos de dados e estados, respectivamente e $T_{SimplificarModeloDados}$ e $T_{SimplificarModeloEstados}$ são as funções de simplificação.

$$\begin{aligned} \mathbf{MES} &= T_{SimplificarModeloEstados}(\mathbf{ME}) \\ \mathbf{MDS} &= T_{SimplificarModeloDados}(\mathbf{MD}) \end{aligned}$$

Passo 2: Geração de *Templates* de Mapeamento

A informação de mapeamento entre dados e características é definida como uma relação entre os nomes das características e os nomes dos membros (campos e métodos) das entidades. Da mesma forma, o entrelaçamento entre estados e características é definido como uma relação entre os nomes das características e os nomes dos elementos constituintes deste modelo, ou seja, telas, subestados, *widgets* e transições. É esperado que todas as características se relacionem com um ou mais membros/elementos dos modelos, bem como que cada membro e elemento esteja relacionado com uma ou mais características, embora, elementos possam estar associados exclusivamente à base de construção da linha de produtos. Para satisfazer a esta condição, a anotação de características relativa a estes elementos deve ser deixada em branco.

O modelo de estados não considera a disposição visual dos elementos de interação que caracterizam cada um dos estados. Para a especificação do relacionamento entre elementos de interação e áreas disponíveis no modelo de *layout*, foi utilizado um modelo complementar que relaciona os nomes dos elementos às posições disponíveis, identificadas pelo modelo de *layout*.

O objetivo desta fase é gerar *templates* para facilitar a declaração manual dos pares da relação entre membros/elementos e características, bem como entre elementos de interação e posições no *layout*. Abaixo seguem as assinaturas das funções de geração usando os resultados do Passo 1.

$$\begin{aligned} \mathbf{TMEC} &= T_{GerarTemplateMapeamentoEstadosCaracteristicas}(\mathbf{MES}) \\ \mathbf{TMDC} &= T_{GerarTemplateMapeamentoDadosCaracteristicas}(\mathbf{MDS}) \\ \mathbf{TMEL} &= T_{GerarTemplateMapeamentoEstadosLayout}(\mathbf{MES}) \end{aligned}$$

Estes modelos devem ser preenchidos manualmente por um especialista de domínio, para a especificação das aplicações pertencentes ao domínio de aplicação envolvido no processo. Abaixo seguem as assinaturas das funções que representam estas transformações manuais:

$$\begin{aligned} \mathbf{ACME} &= T_{EspecificarCaracteristicasModeloEstados}(\mathbf{TMEC}) \\ \mathbf{ACMD} &= T_{EspecificarCaracteristicasModeloDados}(\mathbf{TMDC}) \\ \mathbf{ALME} &= T_{EspecificarLayoutModeloEstados}(\mathbf{TMEL}) \end{aligned}$$

Passo 3: Mapeamento entre Modelos

- **Características**

As relações entre DPIM com ME e MD, declaradas no passo anterior, devem ser incorporadas à estrutura dos modelos de dados e estados simplificados. O resultado ao final desta fase são os modelos de estados e dados anotados por características. As assinaturas das funções para este passo intermediário são as que seguem:

$$\begin{aligned} MEAC &= T_{MapearCaracteristicasModeloEstados}(MES, ACME) \\ MDAC &= T_{MapearCaracteristicasModeloDados}(MDS, ACMD) \end{aligned}$$

- **Layout**

As informações de relação entre ME e ML também devem ser incorporadas ao modelo de estados simplificado. O mapeamento de *layout* utiliza o modelo de estados obtido após o mapeamento de características, para complementação da informação de organização visual. A assinatura da função de transformação deste passo complementar de mapeamento é apresentada a seguir:

$$MEACL = T_{MapearLayoutModeloEstados}(MEAC, ALME)$$

2.2.2 Via A: Composição de características no nível de código fonte

Passo A4: Geração de Modelo da Arquitetura Independente de Plataforma do Domínio

A partir dos modelos de estados (incluindo *layout*) e dados anotados por características do domínio, é gerado um modelo de arquitetura independente de plataforma. Para esta geração deve ser escolhida uma transformação que define um estilo arquitetural específico, por exemplo, Model-View-Control, Presentation-Abstraction-Control [Buschmann et al. 1996], dentre outros.

Este modelo contém representações abstratas dos elementos de projeto (entidades do padrão arquitetural) que permitem a geração de artefatos específicos de plataforma e é a última representação independente de plataforma utilizada no processo. A assinatura da função que representa este passo de transformação é apresentada a seguir:

$$AAD = T_{AnaliseParaProjeto}(MEACL, MDAC)$$

Passo A5: Aplicação dos Componentes de Tradução PSM

O objetivo desta fase é transformar a notação abstrata de projeto do domínio em código fonte orientado por características. Para isto, são utilizados como entradas as

representações de arquitetura geradas do Passo A4 e o modelo de características independentes de plataforma do domínio (DPSM), que identifica diferentes componentes para tradução específica de plataforma.

As transformações definidas pelos diferentes componentes de tradução encapsulam *templates* descritos em linguagens específicas de plataforma, para cada tipo de artefato existente na arquitetura, que mapeiam os elementos da representação abstrata de arquitetura (AAD) em elementos de implementação representados através de arquivos nos formatos JAK [Batory et al. 2003] e XAK [Trujillo et al. 2006], disponíveis no ambiente AHEAD. A assinatura de função a seguir define este passo de transformação:

$$ACD = T_{ProjetoParaCodigoCaracteristicas}(AAD, DPSM)$$

Passo A6: Geração do *Template* de Produto

Neste passo de geração, a estrutura de arquivos resultante do Passo A5 (Seção 2.2.2) é combinada através dos mecanismos de programação orientada por características. Basicamente, consiste na invocação de ferramentas específicas de combinação para cada tipo de artefato. Estas ferramentas irão efetuar a junção de todas as ocorrências de um artefato específico, considerando o componente de tradução selecionado pelo modelo de características PPSM e sua estrutura de implementação, seguindo ordem pré-definida pelo modelo de características do produto PPIM.

$$ACP = T_{CombinarCodigoCaracteristicasProduto}(ACD, PPIM, PPSM)$$

Passo A7: Transformações Finais

Após a combinação das características específicas de um produto, pode ser necessário a utilização de transformações extras. No estudo de caso apresentado no Capítulo 3, os arquivos de *script*(Javascript) gerados não são diretamente suportados pelo ambiente de programação orientada por características AHEAD. Dessa forma, foi necessário definir uma notação de representação intermediária XML, para que estes artefatos pudessem ser combinados através da ferramenta XAK. Assim, ao final do processo de combinação, é preciso retornar os artefatos resultantes deste tipo à notação original. A assinatura da função que representa este procedimento é apresentada a seguir:

$$TP = T_{NotacaoIntermediariaParaNotacaoOriginal}(ACP)$$

Passo A8: Definição e Preenchimentos Manuais

Neste passo final, a lógica dos métodos gerados nas representações específicas de plataforma deve ser preenchida manualmente. A assinatura da função que representa o procedimento manual para obtenção do produto final é apresentada a seguir:

$$PF = T_{PreencherManualmenteProcedimentos}(TP)$$

2.2.3 Via B: Composição de características no nível de modelos

Passo B4: Geração de Modelo da Arquitetura Independente de Plataforma do Produto

Os modelos obtidos ao final do Passo 3 (2.2.1) contém a descrição completa do domínio de aplicação. Para a configuração do produto no nível de modelagem, é necessária a seleção das entidades dos modelos de análise que correspondem às características do produto. Este passo de transformação é executado de forma semelhante a descrita na Seção 2.2.2, porém, utilizando o modelo parcial, que contém apenas características de um produto específico. A assinatura da função que representa este passo é mostrada a seguir:

$$AAP = T_{AnaliseParaProjeto}(MEACL, MDAC, PPIM)$$

Passo B5: Aplicação do Componente de Tradução PSM

Funciona de forma similar ao Passo A5 (Seção 2.2.2), no entanto, utiliza como entrada as representações de arquitetura geradas no Passo B4 (Seção 2.2.3) e o modelo de características independentes de plataforma do produto (PPSM). PPSM, identifica um componente específico de tradução. O objetivo desta fase é transformar a notação abstrata em código fonte. Neste caso, não necessariamente orientado a características, já que a combinação de características é feita anteriormente, no nível de modelagem. Como o código resultante da aplicação do componente de tradução já expressa a combinação de características do produto, os artefatos de código gerados (ACP) equivalem ao *template* de produto (TP). A assinatura da função que representa as transformações deste passo é mostrada a seguir:

$$TP = T_{ProjetoParaCodigo}(AAP, PPSM)$$

Passo B6: Definição e Preenchimentos Manuais

Exatamente como no Passo A8 (Seção 2.2.2), atributos e conteúdo de *tags*, além da lógica dos métodos gerados nas representações específicas de plataforma devem ser preenchidos manualmente. A seguir é mostrada a assinatura da função que representa o procedimento:

$$PF = T_{PreencherManualmenteProcedimentos}(TP)$$

2.3 Considerações Sobre o Processo de Geração

2.3.1 Geração de Linhas de Produtos

O processo de geração dos artefatos de código de uma linha de produtos é caracterizado como um metaprograma que utiliza como parâmetros de entrada, o modelo de análise do

domínio de aplicação, o modelo abstrato da arquitetura alvo, bem como geradores de código, que permitem a obtenção de artefatos por meio de transformações específicas de plataforma, a partir do modelo de arquitetura.

Os formatos da especificação de código para a linha de produtos são encapsulados pelo modelo de análise que contém um modelo de características específicas de plataforma. Este modelo identifica componentes de tradução para código, a serem utilizados na fase final de geração dos artefatos.

A assinatura de função a seguir descreve o processo, onde MAD representa o modelo de análise do domínio e MPD o modelo de projeto.

$$ACD = T_{GerarArtefatosCodigoDominio}(MAD, MPD)$$

2.3.2 Geração de Produtos

Assim como o processo de geração dos artefatos de domínio, a geração dos artefatos de produto é caracterizada como um metaprograma que toma como entrada um conjunto de modelos. No caso específico da geração de produtos, foram analisados dois possíveis caminhos de síntese (caminhos A e B da Figura 1.5). O caminho A faz a geração do produto através de transformações endógenas, aplicadas aos artefatos de código do domínio, em função das características do produto. Os modelos de características do produto (PPIM e PPSM), caracterizam o modelo de análise do produto e selecionam as propriedades específicas do produto alvo. A assinatura de função a seguir resume o processo de geração de produtos por meio de combinações no nível de código, entre as características específicas do produto (modelo de análise MAP).

$$ACP = T_{GerarArtefatosCodigoProdutoA}(ACD, MAP)$$

O segundo caminho de síntese analisado, o caminho B, obtém os artefatos do produto pela seleção orientada a características dos artefatos abstratos de construção (especificados pelos modelos de projeto), e posterior tradução em código fonte. A tradução neste caso não precisa necessariamente ser feita para linguagens que possuam suporte para combinação horizontal de artefatos no nível de código. Ou seja, os elementos de projeto selecionados que configuram a combinação de características podem ser traduzidos por transformações exógenas diretamente para linguagens de programação convencionais, como Java ou PHP.

$$ACP = T_{GerarArtefatosCodigoProdutoB}(MAD, MAP)$$

2.4 Resumo do Capítulo

Neste capítulo foram apresentados os princípios de nossa abordagem, incluindo as propriedades e requisitos dos modelos de análise e projeto utilizados, bem como das transformações que caracterizam o processo de geração de produtos em duas diferentes vias de transformação.

O conjunto de modelos utilizados em ambas as vias é o mesmo. No entanto, eles são utilizados de forma diferente. Basicamente, a diferença entre as duas vias de composição é a ordem em que transformações endógenas são aplicadas aos artefatos que representam as características, i.e., no nível de modelagem ou no nível de código fonte. O resultado final obtido ao final de ambas as vias é o mesmo.

Na primeira das vias (Via A), onde a composição de características é feita no nível de código fonte, o produto final é obtido através de oito passos de transformação. De forma geral, estes passos obtêm os artefatos de implementação do domínio de aplicação por transformações exógenas de modelos a partir dos modelos de análise. Estes artefatos são então combinados por transformações endógenas, em função dos modelos de características do produto, produzindo *templates* de código para a implementação do produto final. Na segunda via de transformação (Via B), onde a combinação de características é feita no nível de modelagem, o mesmo resultado é atingido após seis passos de transformação. De forma geral, estes passos fazem uma seleção de elementos dos modelos de estados e dados do domínio, orientada pelos modelos de características dos produtos. Os modelos e respectivos elementos selecionados são convertidos em modelos de representação arquitetural do produto, que posteriormente são transformados em código, produzindo *templates* de código para a implementação do produto final. Em ambas as vias, a caracterização do produto final é feita por um passo de transformação manual final, para a especificação de métodos/funções, entre outros elementos, como atributos e conteúdo de *tags*.

Capítulo 3

Estudo de Caso: Linhas de Produtos para Lojas Virtuais

O estudo de caso elaborado para a avaliação da abordagem foca na geração de sistemas interativos para a *web*, mais especificamente Lojas Virtuais implementadas utilizando as plataformas JEE (*Java Enterprise Edition*) e PHP (*Hypertext Preprocessor*), segundo padrão arquitetural MVC (Model-View-Controller). A criação de domínios de aplicação e aplicações específicas pertencentes a este domínio, fundamentada no arcabouço de transformação apresentado no Capítulo 2, consiste em: (i) definição dos modelos de características para domínio e produto, (ii) definição de modelos de estados e dados do domínio, (iii) definição do modelo de *layout* do domínio, (iv) definição dos modelos complementares de mapeamento e (v) aplicação dos procedimentos de transformação que caracterizam o processo.

3.1 Tecnologia Utilizada

Para a caracterização dos modelos necessários ao estudo de caso, foram utilizados recursos de modelagem do *framework* EMF, além de linguagens de domínio específico, expressas em XML, definidas especialmente para a especificação dos modelos nos formatos adequados aos passos de transformação, como, por exemplo, a descrição das entidades do padrão arquitetural MVC. Para a especificação dos modelos de características e dados, foram desenvolvidas extensões específicas para diagramas de classes UML, definidas por meio de perfis UML, utilizando a IDE Eclipse. Da mesma forma, para especificação do modelo de estados utilizado, foi desenvolvida uma extensão (perfil UML) de diagramas de estados UML, que contempla telas, subestados, *widgets* e transições.

As transformações que caracterizam a transição entre os passos de geração anteriores a geração de código, onde são feitas conversões entre distintos formatos XML, foram implementadas através de XSLT (transformações mais simples) e programas Java (trans-

formações mais complexas), que utilizam as APIs SAX¹ e DOM², para manipulação de arquivos XML. Embora XSLT tenha sido projetada para uso como parte da XSL³, que é uma linguagem de estilo para XML, neste trabalho ela foi utilizada independentemente de XSL.

Para a especificação das transformações finais, que caracterizam a obtenção de código fonte do *template* de produto, foram utilizados *templates* JET. *Templates* JET é um mecanismo do *framework* EMF, que permite a especificação de modelos de código (e outros como XML e SQL), para a geração automatizada com base nestes modelos e parâmetros de entrada. Na primeira das vias de geração (via A), cada uma das classes de artefatos a serem gerados possui um *template* para a caracterização da base e outro para a caracterização de refinamentos. Na segunda via (via B), cada classe de artefato possui um *template*.

3.2 Modelos Utilizados

3.2.1 Modelos de Análise

Nossa abordagem é fundamentada no estudo de sistemas caracterizados pela interação com o usuário através de interfaces gráficas. Para a consolidação do estudo de avaliação foram utilizados os seguintes modelos abstratos de análise como entrada: Modelo de Dados do Domínio (MD), Modelo de Estados do Domínio (ME), Modelos de Características Independentes de Plataforma para o Domínio (DPIM⁴) e para o Produto (PPIM⁵), Modelos de Características Específicas de Plataforma para Domínio (DPSM⁶) e para o Produto (PPSM⁷) e Modelo de *Layout* do Domínio (ML). Foram utilizados ainda modelos complementares para especificação do mapeamento entre elementos dos modelos de estados e características (MEC), dados e características (MDC) e entre estados e *layout* (MEL).

Modelos de Características

Os modelos de características utilizados no processo de geração do estudo de caso descrevem:

- As características funcionais que definem a linha de produtos de Lojas Virtuais (DPIM).

¹<http://www.saxproject.org/>

²<http://www.w3.org/DOM/>

³<http://www.w3.org/Style/XSL/>

⁴Do inglês, *Domain's Platform Independent Model*

⁵Do inglês, *Product's Platform Independent Model*

⁶Do inglês, *Domain's Platform Specific Model*

⁷Do inglês, *Product's Platform Specific Model*

- As características que identificam alternativas para componentes de tradução para construção dos artefatos de código da linha de produtos Lojas Virtuais (DPSM).
- As características funcionais de um produto específico (PPIM).
- As características que identificam o componente de tradução específico utilizado para obtenção do produto específico (PPSM).

Como apresentado na Seção 2.1, nossa abordagem utilizou um modelo UML estendido para a construção de diagramas de características. A notação utilizada contempla conceitos, características obrigatórias e opcionais, agrupamentos de características com cardinalidades $\langle 0..1 \rangle$ (nenhuma ou somente uma), $\langle 0..* \rangle$ (nenhuma ou várias), $\langle 1..1 \rangle$ (somente uma) e $\langle 1..* \rangle$ (uma ou várias). A notação inclui também associações que identificam relações de dependência (estereótipo *requires*), exclusão (estereótipo *excludes*) e sub-tipagem (associação não estereotipada).

As Figuras 3.1 e 3.2 mostram fragmentos do modelo de características independentes de plataforma do domínio Lojas Virtuais, onde as funcionalidades e elementos de agrupamento funcional e associações do domínio de aplicação são identificados. Os nós terminais deste diagrama identificam o conjunto de características funcionais disponíveis no domínio de aplicação, a partir das quais será feita a caracterização de produtos individuais.

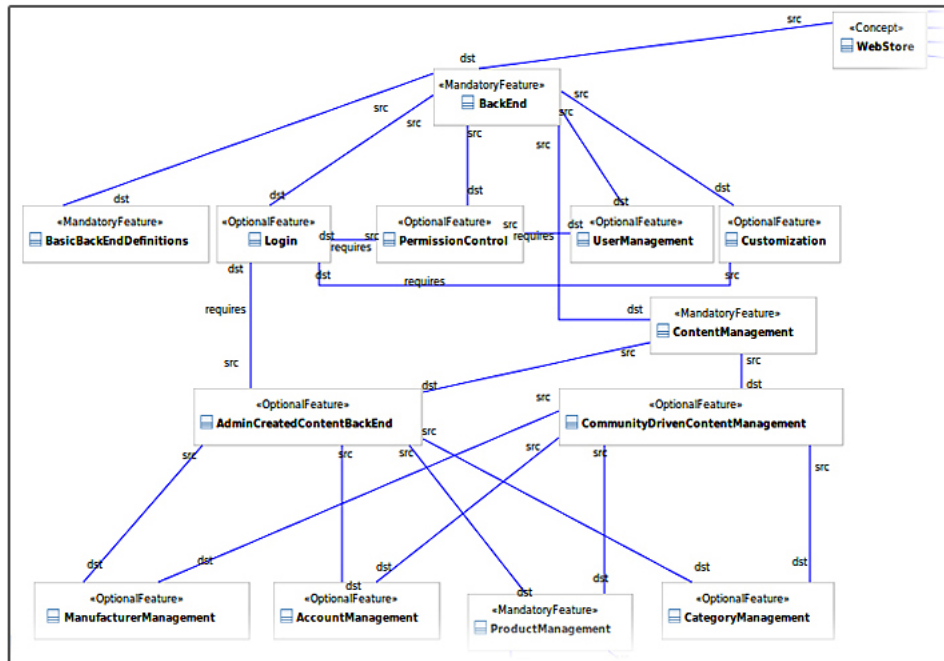


Figura 3.1: Fragmento 1 do modelo de características PIM do domínio *Lojas Virtuais*

A Figura 3.3 mostra o modelo de características específicas de plataforma do domínio de aplicação Lojas Virtuais. As características deste modelo identificam as propriedades que os componentes de tradução em código do domínio de aplicação devem possuir, como mecanismo de persistência e plataforma operacional.

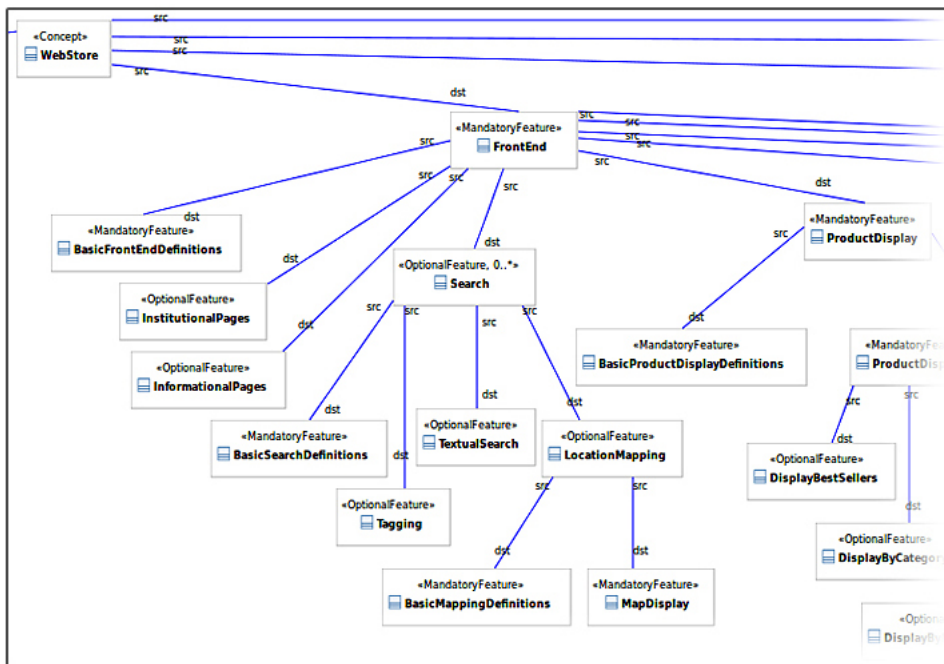


Figura 3.2: Fragmento 2 do modelo de características PIM do domínio *Lojas Virtuais*

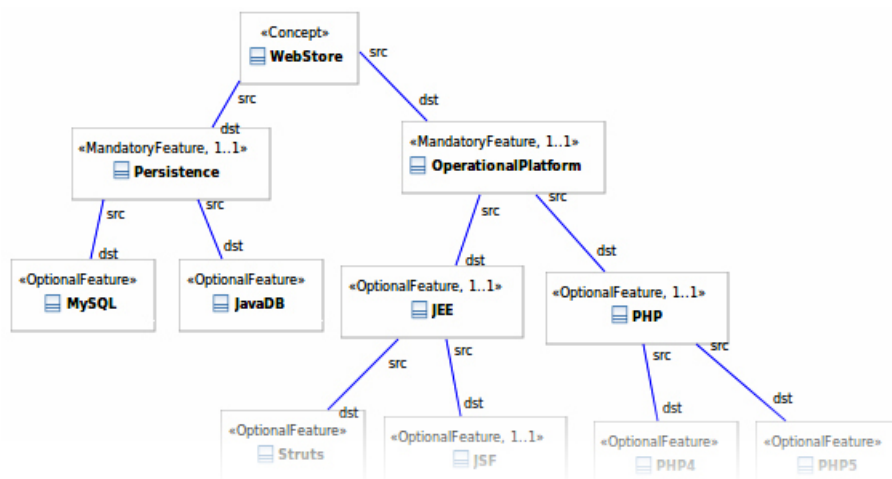


Figura 3.3: Modelo de características PSM do domínio *Lojas Virtuais*

As Figuras 3.4 e 3.5 mostra um fragmento do modelo de características independentes de plataforma do produto. Este modelo contém uma seleção específica de características funcionais do domínio de aplicação, selecionadas para o produto alvo.

A Figura 3.6 mostra o modelo de características específicas de plataforma do produto, que identifica as propriedades de um componente de tradução em código específico a ser utilizado para a geração do produto alvo.

Modelo de Estados

O modelo de estados utilizado no estudo de caso é caracterizado por elementos que identificam telas (estados), configurações distintas de uma mesma tela (subestados) e

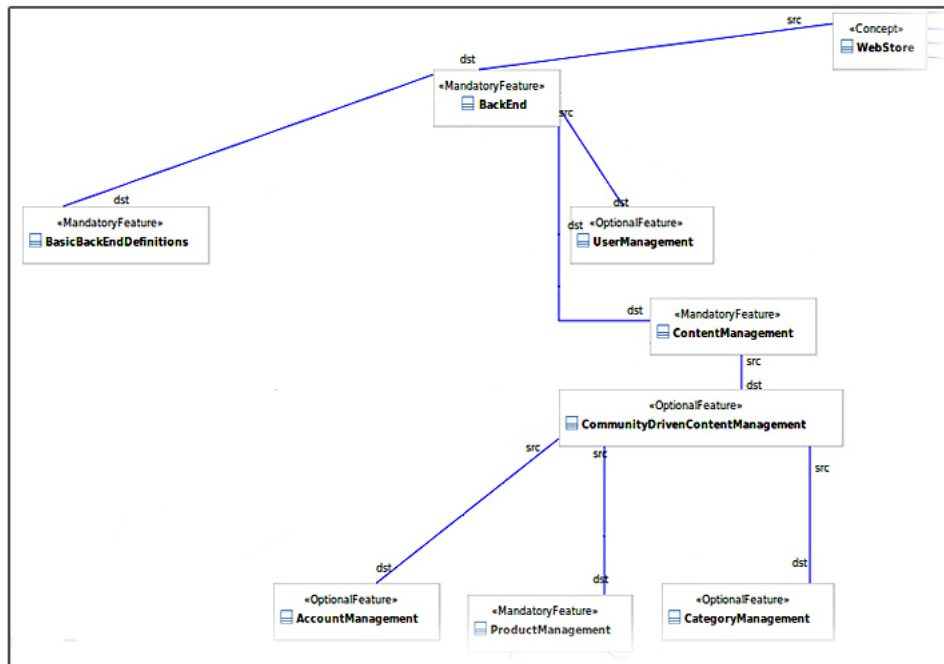


Figura 3.4: Fragmento 1 do modelo de características PIM de produto

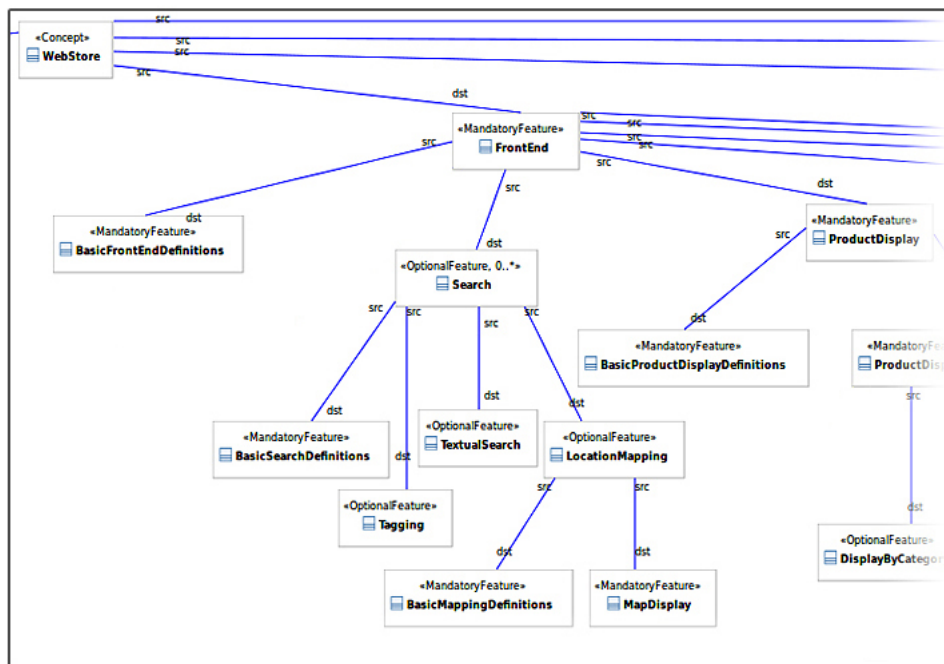


Figura 3.5: Fragmento 2 do modelo de características PIM de produto

elementos de interação que as compõem (*widgets*), bem como transições que caracterizam a interação por meio dos elementos. Estes elementos são caracterizados por estereótipos definidos no metamodelo de estados da Seção 2.1.1. A Figura 3.7 mostra um fragmento do modelo de estados para Lojas Virtuais, que contém a representação da tela mostrada na Figura 3.8. Estas figuras apresentam exemplos (identificados pelos rótulos A e B) mostrando a associação entre os elementos reais de implementação (Figura 3.8), presentes

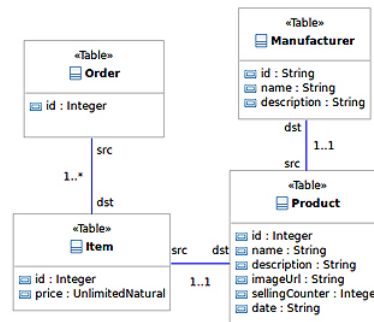


Figura 3.9: Fragmento do modelo de dados do domínio de aplicação *Lojas Virtuais*

Modelo de Dados

O modelo de dados utilizado no estudo de caso contém a representação dos dados persistentes relativos ao domínio de aplicação Lojas Virtuais. A Figura 3.9 mostra um fragmento deste modelo onde são identificadas as entidades de dados do domínio de aplicação (tabelas e visões constituídas por atributos e operações) e associações entre entidades. Estes elementos serão utilizados para a criação da base de dados, bem como de entidades de código para manipulação destes dados.

Modelo de *Layout*

A especificação de modelos de *layout* no estudo de caso é feita através da linguagem HTML. Apesar desta notação ser específica de plataforma, considerando que a *web* caracteriza uma plataforma, para efeito de estudo, ela foi considerada como independente de plataforma, por se tratar de uma representação genérica o suficiente para ser traduzida para diferentes notações e não causar impacto nos resultados deste trabalho. Outra justificativa é que por razões práticas, HTML é uma linguagem cuja manipulação é relativamente simples. A associação entre *widgets* do modelo de estados e as áreas disponíveis no modelo de *layout* é feita através dos identificadores de elementos presentes no modelo de *layout*. No momento da geração de código, as áreas cujos identificadores estiverem associados a elementos do modelo de estados terão seu conteúdo preenchido pelo código que caracteriza tais elementos. A Figura 3.10 mostra um exemplo, caracterizado por elementos de organização (ex.: *head*, *body*, *table*, etc) e áreas identificadas (ex.: *all*, *header*, *column1*, *footer*, etc), a serem preenchidas por elementos de interação.

Modelos de Mapeamento

A seguir, apresentaremos os três diferentes modelos de mapeamento utilizados no estudo:

- Modelo de Mapeamento de Estados x Características (MEC)
- Modelo de Mapeamento de Dados x Características (MDC)

```

<html>
<head>
</head>
<body>
<div id="all">

<div id="header">
<table>
<tr>
<td id="header_top_1"></td>
<td id="header_top_2"></td>
<td id="header_top_3"></td>
</tr>
<tr>
<td colspan="3" id="header_bottom"></td>
</tr>
</table>
</div>

<div id="main">
<div id="column1"></div>
</div>

<div id="footer"></div>

</div>
</body>
</html>

```

Figura 3.10: Exemplo de especificação de *layout*

- Modelo de Mapeamento de Estados x *Layout* (MEL)

A Figura 3.11 mostra um fragmento do *template* de mapeamento entre os modelos de estado e características. Este modelo é composto por pares ordenados que associam os nomes dos elementos (i.e., transições (linha 2), telas (linha 3), subestados (linha 4), elementos de interação (*widgets*) (linhas 5 a 19)) do modelo de estados com as características funcionais do domínio de aplicação, a que estes elementos se referem. Nas linhas 15 e 17, não foram descritas as associações entre os elementos e alguma característica. Neste caso, estes elementos estão associados à base do sistema.

```

1 <mapping>
2 <transition feature="Navigation" name="onClick" source="Home.Default.MainMenu.Item4.MapLink" target="Map.Default"/>
3 <screen feature="Navigation" name="Home"/>
4 <state feature="Navigation" name="Home.Default"/>
5 <element feature="Advertising" name="Home.Default.AdvertisePanel"/>
6 <element feature="Advertising" name="Home.Default.AdvertisePanel.Item1"/>
7 <element feature="Advertising" name="Home.Default.AdvertisePanel.Item1.JavaEEBanner"/>
8 <element feature="Advertising" name="Home.Default.AdvertisePanel.Item2"/>
9 <element feature="Advertising" name="Home.Default.AdvertisePanel.Item2.JavaSEBanner"/>
10 <element feature="Advertising" name="Home.Default.AdvertisePanel.Item3"/>
11 <element feature="Advertising" name="Home.Default.AdvertisePanel.Item3.NetBeansBanner"/>
12 <element feature="DisplayByCategory" name="Home.Default.FishEyeMenu"/>
13 <element feature="DisplayByCategory" name="Home.Default.FishEyeMenu.Categories"/>
14 <element feature="DisplayByCategory" name="Home.Default.FishEyeMenu.Categories.CategoryLink"/>
15 <element feature="" name="Home.Default.Footer"/>
16 <element feature="Navigation" name="Home.Default.HeaderLink"/>
17 <element feature="" name="Home.Default.HeaderTitle"/>
18 <element feature="DisplayByCategory" name="Home.Default.ImageMapMenu"/>
19 <element feature="DisplayByCategory" name="Home.Default.ImageMapMenu.Area1"/>
<element feature="DisplayByCategory" name="Home.Default.ImageMapMenu.Area2"/>

```

Figura 3.11: Fragmento de *template* de mapeamento estados x características

A Figura 3.12, mostra um fragmento do *template* de mapeamento entre os modelos de dados e características. Este modelo é composto por pares ordenados que associam os nomes dos elementos (i.e., tabelas, visões, atributos e operações) do modelo de dados e os nomes das características funcionais do domínio de aplicação.

```

<mapping>
  <table feature="UserManagement" name="User"/>
  <attribute feature="UserManagement" name="userId"/>
  <attribute feature="UserManagement" name="userFirstName"/>
  <attribute feature="UserManagement" name="userLastName"/>
  <attribute feature="Login" name="userLoginName"/>
  <attribute feature="Login" name="userPassword"/>
  <attribute feature="Login" name="userPrivilegeLevel"/>
  <attribute feature="UserManagement" name="userEmail"/>
  <operation feature="UserManagement" name="userAccess"/>
  <view feature="BasicProductDisplayDefinitions" name="Catalog"/>
  <attribute feature="BasicProductDisplayDefinitions" name="catalogLastPage"/>
  <attribute feature="BasicProductDisplayDefinitions" name="catalogProperty"/>
  <attribute feature="BasicProductDisplayDefinitions" name="catalogEmf"/>
  <operation feature="DisplayByCategory" name="catalogGetCategories"/>

```

Figura 3.12: Fragmento de *template* de mapeamento dados x características

Estes *templates* deverão ser preenchidos manualmente, completando a especificação de características necessária ao processo de geração. A associação da informação contida nestes modelos aos modelos de estados e dados originais permitem a realização de novos modelos anotados por características que serão utilizados como base para os processo de geração.

O modelo para mapeamento de estados e *layout*, gerado a partir do modelo de estados original, associa os elementos das telas (i.e., *widgets*) e as posições disponíveis na estrutura de organização visual. A Figura 3.13 mostra um fragmento do *template* utilizado para caracterização desta relação. A cada uma das telas do domínio de aplicação está associado um *layout* específico (atributo *layout*) e a cada elemento de interação contido nas telas, está associada uma posição disponível no *layout* da tela correspondente ao elemento.

```

<mapping>
  <screen layout="4Columns" name="Home"/>
  <screen layout="1Column" name="Tags"/>
  <screen layout="2Columns" name="Map"/>
  <screen layout="3ColumnsA" name="Catalog"/>
  <screen layout="1ColumnA" name="Search"/>
  <screen layout="1Column" name="Seller"/>
  <screen layout="1Column" name="News"/>
  <element at="column1" name="Catalog.Default.AccordionMenu"/>
  <element at="Catalog.Default.AccordionMenu" name="Catalog.Default.AccordionMenu.Item1"/>
  <element at="Catalog.Default.AccordionMenu" name="Catalog.Default.AccordionMenu.Item1"/>
  <element at="Catalog.Default.AccordionMenu.Item1" name="Catalog.Default.AccordionMenu.Item1.MenuTitle"/>
  <element at="Catalog.Default.AccordionMenu.Item1" name="Catalog.Default.AccordionMenu.Item1.State"/>
  <element at="Catalog.Default.AccordionMenu.Item1.State" name="Catalog.Default.AccordionMenu.Item1.State.Item1"/>
  <element at="Catalog.Default.AccordionMenu.Item1.State.Item1" name="Catalog.Default.AccordionMenu.Item1.State.Item1.State.Item1"/>
  <element at="Catalog.Default.AccordionMenu.Item1.State" name="Catalog.Default.AccordionMenu.Item1.State.SubCategories"/>
  <element at="Catalog.Default.AccordionMenu.Item1.State.SubCategories" name="Catalog.Default.AccordionMenu.Item1.State.SubCategories"/>
  <element at="column3" name="Catalog.Default.AdvertisePanel1"/>
  <element at="Catalog.Default.AdvertisePanel1" name="Catalog.Default.AdvertisePanel1.Item1"/>

```

Figura 3.13: Fragmento de *template* de mapeamento estados x *layout*

3.2.2 Modelos de Projeto

O estudo de caso utilizado para a avaliação de nossa abordagem baseia-se na geração de famílias de sistemas que se enquadram na categoria de Sistemas Interativos. Em geral, os sistemas pertencentes a esta categoria são formalizados através do uso de um

dos seguintes padrões arquiteturais: Modelo-Visão-Controle (MVC)⁸ ou Apresentação-Abstração-Controle⁹ [Buschmann et al. 1996].

O padrão MVC, alternativa utilizada no estudo, divide uma aplicação interativa em três componentes: modelo, visão e controlador. O Modelo contém o núcleo funcional e os dados, a Visão exibe informação ao usuário e o Controlador lida com entradas de dados. Visões e controladores em conjunto compõem a interface com o usuário e um mecanismo de propagação de mudanças assegura a consistência entre a interface e o modelo.

A notação utilizada em nossa abordagem segue os princípios do modelo arquitetural MVC, porém de forma abstrata e independente de plataforma com objetivo de permitir que a representação arquitetural final seja obtida por meio de transformações específicas em diferentes plataformas tecnológicas. Além de descreverem as entidades do padrão arquitetural escolhido, os modelos de representação arquitetural preservam as propriedades de mapeamento entre modelos, especificadas anteriormente no processo de geração.

A Figura 3.14 mostra um fragmento da especificação arquitetural das entidades Modelo, que encapsulam campos e operações para a manipulação de dados persistentes relativos ao do domínio de aplicação. Estes elementos são obtidos a partir do modelo de dados, através de uma transformação simples, onde tabelas são convertidas em modelos (MVC) e visões são convertidas em fachadas. Os nomes das entidades obtidas são os mesmos das entidades (i.e., tabelas e visões) a partir das quais elas são obtidas. Os atributos e métodos presentes nas entidades do modelo de dados também são preservados neste modelo de projeto.

```
<models>
  <model feature="UserManagement" name="User">
    <attribute feature="UserManagement" name="userId" type="Integer"/>
    <attribute feature="UserManagement" name="userFirstName" type="String"/>
    <attribute feature="Login" name="userLoginName" type="String"/>
    <attribute feature="Login" name="userPassword" type="String"/>
    <attribute feature="Login" name="userPrivilegeLevel" type="String"/>
    <attribute feature="UserManagement" name="userEmail" type="String"/>
    <operation feature="UserManagement" name="userAccess" type="void">
      <parameters>
        <parameter name="t" type="Integer"/>
      </parameters>
    </operation>
    <attribute feature="" name="userAddress" type="Address"/>
  </model>
  <model feature="" name="Order">
    <attribute feature="BasicProductManagementDefinitions" name="orderId" type="Integer"/>
    <attribute feature="BasicProductManagementDefinitions" name="orderItem" type="collection(Item)"/>
  </model>
</models>
```

Figura 3.14: Fragmento de representação arquitetural abstrata MVC - *Modelo*

A Figura 3.15 mostra um fragmento da especificação arquitetural das entidades Visão, que encapsulam representações dos elementos visuais que caracterizam cada uma das telas do domínio de aplicação. Os estados do primeiro nível do modelo de estados são convertidos em telas. Os estados de segundo nível equivalem a configurações específicas de *widgets* nas telas e são considerados apenas na entidade Controlador, através da es-

⁸Do inglês, *Model-View-Controller*

⁹Do inglês, *Presentation-Abstraction-Control*

pecificação de um procedimento que deve exibir os *widgets* específicos de um subestado. Os estados de terceiro nível em diante, elementos estereotipados, equivalem aos *widgets* presentes nas telas.

```
<views>
<screen feature="Navigation" layout="4Columns" name="Home">
  <element at="header_top_1" depth="0" feature="" name="Home.Default.HeaderLink" type="TextualLink">
    <transition feature="Navigation" name="onClick" source="Home.Default.HeaderLink" target="Home.Default"/>
  </element>
  <element at="header_bottom" depth="0" feature="RSSNews" name="Home.Default.RSSChannel" type="Table">
    <element at="Home.Default.RSSChannel" depth="1" feature="RSSNews" name="Home.Default.RSSChannel.Row1" type="TableRow">
      <element at="Home.Default.RSSChannel.Row1" depth="2" feature="RSSNews" name="Home.Default.RSSChannel.Row1.Column1" type="TextualLink">
        <element at="Home.Default.RSSChannel.Row1.Column1" depth="3" feature="" name="Home.Default.RSSChannel.Row1.Column1.ChannelLink" type="TextualLink">
          <transition feature="RSSNews" name="onClick" source="Home.Default.RSSChannel.Row1.Column1.ChannelLink" target="Home.Default.RSSChannel.Row1.Column1.ChannelLink"/>
        </element>
      </element>
      <element at="Home.Default.RSSChannel.Row1" depth="2" feature="RSSNews" name="Home.Default.RSSChannel.Row1.Column2" type="TextualLink">
        <element at="Home.Default.RSSChannel.Row1.Column2" depth="3" feature="RSSNews" name="Home.Default.RSSChannel.Row1.Column2.RSSItems" type="TextualLink">
          <transition feature="" name="onClick/accessFeed($url)" source="Home.Default.RSSChannel.Row1.Column2.RSSItems" target="Home.Default.RSSChannel.Row1.Column2.RSSItems"/>
        </element>
      </element>
    </element>
  </element>
</screen>
</views>
```

Figura 3.15: Fragmento de representação arquitetural abstrata MVC - *Visão*

A Figura 3.16 mostra um fragmento da especificação arquitetural das entidades Controlador, que encapsulam representações dos elementos de processamento e resposta a eventos de cada uma das telas da aplicação, originados a partir da ação do usuário na interface do sistema, e que são representados pelas transições presentes no modelo de estados. Além disso, o controlador de uma tela contém os métodos necessários para a caracterização dos subestados desta tela.

```
<controllers>
<screen feature="Navigation" name="Home">
  <handler feature="Navigation" name="onClick" source="Home.Default.HeaderLink" target="Home.Default"/>
  <handler feature="RSSNews" name="onClick" source="Home.Default.RSSChannel.Row1.Column1.ChannelLink" target="Home.Default.RSSChannel.Row1.Column1.ChannelLink"/>
  <handler feature="RSSNews" name="onClick/accessFeed($url)" source="Home.Default.RSSChannel.Row1.Column2.RSSItems" target="Home.Default.RSSChannel.Row1.Column2.RSSItems"/>
  <handler feature="Navigation" name="onClick" source="Home.Default.MainMenu.Item1.SellerLink" target="Home.Default.MainMenu.Item1.SellerLink"/>
  <handler feature="Navigation" name="onClick" source="Home.Default.MainMenu.Item2.SearchLink" target="Home.Default.MainMenu.Item2.SearchLink"/>
  <handler feature="Navigation" name="onClick" source="Home.Default.MainMenu.Item3.CatalogLink" target="Home.Default.MainMenu.Item3.CatalogLink"/>
  <handler feature="Navigation" name="" source="Home.Default.MainMenu.Item4.MapLink" target="Home.Default.MainMenu.Item4.MapLink"/>
  <handler feature="Navigation" name="onClick" source="Home.Default.MainMenu.Item5.TagsLink" target="Home.Default.MainMenu.Item5.TagsLink"/>
  <handler feature="Navigation" name="onClick" source="Home.Default.MainMenu.Item6.HomeLink" target="Home.Default.MainMenu.Item6.HomeLink"/>
  <handler feature="DisplayByCategory" name="onClick/loadCategory($categoryId)" source="Home.Default.FishEyeMenu.Item1.CategoryLink" target="Home.Default.FishEyeMenu.Item1.CategoryLink"/>
  <handler feature="DisplayByCategory" name="onClick/loadCategory1" source="Home.Default.ImageMapMenu.Area1" target="Home.Default.ImageMapMenu.Area1"/>
  <handler feature="DisplayByCategory" name="onClick/loadCategory2" source="Home.Default.ImageMapMenu.Area2" target="Home.Default.ImageMapMenu.Area2"/>
  <handler feature="DisplayByCategory" name="onClick/loadCategory3" source="Home.Default.ImageMapMenu.Area3" target="Home.Default.ImageMapMenu.Area3"/>
  <handler feature="DisplayByCategory" name="onClick/loadCategory4" source="Home.Default.ImageMapMenu.Area4" target="Home.Default.ImageMapMenu.Area4"/>
  <handler feature="DisplayByCategory" name="onClick/loadCategory5" source="Home.Default.ImageMapMenu.Area5" target="Home.Default.ImageMapMenu.Area5"/>
  <handler feature="Advertising" name="onClick/goToJavaEEPage" source="Home.Default.AdvertisePanel.Item1.JavaEEPage" target="Home.Default.AdvertisePanel.Item1.JavaEEPage"/>
  <handler feature="Advertising" name="onClick/goToJavaSEPage" source="Home.Default.AdvertisePanel.Item2.JavaSEPage" target="Home.Default.AdvertisePanel.Item2.JavaSEPage"/>
  <handler feature="Advertising" name="onClick/goToNetBeansIDEPAGE" source="Home.Default.AdvertisePanel.Item3.NetBeansIDEPAGE" target="Home.Default.AdvertisePanel.Item3.NetBeansIDEPAGE"/>
  <handler feature="Tagging" name="onClick/loadTag($tagId)" source="Home.Default.TagCloud.Tags.Tag" target="Home.Default.TagCloud.Tags.Tag"/>
</screen>
</controllers>
```

Figura 3.16: Fragmento de representação arquitetural abstrata MVC - *Controle*

O projeto gerado contém os elementos do padrão arquitetural, correspondentes às entidades de Modelo, Visão e Controle, gerados a partir das informações de nível mais alto. Entretanto, podem ainda existir outros elementos de projeto que serão inseridos na base de código na fase de geração ou mesmo na fase de refinamento manual, por meio de componentes fixos de implementação. Os demais elementos de código que caracterizam *templates* de aplicação (esqueletos estruturais), são obtidos a partir dos modelos de

projeto, por meio de transformações específicas de plataforma, que são implementadas por *templates* de geração JET. Para a caracterização do produto final a partir do resultado obtido ao final deste processo é necessária a aplicação de refinamentos manuais, que consistem no preenchimento do corpo de métodos (artefatos Java) e funções (artefatos Javascript), bem como no preenchimento do conteúdo e atributos de *tags* (artefatos JSP).

3.3 Avaliação do Estudo

Métricas de *software* proveem base quantitativa para o desenvolvimento e avaliação de modelos de desenvolvimento de *software*. As medidas tradicionais para avaliação são LOC (linhas de código) [Albrecht e Gaffney 1983] e Análise de Pontos de Função [Abran e Robillard 1996]. Elas são capazes de descrever diversos aspectos dos produtos de *software*, como complexidade, número de linhas de código, etc. Nossa abordagem utiliza conceitos de programação gerativa para obtenção de artefatos de *software* a partir de modelos, ou seja, existe um processo onde modelos de entrada são traduzidos em código fonte. O código resultante da transformação não é complementamente funcional, sendo que a geração é limitada a um *template* de produto (ou seja, um esqueleto semifuncional da aplicação alvo), que deve ter sua especificação completada manualmente. Embora outras métricas possam ser utilizadas, a alternativa adotada para avaliar os resultados da aplicação da nossa abordagem é uma métrica baseada na contagem de linhas de código. A função desta métrica é fazer uma verificação do percentual de código gerado em relação ao percentual de código escrito manualmente, que em conjunto caracterizam uma versão final de produto de *software*.

A aplicação JavaPetStore¹⁰, usada no estudo de caso, é uma aplicação *web* utilizada como referência para ilustrar como a plataforma JEE pode ser utilizada para a construção de sistemas. Mais especificamente, ela é implementada com o uso a tecnologia JSF, cujo projeto de aplicação é baseado no padrão arquitetural MVC e consiste em um conjunto constituído de artefatos de configuração (XML e outros), artefatos para caracterização da interface gráfica (JSP, CSS), artefatos para a especificação de controladores (Java, JavaScript) e artefatos para a especificação de modelos de dados (Java). Parte do conteúdo de um projeto implantável JSF é comum a todo sistema, incluindo sua estrutura e componentes, embora este possa ser refinado por características específicas. Nossa abordagem contempla este tipo de conteúdo, que caracteriza a base de implementação da linha de produtos, através do encapsulamento do mesmo em transformações específicas de plataforma descritas por *templates* JET.

Os *templates* utilizados para a tradução do modelo arquitetural em código fonte, não permitem a construção completa da aplicação a partir dos artefatos gerados. No entanto, estes artefatos já expressam parte considerável do código fonte necessário. Embora o

¹⁰<http://java.sun.com/developer/releases/petstore/>

estudo de caso tenha experimentado a geração da aplicação de referência também em PHP, a aplicação da métrica de contagem de linhas de código foi feita somente a artefatos JSF. Como uma avaliação, foi executada a aplicação da métrica aos artefatos gerados e aos artefatos originais da aplicação JavaPetStore.

Para a aplicação da métrica aos artefatos, foi executado um procedimento de padronização, caracterizado pela remoção de comentários (da aplicação original) e aplicação de *pretty-printers* ao código (original e gerado). O objetivo deste procedimento é aproximar o estilo de especificação dos conjuntos de artefatos analisados e assim, diminuir a possibilidade de ocorrência de inconsistências na contagem de linhas, já que esta considera delimitadores de blocos e linhas em branco.

O processo de geração dos artefatos da primeira classe analisada (arquivos JSP), é executado a partir do modelo de estados e do modelo de *layout*. Estas duas especificações fornecem grande parte do conteúdo a ser exibido nas telas da aplicação, faltando apenas detalhes específicos de *widgets* (atributos e conteúdo textual) e trechos de código para implementação de comportamento dinâmico, por exemplo, para a exibição de um conjunto de imagens, selecionadas de uma base de dados. O resultado obtido na contagem de linhas de código para esta classe foi expressivo: aproximadamente 68,8% do total das linhas existentes na aplicação original.

A segunda classe de artefatos analisados com a aplicação da métrica, arquivos Javascript, são artefatos utilizados para a especificação de funções, principalmente para o tratamento de eventos na interface gráfica. No estudo, a geração de artefatos desta classe é feita a partir das informações que identificam subestados e transições (no formato evento[guarda]/ação) do modelo de estados. O código Javascript gerado contém basicamente a assinatura das funções obtidas a partir das transições do modelo de estados, bem como a assinatura das funções responsáveis pela caracterização visual de cada um dos subestados de uma tela da aplicação. Em alguns casos, a especificação é um pouco mais completa e pode conter código com complexidade superior a de assinaturas de função, como por exemplo, quando uma transição contém funções de guarda. Neste caso, a especificação contém a assinatura da função que corresponde à guarda e outra especificação que contém a assinatura da função correspondente à ação, mas cujo corpo contém uma condição de desvio identificada pela função de guarda.

A obtenção dos artefatos da terceira classe analisada através da métrica (arquivos Java), consiste na geração de artefatos para especificação das entidades Controlador e Modelo (do padrão MVC), além da cópia de componentes utilitários básicos da especificação da linha de produtos. A geração das entidades de controle é limitada a criação de artefatos para a centralização do processamento de ações disparadas a partir da interface gráfica. O resultado obtido na contagem de linhas para esta classe de artefatos foi de 20,5%, em relação a aplicação original, que contém além dos artefatos de controle centrais, outras especificações para tratamento das ações, de maneira específica. Os artefatos

relativos às entidades modelo são obtidos a partir do modelo de dados e, basicamente, representam a tradução das tabelas e visões deste modelo, em entidades de código que contém campos e métodos. Os conteúdos destas entidades é basicamente composto por campos, métodos completamente funcionais para o encapsulamento dos campos (i.e., métodos *get* e *set*) e de assinaturas de outros métodos utilitários especificados nos modelos de entrada. O resultado da contagem de linhas de código para este tipo de artefato foi em torno de 44% do total das linhas presentes na aplicação original.

A Tabela 3.1 apresenta os resultados obtidos na aplicação da métrica a arquivos Javascript (controladores), JSP (visões) e Java (modelos, controladores e outros componentes). Os resultados mostram que, de forma geral, aproximadamente 52% do total das linhas de código dos elementos analisados puderam ser obtidas por meio de transformações.

	JSP	Javascript	Java (Controle)	Java (Modelo)	Java (outros)	Total
Linhas da JavaPetStore	1623	2203	1452	1255	1827	8360
Linhas Geradas	1116 (68,8%)	452 (20,5%)	54 (3,7%)	610 (44%)	0	2232 (26,7%)
Linhas de Componentes	0	0	295 (20,3%)	0	1827 (100%)	2122 (25,4%)
Total Geral	1116 (68,8%)	452 (20,5%)	349 (24%)	610 (44%)	1827 (100%)	4354 (52,1%)

Tabela 3.1: Métricas de Linhas de Código

3.4 Limitações da Validação/Avaliação

A avaliação aplicada aos artefatos resultantes do processo de geração, considera apenas a contagem de linhas de código, desprezando somente os comentários existentes na aplicação original e nos componentes fixos da base, já que os artefatos gerados não possuem este tipo de elemento. Ainda, algumas das linhas geradas, e consideradas na contagem, não são completamente funcionais e requerem que sejam aplicados refinamentos manuais. Por exemplo, é possível gerar o esqueleto de especificação de elementos como imagens, no entanto não é possível preencher os atributos deste elemento.

Na avaliação do estudo de caso, não houve caracterização das linhas que foram geradas em relação ao esforço necessário para a codificação manual destas linhas.

Outra limitação da avaliação é que, não é considerado o esforço de desenvolvimento dos modelos a partir dos quais é feita a geração de código, tampouco das transformações que permitem a obtenção dos modelos intermediários. Além disso, maior parte das linhas de código obtidas a partir do processo de geração são trechos de código de baixa complexidade ciclomática, por se tratarem de uma especificação estrutural de um produto alvo. Desta forma, o resultado obtido (52%), não pode ser considerado como uma redução de mesma magnitude do esforço de desenvolvimento.

3.5 Resumo do Capítulo

Este capítulo apresentou o estudo de caso, utilizado para a avaliação da abordagem proposta no Capítulo 2. São apresentados os modelos utilizados, bem como a relação entre os modelos de cada um dos passos de geração. A aplicação desenvolvida foi feita com base em uma aplicação de referência, a partir da qual foram identificadas variabilidades e componentes comuns a uma família de sistemas para lojas virtuais. Ainda, a aplicação de referência foi utilizada como base para a avaliação dos resultados obtidos por geração, através de transformações específicas para a plataforma JEE. O estudo é desenvolvido apenas para o caso de sistemas interativos, representados por um conjunto de modelos de análise composto por modelos de características, modelo de dados, modelo de estados, modelo de *layout*, além de modelos de mapeamento responsáveis pelo relacionamento entre elementos de modelos distintos, necessário para a caracterização dos módulos de construção.

Capítulo 4

Discussão e Trabalhos Relacionados

Como mostrado no Capítulo 1, diversas abordagens podem ser utilizadas para a implementação de linhas de produtos de *software*. No entanto, as principais tecnologias de implementação de sistemas adotadas atualmente para a implementação de sistemas modulares apresentam algumas limitações não desejáveis em projetos de linhas de produtos de *software*. A seguir apresentamos um conjunto de parâmetros para avaliação das abordagens, baseados em critérios de avaliação apresentados em [Alves 2007] (granularidade de refinamentos, reusabilidade, tempo de ligação, suporte para a implementação modular de interesses transversais) e [Apel et al. 2006] (coesão de características).

- Complexidade do Mecanismo de Implementação de Características

A implementação do mecanismo de variabilidade pode ser tão simples como definir um valor variável, ou tão complicada como a execução de um processo de geração de código. Ainda, em um cenário mais sofisticado, pode dar suporte a descrição explícita do tipo de variação utilizado por um ativo reutilizável. Este critério de comparação está relacionado ao nível de complexidade de implementação do mecanismo de variabilidade e os possíveis valores que este parâmetro pode assumir são: baixa, média e alta complexidade.

- Granularidade de Refinamentos

Os mecanismos de implementação de linhas de produtos podem diferir quanto a granularidade de refinamentos, ou seja, as posições em que novos itens podem ser inseridos em uma estrutura básica de implementação e o nível de modularização que as peças reutilizáveis que caracterizam os diferentes produtos pode assumir. Os valores utilizados para este parâmetro são: granularidades fina e grossa, onde granularidade fina representa a possibilidade de refinamentos de baixo nível como, por exemplo, refinamento de métodos de classe, e granularidade grossa diz respeito a refinamentos no nível de estruturas, como classes e pacotes.

- Reusabilidade

Com o objetivo de gerenciar a variabilidade em linhas de produtos de *software*, é importante a reutilização de recursos, tanto quanto possível, de modo a conseguir o melhor retorno sobre o investimento no tempo de desenvolvimento destes ativos. Este parâmetro está relacionado ao nível de adaptabilidade dos módulos independentes para implementação de produtos específicos e pode assumir os seguintes valores: baixa, média e alta reusabilidade.

- Tempo de Ligação

Este critério está relacionado ao ponto do tempo de desenvolvimento, no qual as decisões relacionadas às variações são feitas, com o objetivo de combinar módulos de construção que determinam o comportamento do produto final. Os possíveis valores para este parâmetro são: pré-processamento, compilação, implantação e tempo de execução.

- Coesão de Características

A coesão é propriedade de uma característica encapsular todas as unidades que contribuem para a sua realização em um único módulo de implementação. Os valores utilizados para caracterização deste parâmetro são: alta, média e baixa coesão.

- Suporte para implementação de interesses transversais

Este critério é utilizado para a avaliação do apoio para a implementação modular de interesses transversais. Este parâmetro é considerado, pois a aplicação da variabilidade em linhas de produtos de *software* pode ser frequentemente transversal, como apontado por um conjunto significativo de trabalhos [Gacek e Anastasopoulos 2001] [Batory et al. 2003] [Mezini e Ostermann 2004] [Lopez-herrejon et al. 2005] [Liu et al. 2006]. Segundo Colyer e outros [Colyer et al. 2004], interesses transversais podem ser classificados como homogêneos ou heterogêneos. Interesses transversais homogêneos refinam múltiplos pontos de extensão de uma aplicação (*joinpoints*) através de uma única estrutura (*advice*). Interesses transversais heterogêneos refinam múltiplos pontos de extensão, com múltiplas estruturas. Os valores utilizados para avaliação deste parâmetro são: suportada, parcialmente suportada ou não suportada.

- Nível de Abstração

Embora linhas de produtos de *software* possam ser realizadas somente no nível de código, diferentes abordagens utilizam conceitos de programação gerativa com o objetivo de aumentar o nível de abstração no processo de geração, através de modelos e transformações entre modelos, e possibilitar a geração de código.

4.1 Avaliação das abordagens

Nas seções seguintes é feita a avaliação de algumas das possíveis abordagens para desenvolvimento de linhas de produtos de *software*. Parte desta avaliação é feita através dos resultados apresentados em [Alves 2007] e [Apel et al. 2006]. A implementação de linhas de produtos de *software* pode ser feita por técnicas que lidam com variabilidade somente no nível de código como Compilação Condicional, *Frameworks*, Programação Orientada a Aspectos, Programação Orientadas a Características e Programação Orientada a Características com *Aspectual Mixin-layers*. No entanto, outras técnicas de implementação utilizam Programação Gerativa para a obtenção de artefatos de código a partir de artefatos de mais alto nível: a nossa abordagem, Sobreposição de Variantes, Desenvolvimento Orientado a Características e Dirigido por Modelos, Desenvolvimento Dirigido por Modelos com *Early-Aspects*, PL-AspectualACME e MAS-PL.

4.1.1 Compilação Condicional

A Compilação Condicional é uma técnica de baixa complexidade e baixo nível de abstração, capaz de especificar a inclusão ou não de um módulo inteiro (granularidade grossa) ou inclusão ou não de uma única linha de código (granularidade fina). Neste último caso, mais frequentemente utilizado. A reusabilidade é limitada e a manutenção é problemática. Em Compilação Condicional a combinação de características é feita através de *scripts* de pré-processamento, que selecionam os componentes a serem compilados. Como o código que especifica os ativos reutilizáveis não é expresso por módulos explícitos de características, o grau de coesão dos módulos é baixo. A Tabela 4.1 a seguir sumariza as propriedades desta abordagem.

Critério	
Complexidade	baixa
Granularidade	fina e grossa
Reusabilidade	baixa
Tempo de Ligação	pré-processamento
Coesão	baixa
Interesses Transversais	não suportada
Nível de Abstração	baixo

Tabela 4.1: Propriedades da Compilação Condicional

4.1.2 *Frameworks* de aplicação

Um *framework* é um aplicativo parcial que fornece um conjunto integrado de estruturas específicas de domínio e funcionalidade, com alto índice de reutilização. A implementação de linhas de produtos de *software* com *frameworks* é uma abordagem complexa,

baseada frequentemente em padrões de projeto [Gamma et al. 1995] para implementação de variabilidade, e permitem refinamentos de granularidade grossa. No entanto, alguns padrões de projeto como *Template Method*, permitem a especificação de refinamentos de granularidade fina. Em geral, a combinação de características é feita em tempo de compilação, embora em função de alguns padrões de projeto como *Strategy* e *Bridge*, a combinação possa vir a ser feita em tempo de execução. *Frameworks* não suportam implementação modular de interesses transversais, pois em Programação Orientada a Objetos o comportamento transversal é usualmente expresso por fragmentos espalhados em diversos componentes funcionais. A Tabela 4.2 a seguir sumariza as propriedades desta abordagem de implementação.

Critério	
Complexidade	alta
Granularidade	geralmente grossa
Reusabilidade	alta
Tempo de Ligação	compilação, execução
Coesão	baixa
Interesses Transversais	não suportada
Nível de Abstração	geralmente baixo

Tabela 4.2: Propriedades de *Frameworks*

4.1.3 Programação Orientada a Aspectos

As principais vantagens da utilização de aspectos são: o suporte a implementação de interesses transversais e a possibilidade de introdução de variabilidade imprevista. A implementação de linhas de produtos de *software* baseadas em Programação Orientada a Aspectos é uma técnica de média complexidade, que oferece um paradigma que visa obter melhor separação de interesses e, assim, facilitar o desenvolvimento de aplicações proporcionando maior apoio para a modularização nos níveis de projeto e implementação. Embora possua mecanismos avançados de modularização, o reuso não é alto, pois a maioria dos mecanismos de pontos de junção das linguagens orientadas a aspectos são baseados em construções de nível sintático, o que torna os aspectos altamente dependentes de identificadores [Alves 2007]. A granularidade de refinamentos oferecidos por linguagens orientadas a aspectos depende do seu modelo de pontos de junção, ou seja, os lugares em que aspectos podem anexar nova funcionalidade. AspectJ¹ e CaesarJ² suportam pontos de junção que interceptam os limites de métodos, como por exemplo, chamadas de métodos e acessos a campos. Nesta forma de extensão, refinamentos de granularidade fina são suportados. Aspectos também suportam introduções que permitem refinamento na

¹<http://www.eclipse.org/aspectj/>

²<http://caesarj.org/>

estrutura estática de classes existentes, que podem introduzir novos campos, métodos e relações de herança. Combinação em tempo de execução só é possível em implementações que suportam *weaving* em tempo de carregamento ou execução [Bergel et al. 2006], [Bockisch et al. 2004], [Hirschfeld 2003]. A Tabela 4.3 resume as propriedades de abordagens baseadas em aspectos.

Critério	
Complexidade	média
Granularidade	finha e grossa
Reusabilidade	média
Tempo de Ligação	compilação, execução, implantação
Coesão	baixa
Interesses Transversais	suportada
Nível de Abstração	baixo

Tabela 4.3: Propriedades da Programação Orientada a Aspectos

4.1.4 Programação Orientada a Características

A Programação Orientada a Características representada no ambiente AHEAD utiliza um mecanismo para a implementação de módulos independentes baseados em camadas, que captura *mixin-layers* explicitamente, e provê ferramentas para a composição de camadas que consistem em definições parciais de classes, etc. Em programação orientada a características os módulos podem ser combinados livremente, o que contribui para o reuso. No entanto, a reusabilidade não é alta, devido à potencial alta fragmentação de código na forma de declarações *refines*. Além disso, a Programação Orientada a Características tem uma idéia de granularidade grossa para a variabilidade, onde a menor unidade de composição é uma camada *delta* de declarações *refines* sobre uma base existente. Neste paradigma, a combinação de ativos geralmente é feita em pré-processamento, sendo que o resultado da combinação é traduzido para uma linguagem compilável (como Java), embora outras alternativas de implementação possam apoiar diretamente a combinação de características em tempo de execução. A Programação Orientada a Características não oferece suporte a implementação modular de interesses transversais homogêneos, apenas heterogêneos, por meio das camadas *delta* [Alves 2007]. A Tabela 4.4 a seguir sumariza as propriedades desta abordagem.

4.1.5 Programação Orientada a Características com *Aspectual-mixin-layers*

Em [Apel et al. 2006], aspectos e características são utilizados de forma conjunta para a especificação de ativos reutilizáveis, integrando conceitos de Programação Orien-

Critério	
Complexidade	média
Granularidade	grossa
Reusabilidade	média
Tempo de Ligação	pré-processamento
Coesão	alta
Interesses Transversais	parcialmente suportada
Nível de Abstração	baixo

Tabela 4.4: Propriedades da Programação Orientada a Características

tada a Aspectos a Programação Orientada a Características. A idéia fundamental desta abordagem é incorporar aspectos a *mixin-layers* (Figura 4.1). *Mixin-layers* implementam fragmentos de classes, da mesma forma os aspectos que interceptam estas classes podem ser também considerados como fragmentos. A Programação Orientada a Características com Aspectual-Mixin-layers decompõe um projeto orientado a aspectos em camadas de implementação de características. Assim, cada camada (i.e., uma Aspectual-mixin-layer) contém um conjunto de *mixins* e um conjunto de aspectos. Nesta abordagem, *mixins* são utilizadas para a implementação de estruturas estáticas, heterogêneas relativas a hierarquia transversal, enquanto os aspectos são utilizados para a implementação de estruturas dinâmicas, homogêneas, não transversais. Em outras palavras, *aspectual-mixin-layers*, são *mixins* utilizados para refinar outros *mixins* e dependem, portanto, da estrutura da camada superior, a qual são aplicadas.

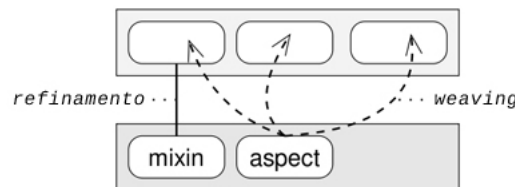


Figura 4.1: Aspectual mixin-layers

Aspectual-mixin-layers permite programação orientada a características com melhor suporte a implementação de interesses transversais. A Tabela 4.5 a seguir sumariza as propriedades desta abordagem.

4.1.6 Abordagem de *Templates* Baseados em Sobreposição de Variantes

Em [Czarnecki 2005], Czarnecki propõe uma abordagem genérica baseada em *templates* para mapeamento de modelos de características para outros modelos que representam a variabilidade em diversos formatos. Nesta abordagem, a representação de uma linha de produtos é feita por um modelo de características e um *template* de modelo. O modelo de características define a hierarquia de características, bem como as restrições em

Critério	
Complexidade	alta
Granularidade	finha e grossa
Reusabilidade	média
Tempo de Ligação	pré-processamento e compilação
Coesão	alta
Interesses Transversais	suportada
Nível de Abstração	baixo

Tabela 4.5: Propriedades da Programação Orientada a Características com *Aspectual-mixin-layers*

suas possíveis configurações. Em contraste com abordagens baseadas em variabilidade em que modelos distintos correspondentes a diferentes características são compostos, a abordagem utiliza um modelo que representa a sobreposição de todos os variantes cujos elementos estão relacionados a características através de anotações (i.e., um grande modelo anotado que contém conjuntamente as diferentes possibilidades). Os elementos do *template* de modelo podem ser anotados com condições de presença e meta-expressões. Estas anotações são definidas em termos de características, e podem ser avaliadas em relação a configurações específicas de características. Uma anotação de presença indica se o elemento deve estar presente ou deve ser removido de uma instância do *template*. Meta-expressões são utilizadas para computar atributos de elementos do modelo, como o nome do elemento ou o tipo retornado por uma operação. A abordagem mostra um protótipo de implementação que utiliza modelos de classes e atividades UML, para a geração de *templates*. A Figura 4.2 sumariza a abordagem.

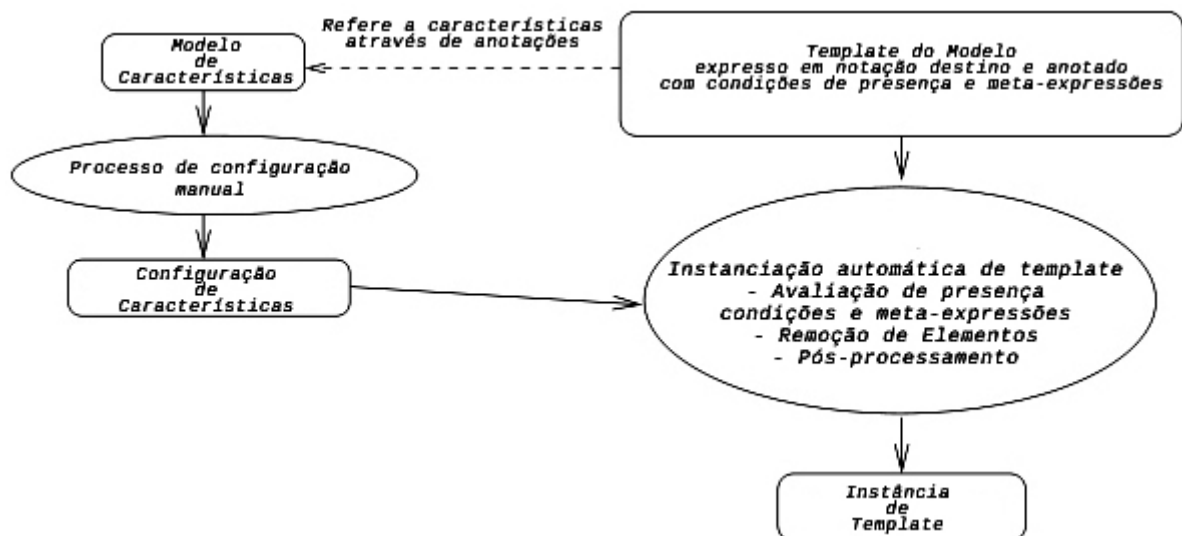


Figura 4.2: Sobreposição de variantes de [Czarnecki 2005]

O estudo de caso apresentado no trabalho de [Czarnecki 2005] mostra como a obtenção de instâncias de *templates* de modelos pode ser alcançada a partir de instanciação de mo-

delos, cujos elementos marcados por anotações identificam características. O experimento é desenvolvido apenas no nível de modelagem e não considera detalhes de implementação. A técnica de Sobreposição de Variantes pode ser classificada como uma técnica de média complexidade, de alto nível de abstração, com granularidade fina e grossa no nível de modelagem, já que anotações podem ser feitas em diferentes pontos na estrutura dos modelos. No entanto, a reusabilidade dos módulos nesta abordagem é limitada e a coesão de características é baixa, já que a variabilidade não é expressa através de módulos independentes. A Tabela 4.6 resume as propriedades da abordagem.

Critério	
Complexidade	média
Granularidade	fina e grossa
Reusabilidade	média
Tempo de Ligação	pré-processamento
Coesão	baixa
Interesses Transversais	não suportada
Nível de Abstração	alto

Tabela 4.6: Propriedades da Superimposição de Variantes

4.1.7 Desenvolvimento Orientado por Características e Dirigido por Modelos

Em [Trujillo et al. 2007], Trujillo e outros propõem uma abordagem que une Programação Orientada a Características e Desenvolvimento Dirigido por Modelos. Nela, é mostrada uma forma de se obter a implementação de características a partir de modelos em alto nível de abstração e transformações entre modelos. A técnica para realização de variabilidade lida não somente com código mas também diretamente com modelos, permitindo a composição dos ativos reutilizáveis em nível de modelagem, ao invés de um conjunto de artefatos de implementação. A abordagem utiliza um mecanismo avançado, que preserva as propriedades da Programação Orientada a Características e permite a realização de ativos reutilizáveis a partir de modelos de alto nível de abstração. A Tabela 4.7 resume as propriedades desta abordagem.

4.1.8 Nossa abordagem para Desenvolvimento Orientado por Características e Dirigido por Modelos (FOMDD*)

Nossa abordagem para criação de linhas de produtos utiliza conceitos apresentados nas abordagens de [Trujillo et al. 2007] e [Czarnecki 2005]. Modelos de características são associados a modelos de análise completos (i.e., sem fragmentação orientada a características) através de anotações, com o objetivo de caracterizar os módulos que constituem os

Critério	
Complexidade	alta
Granularidade	grossa
Reusabilidade	média
Tempo de Ligação	pré-processamento
Coesão	alta
Interesses Transversais	parcialmente suportada
Nível de Abstração	alto

Tabela 4.7: Propriedades do Desenvolvimento Orientado por Características e Dirigido por Modelos

produtos de *software*. O mecanismo de transformação implementado na abordagem toma como entrada estes modelos de análise anotados para a geração de modelos intermediários de arquitetura. Tais modelos arquiteturais também são anotados por características, e são finalmente convertidos em especificações de código, de acordo com a tecnologia de implementação. O processo de geração pode ser executado em duas vias. Na primeira delas, o código é gerado para um domínio de aplicação, e a combinação de características para produtos específicos é feita através dos ativos de código obtidos por transformações dos modelos. Na segunda via, o código é obtido através da seleção de características expressas no nível de modelos, que são transformados sucessivamente em função de características específicas de um produto.

Uma propriedade importante da nossa abordagem quanto ao processo utilizado para a especificação de características no nível de modelagem, é que, diferentemente da abordagem original de desenvolvimento orientado a características e dirigido por modelos [Trujillo et al. 2007] que utiliza modelos segmentados para a representação modular de características, nossa abordagem utiliza modelos não fragmentados, cujos elementos são marcados com anotações. Estas anotações identificam a fragmentação orientada a características do domínio de aplicação. Segundo Czarnecki [Czarnecki 2005], apesar de um modelo de características poder representar semelhanças e variabilidades de uma forma taxonômica muito concisa, características em um modelo de características são apenas símbolos. O mapeamento de características para outros modelos, tais como comportamentais ou de especificações de dados, é que confere-lhes semântica.

Nossa abordagem utiliza uma idéia simples de implementação de variabilidade, tal qual na Compilação Condicional, baseada em anotação de elementos, embora utilize um arcabouço complexo fundamentado em Desenvolvimento Dirigido por Modelos. A abordagem utiliza modelos não fragmentados em função de características, ou seja, sem modularização explícita, o que seguindo a rigor as práticas de Programação Orientada a Características não é desejável. No entanto, o estudo apresentado utiliza modelos auxiliares para o mapeamento entre características e elementos de outros modelos, para facilitar a anotação dos modelos de entrada. São permitidos refinamentos de granularidades fina e grossa, já que

anotações podem ser feitas em elementos de diferentes níveis da estrutura dos modelos, embora a noção de refinamento esteja relacionada a uma atualização manual dos modelos, bem como das anotações que caracterizam os elementos adicionados ou removidos. A Tabela 4.8 a seguir sumariza as propriedades da nossa abordagem.

Critério	
Complexidade	média
Granularidade	finas e grossas
Reusabilidade	média
Tempo de Ligação	pré-processamento
Coesão	baixa (modelagem) e alta (código)
Interesses Transversais	não suportada
Nível de Abstração	alto

Tabela 4.8: Propriedades da nossa Abordagem para Desenvolvimento Orientado por Características e Dirigido por Modelos

A seguir apresentamos outras abordagens para implementação de linhas de produtos de *software* que oferecem suporte a programação orientada a aspectos. Nossa abordagem não oferece este tipo de suporte e o objetivo da apresentação das abordagens a seguir é simplesmente possibilitar a avaliação da nossa abordagem em relação às propriedades de abordagens fundamentadas em programação orientada a aspectos mais recentes.

4.1.9 Desenvolvimento Dirigido por Modelos com *Early-Aspects*

Tradicionalmente, o desenvolvimento orientado a aspectos concentra-se na fase de implementação de *software*: aspectos são identificados e capturados principalmente no código. Mas aspectos são evidentes no início do ciclo de vida de um *software*, como durante a análise de requisitos e desenvolvimento da arquitetura. *Early-Aspects* [Baniassad et al. 2006] permite identificar estes aspectos antecipadamente e garante a captura adequada dos aspectos relacionados com o domínio do problema (em oposição a apenas à implementação).

Em [Batista et al. 2008] é apresentada uma abordagem para desenvolvimento dirigido por modelos em conjunto com *early-aspects*, que utiliza um modelo de características orientado por aspectos a ser transformado em um modelo de aspectos antecipados e este em uma especificação arquitetural orientada a aspectos. Esta especificação é feita através de uma extensão orientada a aspectos da linguagem de descrição arquitetural ACME [Garlan et al. 1997], chamada AspectualACME [Batista et al. 2006] e que provê um mecanismo de seleção de variantes para a configuração de arquiteturas específicas. A especificação de uma linha de produtos é feita em termos de modelos de características por questões de usabilidade e os atributos de cada uma das características podem ser anotados para indicar se elas são concretas ou aspectuais. No segundo caso, devem também ser indicadas quais

as outras características serão interceptadas pela característica transversal. O modelo de características é transformado para um formato intermediário chamado *early-design*, entre a especificação de requisitos e desenvolvimento da arquitetura. Neste formato os *early-aspects* são identificados e finalmente este formato é transformado em uma descrição arquitetural orientada a aspectos. Esta abordagem preserva as propriedades da programação orientada a aspectos como granularidades fina e grossa (em níveis de modelagem e código), suporte a implementação modular de interesses transversais, etc. Ela pode ser classificada como uma estratégia de alta complexidade e alto nível de abstração. A Tabela 4.9 resume as propriedades da abordagem.

Critério	
Complexidade	alta
Granularidade	fin e grossa
Reusabilidade	média
Tempo de Ligação	pré-processamento e compilação
Coesão	baixa
Interesses Transversais	suportada
Nível de Abstração	alto

Tabela 4.9: Propriedades do Desenvolvimento Dirigido por Modelos com *Early-Aspects*

4.1.10 PL-AspectualACME

Em [Adachi et al. 2009], é apresentada uma abordagem que combina uma linguagem de descrição arquitetural orientada a aspectos, PL-AspectualACME, com uma linguagem para modelagem de variabilidade, VML4Arch. PL-AspectualACME é usada para especificar a descrição arquitetural geral que consiste na representação modular de similaridades e variabilidades arquiteturais. VML4Arch é usada para derivar automaticamente descrições arquiteturais de produtos específicos através da especificação de dependências entre características e as variabilidades que devem ser incluídas nos produtos. PL-AspectualACME promove uma união natural entre linhas de produtos de *software* e abstrações arquiteturais orientadas a aspectos, caracterizada por duas fases: na primeira delas são especificados invariáveis e variabilidade, e a configuração geral da linha de produtos na fase de projeto do domínio de aplicação. A segunda especifica a derivação de uma descrição arquitetural de um produto específico. VML4Arch é utilizada para estabelecer a ligação entre o modelo de características e os modelos de arquitetura de *software*, provendo primitivas para a descrição de configurações específicas para produtos. A utilização conjunta destas linguagens permite que a arquitetura da linha de produtos e instanciação da linha de produtos sejam feitas de forma modular, e simplificam a evolução da arquitetura geral, bem como das descrições de instanciação para caracterização de produtos. A Tabela 4.10 exhibe as propriedades da abordagem.

Critério	
Complexidade	alta
Granularidade	finas e grossas
Reusabilidade	média
Tempo de Ligação	pré-processamento e compilação
Coesão	baixa
Interesses Transversais	suportada
Nível de Abstração	alto

Tabela 4.10: Propriedades de PL-AspectualACME

4.1.11 MAS-PL

Em [Cirilo et al. 2009], é apresentada uma abordagem fundamentada em Engenharia de *Software* Orientada a Agentes [Wooldridge e Ciancarini 2001], que permite lidar com gerência de variabilidades e derivação automática de produtos em linhas de produtos de sistemas multi-agentes. Sistemas multi-agentes são sistemas que contêm diversos componentes dinâmicos em interação, cada um deles com uma *thread* de controle específica, durante a execução de complexos protocolos coordenados. A Engenharia de *Software* Orientada a Agentes é um paradigma que endereça a análise, o projeto, e a implementação de sistemas de *software* baseados em definições em alto nível de abstração, como agentes, papéis, organizações, etc.

A abordagem estende a abordagem dirigida por modelos da ferramenta GenArch [Cirilo et al. 2008], com três modelos: modelo de características, modelo de arquitetura e modelo de configuração, e sua principal idéia é incorporar um modelo de arquitetura específico de agentes à especificação de linhas de produtos de *software*, com o objetivo de lidar com conceitos de mais alto nível, incluindo conceitos de agentes como crenças, objetivos e planos. O modelo de arquitetura define uma representação visual dos elementos de implementação da linha de produtos (classes, aspectos, *templates*, diretórios, arquivos e fragmentos) a serem relacionados às características do domínio. Além disso, o modelo de arquitetura permite definir mapeamentos entre os seus conceitos e respectivos elementos de implementação. Desta forma, o código fonte que implementa os fragmentos de um produto é obtido pela seleção de elementos específicos.

Esta abordagem oferece uma estratégia de gerência de variabilidade orientada a código, que suporta derivação automática de produtos baseada em três passos: construção automática de modelos, sincronização de artefatos e derivação de produto. A abordagem pode ser classificada como uma alternativa de alta complexidade, alto nível de abstração, com suporte a granularidade fina e grossa nos níveis de modelagem, projeto e código. A Tabela 4.11 resume as propriedades da abordagem.

Critério	
Complexidade	alta
Granularidade	finas e grossas
Reusabilidade	média
Tempo de Ligação	pré-processamento e compilação
Coesão	baixa
Interesses Transversais	suportada
Nível de Abstração	alto

Tabela 4.11: Propriedades de MAS-PL

4.2 Comparação de Abordagens

A avaliação das propriedades das abordagens para desenvolvimento de linhas de produtos permite a escolha do mecanismo de implementação de variabilidade em função de um conjunto consistente de parâmetros. Por exemplo, reusabilidade, granularidade de refinamentos e coesão podem ser consideradas as propriedades fundamentais para desenvolvimento de uma linha de produtos de *software* em particular. Alto nível de abstração também é desejável no desenvolvimento de *software*, no entanto, em geral o suporte para especificação de *software* em termos de modelos requer processos de transformação elaborados. Abordagens com suporte a aspectos permitem a implementação de interesses transversais de forma modular, o que também tem impacto positivo na granularidade de refinamentos. Os diferentes mecanismos de implementação podem ter diferentes desempenhos na avaliação segundo os critérios apresentados, e a importância desses parâmetros pode variar em função de diferentes domínios e diferentes linhas de produtos. Assim, não é possível presumir que uma única técnica irá maximizar todos os critérios e ser utilizada de maneira uniforme em todos os projetos. A Tabela 4.12 resume as propriedades de opções para implementação de linhas de produtos, apresentadas na Seção 4.1.

A Tabela 4.13, apresenta uma comparação mais específica entre a abordagem original de FOMDD e nossa abordagem.

Técnica	Complexidade	Granularidade	Reusabilidade	Tempo de Ligação	Coesão	Interesses Transversais	Nível de Abstração
Compilação Condicional	●	○ ●	○	PP	○	○	○
Frameworks	○	●	●	CP, TE	○	○	○
AOP	●	○ ●	●	CP, TE, IM	○	●	○
FOP	●	●	●	PP	●	●	○
FOP com AMLs	○	○ ●	●	PP, CP	●	●	○
Sobreposição de Variantes	●	○ ●	●	PP	○	○	●
FOMDD	○	●	●	PP	●	●	●
FOMDD*	●	○ ●	●	PP	○ ⁺ ● ⁺⁺	○	●
MDD com Early-Aspects	○	○ ●	●	PP, CP	○	●	●
PL-AspectualACME	○	○ ●	●	PP, CP	○	●	●
MAS-PL	○	○ ●	●	PP, CP	○	●	●

Legenda: ● nível alto / granularidade grossa / complexidade baixa
 ● nível médio
 ○ nível baixo / granularidade fina / complexidade alta
 + Modelagem
 ++ Código

PP: Pré-processamento
 CP: Compilação
 TE: Tempo de execução
 IM: Implantação

Tabela 4.12: Tabela Comparativa

	FOMDD	FOMDD*
Modelos de Entrada	Modelo de Estados (W3C SCXML)	- Modelos de Características (Extensão de Diagrama de Classes UML) - Modelo de Estados (Extensão de Diagrama de Estados UML) - Modelo de Dados (Extensão de Diagrama de Classes UML) - Modelo de <i>Layout</i> (HTML)
Modularização de Características	Fragmentação explícita de modelos de entrada	Fragmentação expressa por modelos de mapeamento
Arquitetura dos Sistemas	Específica para <i>portlets</i>	Padrão arquitetural especificado de forma abstrata
Tecnologia Utilizada	Portlets/JEE	- JSF/JEE - PHP
Passos de Geração	6 (A e B)	8 (A) / 6 (B)
Refinamentos Manuais	- Refinamentos em linguagem específica de plataforma (<i>portlets</i>) em estágio intermediário de geração - Refinamentos finais do <i>template</i> de aplicação gerado	- Mapeamento entre modelos - Refinamentos finais do <i>template</i> de aplicação gerado

Tabela 4.13: Tabela Comparativa FOMDD e FOMDD*

Capítulo 5

Conclusões e Perspectivas Futuras

Este trabalho apresentou uma abordagem integrada para desenvolvimento orientado a características e dirigido por modelos baseada em [Trujillo et al. 2007] e [Czarnecki 2005]. Ele apresenta também um estudo de caso para a avaliação de sua aplicabilidade, consistindo no desenvolvimento de um exemplo não trivial de linha de produtos para Lojas Virtuais.

Segundo [Boehm 1999], existem três estratégias para aumentar a produtividade de *software*: trabalhar mais rápido, trabalhar mais inteligentemente e evitar trabalho desnecessário. Esta última permite maior retorno do esforço aplicado, e pode ser alcançada através de abordagens sistemáticas de reuso, onde a Programação Orientada por Características e o Desenvolvimento Dirigido por Modelos podem ser consideradas alternativas para capitalizar os esforços de reutilização.

Nossa abordagem é uma proposta para o desenvolvimento automatizado de linhas de produtos de *software*, que utiliza como entrada um conjunto de modelos anotados por características, responsável pela caracterização das aplicações em diferentes níveis de abstração, e que possibilita a obtenção de *templates* de aplicações (i.e., esqueletos semi-funcionais) por meio de transformações sucessivas. A principal vantagem na utilização de abordagens gerativas como a nossa, é a geração de código fonte a partir de elementos abstratos reusáveis, o que aumenta a produtividade no processo por meio da automação de tarefas tediosas do desenvolvimento de *software*. Além disso, um processo caracterizado por etapas facilita a racionalização do fluxo de trabalho, permitindo que as tarefas sejam mais facilmente divididas entre os membros de uma equipe de desenvolvimento. Em geral, o trabalho dos arquitetos concentra-se na definição dos modelos e das transformações entre modelos, que caracterizam a implementação de um domínio de aplicações. A função de programadores é, basicamente, caracterizada pelo preenchimento de lacunas existentes na especificação de produtos individuais. Desta forma, o grau de especialização necessário aos programadores das equipes de desenvolvimento é mais baixo, já que eles não precisam conhecer detalhes de arquitetura.

Outro ponto positivo da abordagem é que a utilização de modelos de arquitetura en-

capsulados por transformações possibilita a geração de sistemas perfeitamente aderentes ao estilo arquitetural escolhido. Como o processo de construção é feito de forma automática, com base em modelos e transformações, não existe a possibilidade de introdução de interferências, em se tratando da construção da arquitetura de um produto alvo específico. Além disso, a utilização de um formato intermediário independente de plataforma para a especificação do padrão arquitetural permite sua implementação concreta em diferentes plataformas alvo, diferentemente do estudo apresentado em [Trujillo et al. 2007].

O estudo de caso fundamentado na metodologia proposta mostrou a viabilidade da aplicação do processo de geração de *templates* de código baseado em modelos e anotações de modelos. Os resultados da aplicação da métrica de contagem de linhas de código mostraram que aproximadamente 52% do total do conteúdo dos artefatos analisados puderam ser obtidos por meio de transformações.

Considerando a discussão apresentada, a principal contribuição deste trabalho é a caracterização de uma abordagem gerativa de média complexidade para geração de *templates* de aplicações a partir de modelos de análise anotados por características, além da caracterização dos modelos necessários para a generalização da abordagem apresentada em [Trujillo et al. 2007]. Além disso, a abordagem permite a utilização de modelos não fragmentados explicitamente em termos de características como entrada num processo de geração de código orientado a características, permitindo assim a utilização de modelos legados, bem como de ferramentas tradicionais de modelagem de *software*. A utilização da abordagem reduz o esforço empregado na realização dos artefatos de código de uma linha de produtos especificada em alto nível de abstração, através da transformação sucessiva dos modelos de entrada, além de racionalizar o fluxo de desenvolvimento em um processo caracterizado por etapas. A abordagem responde às questões da Seção 1.2 como segue:

- Como representar através de modelos, os elementos essenciais de construção de uma linha de produtos de *software*?

Nossa abordagem representa estes elementos através de modelos de análise anotados por características. Os modelos de análise capturam os requisitos do domínio de aplicação e a associação de relação entre elementos destes modelos e de modelos de características, por meio de anotações, possibilita a especificação de produtos individuais.

- Como representar através de modelos a arquitetura de famílias de sistemas de *software*?

O estudo de caso para geração da família de sistemas *web* interativos Lojas Virtuais mostra como um padrão arquitetural pode ser representado de forma abstrata, para possibilitar a geração de implementações em diferentes plataformas específicas. No estudo as aplicações interativas foram representadas segundo o padrão arquitetural MVC, através de modelos abstratos e respectivos elementos que representam as en-

tidades do padrão: Modelo, Visão e Controlador. Apesar do estudo de caso utilizar o padrão MVC, acreditamos que a abordagem seja genérica o suficiente para que esta possa ser aplicada a diferentes padrões arquiteturais. No entanto, diferentes padrões arquiteturais podem vir a exigir conjuntos específicos de modelos de análise, a partir dos quais a especificação arquitetural possa ser obtida.

- Qual o impacto da ordem de aplicação da composição horizontal (composição de características) em relação ao refinamento vertical (refinamento para o nível de implementação)?

No estudo de caso são apresentados dois possíveis caminhos para a síntese de produtos. O primeiro deles (Via A), utiliza transformações sucessivas entre modelos para obtenção de código orientado a características, que ao final do processo é combinado através dos mecanismos de composição do ambiente AHEAD. O processo completo de geração nesta via de geração é executado em oito passos de transformação, sendo que inicialmente são gerados os artefatos de código do domínio, para posterior composição e consequente configuração de produtos específicos. Além disso, os artefatos orientados a características necessitam de conversão para formatos de mais baixo nível, antes de configurarem um projeto de código compilável. No entanto, esta via de geração permite que os refinamentos manuais finais sejam feitos em módulos individuais, que podem ser aplicados ao *template* do produto, de forma não intrusiva. A segunda via de síntese (ViaB), utiliza a composição de características por meio de modelos de alto nível, definindo em alto nível de abstração a configuração do produto alvo. O processo de geração por esta via é executado em 6 passos de transformação e, como a combinação de características é feita a nível de modelos, a transformação para obtenção de código é feita diretamente para uma notação compilável. Desta forma, esta via de geração, permite a utilização de linguagens sem suporte a implementação modular de características. No entanto, a caracterização dos refinamentos finais nesta via é feita diretamente no código do *template* da aplicação, de forma intrusiva.

- Como se dá a evolução de sistemas gerados através da abordagem apresentada?
Em nossa abordagem a evolução de uma linha de produtos é feita principalmente através da atualização manual dos modelos de análise, onde são feitas inserções de novos elementos. Nossa abordagem não permite a geração de um módulo de refinamento individual. O processo de geração para um produto refinado consiste na elaboração de um projeto de produto que contenha, além das características do produto original, novas características. O processo de geração deve ser executado do início ao fim, independente da via utilizada, para a geração dos novos artefatos. A informação das etapas manuais de preenchimento de modelos de mapeamento e código devem ser salvas, para poupar o trabalho de refazer completamente os

modelos de mapeamento, bem como o preenchimento de métodos/funções, atributos e conteúdo de *tags*, já que, no caso da evolução da linha de produtos, parte desta informação deverá ser reutilizada.

Embora o estudo de caso tenha mostrado o processo de geração de um único produto específico da linha de produtos, sua obtenção foi feita a partir de combinação de características, o que sugere que outros produtos possam ser obtidos da mesma forma. Além, disso a implementação dos módulos foi obtida a partir da seleção orientada a características de elementos de modelos auxiliares, que contém informação funcional sobre o domínio de aplicação, que pode ser combinada para a caracterização de diferentes produtos individuais.

As principais limitações da abordagem apresentada dizem respeito a falta de suporte para implementação modular de interesses transversais e ao baixo nível de coesão de características no nível de modelagem. Ainda, a noção de refinamentos de modelos está relacionada à introdução de novos elementos e respectivas anotações nos modelos de entrada. Em abordagens orientadas por características é desejável que refinamentos sejam elementos modulares e que possam ser aplicados individualmente. Além disso, existe a limitação de que sempre quando ocorre uma regeneração do sistema, é necessária a reatualização dos refinamentos manuais.

O sistema gerador utilizado em nossa abordagem é um protótipo para a geração de código orientado a características a partir de modelos. Algumas melhorias que podem ser aplicadas ao mecanismo dizem respeito a utilização integral de modelos compatíveis com metamodelos padrão como MOF¹ e utilização de linguagens formais de transformação baseadas em QVT². Ainda, o desenvolvimento de uma linguagem para a especificação de *templates* JET orientados a características, pode reduzir consideravelmente o esforço de desenvolvimento das transformações específicas de plataforma. Um outro possível trabalho futuro diz respeito à generalização dos modelos de entrada para que seja possível a especificação de sistemas onde a interação não é baseada em interfaces gráficas com o usuário.

¹<http://www.omg.org/mof/>

²<http://www.omg.org/spec/QVT/1.0/>

Referências Bibliográficas

- [Abran e Robillard 1996] Abran, A. e Robillard, P. N. (1996). Function Points Analysis: An Empirical Study of Its Measurement Processes. *IEEE Trans. Softw. Eng.*, 22(12):895–910.
- [Adachi et al. 2009] Adachi, E., Batista, T., Kulesza, U., Medeiros, A. L., Chavez, C., e Garcia, A. (2009). Variability Management in Aspect-Oriented Architecture Description Languages: An Integrated Approach. *Software Engineering, Brazilian Symposium on*, 0:1–11.
- [Adams et al. 2009] Adams, B., De Meuter, W., Tromp, H., e Hassan, A. E. (2009). Can we refactor conditional compilation into aspects? In *AOSD '09: Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development*, pp. 243–254, Charlottesville, Virginia, USA, New York, NY, USA. ACM.
- [Albrecht e Gaffney 1983] Albrecht, A. J. e Gaffney, J. E. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Trans. Softw. Eng.*, 9(6):639–648.
- [Altunel e Tolun 2007] Altunel, Y. e Tolun, M. R. (2007). Component-based software development with component variants. In *SE'07: Proceedings of the 25th conference on IASTED International Multi-Conference*, pp. 235–241, Innsbruck, Austria, Anaheim, CA, USA. ACTA Press.
- [Alves 2007] Alves, V. (2007). *Implementing Software Product Line Adoption Strategies*. PhD thesis, Universidade Federal de Pernambuco.
- [Alves et al. 2006] Alves, V., Neto, A. C., Soares, S., Santos, G., Calheiros, F., Nepomuceno, V., Pires, D., Leal, J., e Borba., P. (2006). From conditional compilation to aspects: A case study in software product lines migration. In *First Workshop on Aspect-Oriented Product Line Engineering (AOPLE)*, Portland, USA.
- [Apel et al. 2006] Apel, S., Leich, T., e Saake, G. (2006). Aspectual Mixin Layers: Aspects and Features in Concert. In *ICSE '06: Proceedings of IEEE and ACM SIGSOFT 28th International Conference on Software Engineering*, pp. 122–131, Shanghai, China. ACM Press.
- [Baniassad et al. 2006] Baniassad, E., Clements, P. C., Araujo, J., Moreira, A., Rashid, A., e Tekinerdogan, B. (2006). Discovering Early Aspects. *IEEE Softw.*, 23(1):61–70.
- [Batista et al. 2006] Batista, T., Chavez, C., Garcia, A., Kulesza, U., e Lucena, C. (2006). Aspectual Connectors: Supporting the Seamless Integration of Aspects and ADLs. In *SBES '06: Proceedings of Brazilian Symposium on Software Engineering*, pp. 17–32, <http://www.scientificcommons.org/43519059>.

- [Batista et al. 2008] Batista, T. V., Bastarrica, M. C., Soares, S., e da Silva, L. F. (2008). A Marriage of MDD and Early Aspects in Software Product Line Development. In *SPLC '08: InProceedings of the 12th International Software Product Lines Conference*, pp. 97–103.
- [Batory 2000] Batory, D. (2000). Product-line architectures, aspects, and reuse (tutorial session). In *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, p. 832, Limerick, Ireland, New York, NY, USA. ACM.
- [Batory 2003] Batory, D. (2003). A tutorial on feature oriented programming and product-lines. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pp. 753–754, Portland, Oregon, Washington, DC, USA. IEEE Computer Society.
- [Batory et al. 2000] Batory, D., Cardone, R., e Smaragdakis, Y. (2000). Object-oriented framework and product lines. In *Proceedings of the first conference on Software product lines : experience and research directions*, pp. 227–247, Denver, Colorado, United States, Norwell, MA, USA. Kluwer Academic Publishers.
- [Batory e O'Malley 1992] Batory, D. e O'Malley, S. (1992). The design and implementation of hierarchical software systems with reusable components. *ACM Trans. Softw. Eng. Methodol.*, 1(4):355–398.
- [Batory et al. 2003] Batory, D., Sarvela, J. N., e Rauschmayer, A. (2003). Scaling step-wise refinement. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pp. 187–197, Portland, Oregon, Washington, DC, USA. IEEE Computer Society.
- [Bergel et al. 2006] Bergel, A., Hirschfeld, R., Clarke, S., e Costanza, P. (2006). Aspect-boxes - controlling the visibility of aspects. In Filipe, J., Shishkov, B., e Helfert, M. (editores), *ICSOFTE '06: Proceedings of the International Conference on Software and Data Technologies*, pp. 74–83. INSTICC Press.
- [Bockisch et al. 2004] Bockisch, C., Haupt, M., Mezini, M., e Ostermann, K. (2004). Virtual machine support for dynamic join points. In *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*, pp. 83–92, Lancaster, UK, New York, NY, USA. ACM.
- [Boehm 1999] Boehm, B. (1999). Managing Software Productivity and Reuse. *Computer*, 32(9):111–113.
- [Buschmann et al. 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., e Stal, M. (1996). *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons.
- [Cirilo et al. 2008] Cirilo, E., Kulesza, U., e de Lucena, C. P. (2008). A Product Derivation Tool Based on Model-Driven Techniques and Annotations. *Journal of Universal Computer Science*, 14(8):1344–1367.
- [Cirilo et al. 2009] Cirilo, E., Nunes, I., Kulesza, U., e Lucena, C. (2009). Automating the Product Derivation Process of Multi-Agent Systems Product Lines. In *SBES '09: Proceedings of the 23th Brazilian Symposium on Software Engineering*, Fortaleza, Brazil, New York, NY, USA. IEEE Computer Society Press.

- [Colyer et al. 2004] Colyer, A., Rashid, A., e Blair, G. (2004). On the Separation of Concerns in Program Families. Technical Report COMP-001-2004, Lancaster University.
- [Crnkovic 2003] Crnkovic, I. (2003). Component-based software engineering - new challenges in software development. In *ITI '03. Proceedings of the 25th International Conference on Information Technology Interfaces*, pp. 9–18.
- [Czarnecki 2005] Czarnecki, K. (2005). Mapping features to models: A template approach based on superimposed variants. In *GPCE '05: Proceedings of the 4th international Conference on Generative Programming and Component Engineering*, volume 3676 of LNCS, pp. 422–437.
- [Czarnecki e Eisenecker 2000] Czarnecki, K. e Eisenecker, U. W. (2000). *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.
- [Davis 1987] Davis, S. M. (1987). *Future perfect*. Addison-Wesley, Reading, Mass., New York, NY, USA.
- [Díaz et al. 2007] Díaz, O., Trujillo, S., e Pérez, S. (2007). Turning portlets into services: the consumer profile. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pp. 913–922, Banff, Alberta, Canada, New York, NY, USA. ACM.
- [Dijkstra 1997] Dijkstra, E. W. (1997). *A Discipline of Programming*. Prentice Hall PTR, Upper Saddle River, NJ, USA.
- [Forman 1998] Forman, I. R. (1998). Putting Metaclasses to Work. In *TOOLS '98: Proceedings of the Technology of Object-Oriented Languages and Systems*, p. 447.
- [Gacek e Anastasopoulos 2001] Gacek, C. e Anastasopoulos, M. (2001). Implementing product line variabilities. *SIGSOFT Softw. Eng. Notes*, 26(3):109–117.
- [Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.
- [Garlan et al. 1997] Garlan, D., Monroe, R. T., e Wile, D. (1997). ACME: An Architecture Description Interchange Language. In *CASCON '97: Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, pp. 169–183, Toronto, Ontario.
- [Harrison e Ossher 1993] Harrison, W. e Ossher, H. (1993). Subject-oriented programming: a critique of pure objects. *SIGPLAN Not.*, 28(10):411–428.
- [Hirschfeld 2003] Hirschfeld, R. (2003). AspectS - Aspect-Oriented Programming with Squeak. In *NODE '02: Revised Papers from the International Conference NetObject-Days on Objects, Components, Architectures, Services, and Applications for a Networked World*, pp. 216–232, London, UK. Springer-Verlag.
- [Hu et al. 2000] Hu, Y., Merlo, E., Dagenais, M., e Lagüe, B. (2000). C/C++ Conditional Compilation Analysis Using Symbolic Execution. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, p. 196, Washington, DC, USA. IEEE Computer Society.

- [Jouault et al. 2008] Jouault, F., Allilaire, F., Bézivin, J., e Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39.
- [Kang et al. 1990] Kang, K., Cohen, S., Hess, J., Nowak, W., e Peterson, S. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA.
- [Kiczales e Mezini 2005] Kiczales, G. e Mezini, M. (2005). Aspect-oriented programming and modular reasoning. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pp. 49–58, St. Louis, MO, USA, New York, NY, USA. ACM.
- [Kleppe et al. 2003] Kleppe, A. G., Warmer, J., e Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- [Liu et al. 2006] Liu, J., Batory, D., e Lengauer, C. (2006). Feature oriented refactoring of legacy applications. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pp. 112–121, Shanghai, China, New York, NY, USA. ACM.
- [Lopez-herrejon et al. 2005] Lopez-herrejon, R. E., Batory, D., e Cook, W. (2005). Evaluating Support for Features in Advanced Modularization Technologies. In *ECOOP '05: Proceedings of the 19th European Object-Oriented Programming Conference*, pp. 169–194. Springer.
- [Mezini e Ostermann 2004] Mezini, M. e Ostermann, K. (2004). Variability management with feature-oriented programming and aspects. *SIGSOFT Softw. Eng. Notes*, 29(6):127–136.
- [Parnas 1972] Parnas, D. L. (1972). On the Criteria to Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058.
- [Pohl et al. 2007] Pohl, C., Rummler, A., Gasiunas, V., Loughran, N., Arboleda, H., de Alexandria Fernandes, F., Noyé, J., Núñez, A., Passama, R., Royer, J.-C., e Südholt, M. (2007). Survey of Existing Implementation Techniques with respect to their support for the requirements identified in M3.2. Technical Report Specific Targeted Research Project: IST- 33710, AMPLE Consortium.
- [Pohl et al. 2005] Pohl, K., Böckle, G., e van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition.
- [Prehofer 1997] Prehofer, C. (1997). Feature-Oriented Programming: A Fresh Look at Objects. In *ECOOP '97: Proceedings of the 11th European Conference on Object-Oriented Programming*, volume 1241 de *Lecture Notes in Computer Science*, pp. 419–443. Springer-Verlag.
- [Prehofer 2001] Prehofer, C. (2001). Feature-oriented programming: A new way of object composition. *Concurrency and Computation: Practice and Experience*, 13(6):465–501.
- [Pulvermüller et al. 2000] Pulvermüller, E., Speck, A., e Rashid, A. (2000). Collaboration-Based Designs Using Aspect-Oriented Programming. In *TOOLS '00: Proceedings of the Technology of Object-Oriented Languages and Systems*, p. 95, Washington, DC, USA. IEEE Computer Society.

- [Reenskaug et al. 1992] Reenskaug, T., Andersen, E., Berre, A., Hurlen, A., Landmark, A., Lehne, O., Nordhagen, E., Ness-Ulseth, E., Oftedal, G., Skaar, A., e Stenslet, P. (1992). OORASS: Seamless Support for the Creation and Maintenance of Object-Oriented Systems. *J. OO Programming*, 5(6):27–41.
- [Shaw e Garlan 1996] Shaw, M. e Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall.
- [Smaragdakis e Batory 1998] Smaragdakis, Y. e Batory, D. S. (1998). Implementing Layered Designs with Mixin Layers. In *ECOOP '98: Proceedings of the 12th European Conference on Object-Oriented Programming*, pp. 550–570, London, UK. Springer-Verlag.
- [Smaragdakis e Batory 1999] Smaragdakis, Y. e Batory, D. S. (1999). Building Product-Lines with Mixin-Layers. In *ECOOP '99: Proceedings of the Workshop on Object-Oriented Technology*, p. 197, London, UK. Springer-Verlag.
- [Sobreira e Maia 2008] Sobreira, V. e Maia, M. (2008). A Visual Trace Analysis Tool for Understanding Feature Scattering. In *WCRE '08: Proceedings of the 2008 15th Working Conference on Reverse Engineering*, pp. 337–338, Washington, DC, USA. IEEE Computer Society.
- [Tarr et al. 1999] Tarr, P., Ossher, H., Harrison, W., Sutton, S. M., e Jr. (1999). N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, pp. 107–119. ACM.
- [Trujillo et al. 2006] Trujillo, S., Batory, D., e Diaz, O. (2006). Feature refactoring a multi-representation program into a product line. In *GPCE '06: Proceedings of the 5th international conference on Generative programming and component engineering*, pp. 191–200, Portland, Oregon, USA, New York, NY, USA. ACM.
- [Trujillo et al. 2007] Trujillo, S., Batory, D., e Diaz, O. (2007). Feature Oriented Model Driven Development: A Case Study for Portlets. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pp. 44–53, Washington, DC, USA. IEEE Computer Society.
- [Wooldridge e Ciancarini 2001] Wooldridge, M. e Ciancarini, P. (2001). Agent-Oriented Software Engineering: The State of the Art. In *AOSE '00: First international workshop on Agent-oriented Software Engineering*, pp. 1–28. Springer-Verlag.

Apêndice A

Transformações

A.1 $T_{SimplificarModeloEstados}(ME)$

```
MES.Screen = ME.Level01State;  
MES.Screen.name = ME.Level01State.name;  
MES.State = ME.Level02State;  
MES.State.name = ME.Level02State.name;  
MES.Widget = ME.Level03+State;  
MES.Widget.name = ME.Level03+State.name;  
MES.Widget.type = ME.Level03+State.type;  
MES.Widget.Transition = ME.Level03+State.Transition;  
MES.Widget.Transition.name = ME.Level03+State.Transition.name;  
MES.Widget.Transition.source = ME.Level03+State.Transition.source;  
MES.Widget.Transition.target = ME.Level03+State.Transition.target;
```

A.2 $T_{SimplificarModeloDados}(MD)$

```
MDS.Table = MD.Table;  
MDS.Table.name = MD.Table.name;  
MDS.Table.Attribute = MD.Table.Attribute;  
MDS.Table.Attribute.name = MD.Table.Attribute.name;  
MDS.Table.Attribute.type = MD.Table.Attribute.type;  
MDS.Table.Operation = MD.Table.Operation;  
MDS.Table.Operation.name = MD.Table.Operation.name;  
MDS.Table.Operation.type = MD.Table.Operation.type;  
MDS.Table.Operation.Parameter = MD.Table.Operation.Parameter;  
MDS.Table.Operation.Parameter.name = MD.Table.Operation.Parameter.name;  
MDS.Table.Operation.Parameter.type = MD.Table.Operation.Parameter.type;
```

```

MDS.View = MD.View;
MDS.View.name = MD.View.name;
MDS.View.Attribute = MD.View.Attribute;
MDS.View.Attribute.name = MD.View.Attribute.name;
MDS.View.Attribute.type = MD.View.Attribute.type;
MDS.View.Operation = MD.View.Operation;
MDS.View.Operation.name = MD.View.Operation.name;
MDS.View.Operation.type = MD.View.Operation.type;
MDS.View.Operation.Parameter = MD.View.Operation.Parameter;
MDS.View.Operation.Parameter.name = MD.View.Operation.Parameter.name;
MDS.View.Operation.Parameter.type = MD.View.Operation.Parameter.type;

```

A.3 $T_{GerarTemplateMapeamentoEstadosCaracteristicas}$ (MES)

```

TMEC.Screen = MES.Screen;
TMEC.Screen.name = MES.Screen.name;
TMEC.Screen.feature = “ ”;
TMEC.State = MES.State;
TMEC.State.name = MES.State.name;
TMEC.State.feature = “ ”;
TMEC.Widget = MES.Widget;
TMEC.Widget.name = MES.Widget.name;
TMEC.Widget.feature = “ ”;
TMEC.Widget.Transition = MES.Widget.Transition;
TMEC.Widget.Transition.name = MES.Widget.Transition.name;
TMEC.Widget.Transition.source = MES.Widget.Transition.source;
TMEC.Widget.Transition.target = MES.Widget.Transition.target;
TMEC.Widget.Transition.feature = “ ”;

```

A.4 $T_{GerarTemplateMapeamentoDadosCaracteristicas}$ (MDS)

```

TMDC.Table = MDS.Table;
TMDC.Table.name = MDS.Table.name;
TMDC.Table.feature = “ ”;
TMDC.Table.Attribute = MDS.Table.Attribute;
TMDC.Table.Attribute.name = MDS.Table.Attribute.name;
TMDC.Table.Attribute.feature = “ ”;
TMDC.Table.Operation = MDS.Table.Operation;
TMDC.Table.Operation.name = MDS.Table.Operation.name;

```

```

TMDC.Table.Operation.feature = “ ”;
TMDC.View = MDS.View;
TMDC.View.name = MDS.View.name;
TMDC.View.feature = “ ”;
TMDC.View.Attribute = MDS.View.Attribute;
TMDC.View.Attribute.name = MDS.View.Attribute.name;
TMDC.View.Attribute.feature = “ ”;
TMDC.View.Operation = MDS.View.Operation;
TMDC.View.Operation.name = MDS.View.Operation.name;
TMDC.View.Operation.feature = “ ”;

```

A.5 $T_{GerarTemplateMapeamentoEstadosLayout}(MDS)$

```

TMEL.Screen = MES.Screen;
TMEL.Screen.name = MES.Screen.name;
TMEL.Screen.layout = “ ”; TMEL.Widget = MES.Widget;
TMEL.Widget.name = MES.Widget.name;
TMEL.at = “ ”;

```

A.6 Preenchimentos Manuais para Mapeamento de Modelos

As transformações $T_{EspecificarCaracteristicasModeloEstados}(TMEC)$, $T_{EspecificarCaracteristicasModeloDados}(TMDC)$ e $T_{EspecificarLayoutModeloEstados}(TMEL)$ são caracterizadas por preenchimentos manuais nos *templates* para mapeamento entre modelos TMEC, TMDC e TMEL, para a obtenção dos modelos de mapeamento ACME, ACMD e ALME, respectivamente.

A.7 $T_{MapearCaracteristicasModeloEstados}(MES,ACME)$

```

MEAC.Screen = MES.Screen;
MEAC.Screen.name = MES.Screen.name;
MEAC.Screen.feature = ACME.Screen.feature;
MEAC.State = MES.State;
MEAC.State.name = MES.State.name;
MEAC.State.feature = ACME.State.feature;
MEAC.Widget = MES.Widget;

```

```

MEAC.Widget.name = MES.Widget.name;
MEAC.Widget.type = MES.Widget.type;
MEAC.Widget.feature = ACME.Widget.feature;
MEAC.Widget.Transition = MES.Widget.Transition;
MEAC.Widget.Transition.name = MES.Widget.Transition.name;
MEAC.Widget.Transition.source = MES.Widget.Transition.source;
MEAC.Widget.Transition.target = MES.Widget.Transition.target;
MEAC.Widget.Transition.feature = ACME.Widget.Transition.feature;

```

A.8 $T_{MapearCaracteristicasModeloDados}$ (MDS, ACMD)

```

MDAC.Table = MDS.Table;
MDAC.Table.name = MDS.Table.name;
MDAC.Table.feature = ACMD.Table.feature;
MDAC.Table.Attribute = MDS.Table.Attribute;
MDAC.Table.Attribute.name = MDS.Table.Attribute.name;
MDAC.Table.Attribute.type = MDS.Table.Attribute.type;
MDAC.Table.Attribute.feature = ACMD.Table.Attribute.feature;
MDAC.Table.Operation = MDS.Table.Operation;
MDAC.Table.Operation.name = MDS.Table.Operation.name;
MDAC.Table.Operation.type = MDS.Table.Operation.type;
MDAC.Table.Operation.feature = ACMD.Table.Operation.feature;
MDAC.Table.Operation.Parameter = MDS.Table.Operation.Parameter;
MDAC.Table.Operation.Parameter.name = MDS.Table.Operation.Parameter.name;
MDAC.Table.Operation.Parameter.type = MDS.Table.Operation.Parameter.type;
MDAC.View = MDS.View;
MDAC.View.name = MDS.View.name;
MDAC.View.feature = ACMD.View.feature;
MDAC.View.Attribute = MDS.View.Attribute;
MDAC.View.Attribute.name = MDS.View.Attribute.name;
MDAC.View.Attribute.type = MDS.View.Attribute.type;
MDAC.View.Attribute.feature = ACMD.View.Attribute.feature;
MDAC.View.Operation = MDS.View.Operation;
MDAC.View.Operation.name = MDS.View.Operation.name;
MDAC.View.Operation.type = MDS.View.Operation.type;
MDAC.View.Operation.feature = ACMD.View.Operation.feature;
MDAC.View.Operation.Parameter = MDS.View.Operation.Parameter;
MDAC.View.Operation.Parameter.name = MDS.View.Operation.Parameter.name;
MDAC.View.Operation.Parameter.type = MDS.View.Operation.Parameter.type;

```

A.9 *T_{MapearLayoutModeloEstados}(MEAC, ALME)*

```

MEACL.Screen = MEAC.Screen;
MEACL.Screen.name = MEAC.Screen.name;
MEACL.Screen.feature = MEAC.Screen.feature;
MEACL.Screen.layout = ALME.Screen.layout;
MEACL.State = MEAC.State;
MEACL.State.name = MEAC.State.name;
MEACL.State.feature = MEAC.State.feature;
MEACL.Widget = MEAC.Widget;
MEACL.Widget.name = MEAC.Widget.name;
MEACL.Widget.type = MEAC.Widget.type;
MEACL.Widget.feature = MEAC.Widget.feature;
MEACL.Widget.at = ALME.at;
MEACL.Widget.Transition = MEAC.Widget.Transition;
MEACL.Widget.Transition.name = MEAC.Widget.Transition.name;
MEACL.Widget.Transition.source = MEAC.Widget.Transition.source;
MEACL.Widget.Transition.target = MEAC.Widget.Transition.target;
MEACL.Widget.Transition.feature = MEAC.Widget.Transition.feature;

```

A.10 *T_{AnaliseParaProjeto}(MEACL, MDAC)*

Os artefatos arquiteturais de projeto (AAD) são obtidos a partir dos modelos de análise anotados MDAC e MEACL. No estudo de caso utilizado para a avaliação da nossa abordagem, os modelos de análise são convertidos em artefatos que representam as entidades do padrão arquitetural MVC. As transformações a seguir caracterizam a obtenção das entidades Modelo, Visão e Controlador, respectivamente.

A.10.1 *T_{DadosParaMVCModelo}(MEACL)*

```

MVCModel.Model = MDAC.Table;
MVCModel.Model.name = MDAC.Table.name;
MVCModel.Model.feature = MDAC.Table.feature;
MVCModel.Model.Attribute = MDAC.Table.Attribute;
MVCModel.Model.Attribute.name = MDAC.Table.Attribute.name;
MVCModel.Model.Attribute.type = MDAC.Table.Attribute.type;
MVCModel.Model.Attribute.feature = MDAC.Table.Attribute.feature;
MVCModel.Model.Operation = MDAC.Table.Operation;
MVCModel.Model.Operation.name = MDAC.Table.Operation.name;

```

```

MVCMModel.Model.Operation.type = MDAC.Table.Operation.type;
MVCMModel.Model.Operation.Parameter = MDAC.Table.Operation.Parameter;
MVCMModel.Model.Operation.Parameter.name = MDAC.Table.Operation.Parameter.name;
MVCMModel.Model.Operation.Parameter.type = MDAC.Table.Operation.Parameter.type;

MVCMModel.Facade = MDAC.View;
MVCMModel.Facade.name = MDAC.View.name;
MVCMModel.Facade.feature = MDAC.View.feature;
MVCMModel.Facade.Attribute = MDAC.View.Attribute;
MVCMModel.Facade.Attribute.name = MDAC.View.Attribute.name;
MVCMModel.Facade.Attribute.feature = MDAC.View.Attribute.feature;
MVCMModel.Facade.Operation = MDAC.View.Operation;
MVCMModel.Facade.Operation.name = MDAC.View.Operation.name;
MVCMModel.Facade.Operation.type = MDAC.View.Operation.type;
MVCMModel.Facade.Operation.Parameter = MDAC.View.Operation.Parameter;
MVCMModel.Facade.Operation.Parameter.name = MDAC.View.Operation.Parameter.name;
MVCMModel.Facade.Operation.Parameter.type = MDAC.View.Operation.Parameter.type;

```

A.10.2 TEstadosParaVisao(MEACL)

```

MVCView.Screen = MEACL.Screen;
MVCView.Screen.name = MEACL.Screen.name;
MVCView.Screen.feature = MEACL.Screen.feature;
MVCView.Screen.layout = MEACL.Screen.layout;
MVCView.Screen.Widget = MEACL.Screen.Widget;
MVCView.Screen.Widget.name = MEACL.Screen.Widget.name;
MVCView.Screen.Widget.type = MEACL.Screen.Widget.type;
MVCView.Screen.Widget.feature = MEACL.Screen.Widget.feature;
MVCView.Screen.Widget.at = MEACL.Screen.Widget.at;
MVCView.Screen.Widget.Transition = MEACL.Screen.Widget.Transition;
MVCView.Screen.Widget.Transition.name = MEACL.Screen.Widget.Transition.name;
MVCView.Screen.Widget.Transition.source = MEACL.Screen.Widget.Transition.source;
MVCView.Screen.Widget.Transition.target = MEACL.Screen.Widget.Transition.target;
MVCView.Screen.Widget.Transition.feature = MEACL.Screen.Widget.Transition.feature;

```

A.10.3 TEstadosParaControlador(MEACL)

```

MVCController.Screen = MEACL.Screen;
MVCController.Screen.name = MEACL.Screen.name;
MVCController.Screen.feature = MEACL.Screen.feature;

```

```
MVCController.Handler = MEACL.Screen.Widget.Transition;  
MVCController.Handler.name = MEACL.Screen.Widget.Transition.name;  
MVCController.Handler.source = MEACL.Screen.Widget.Transition.source;  
MVCController.Handler.target = MEACL.Screen.Widget.Transition.target;  
MVCController.Handler.feature = MEACL.Screen.Widget.Transition.feature;
```


Apêndice B

Exemplos de Artefatos de Implementação

B.1 Artefatos Originais da Aplicação de Referência

B.1.1 JSP

```
1 <jsp:root xmlns:c="http://java.sun.com/jsp/jstl/core" xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html"
2   xmlns:jsp="http://java.sun.com/JSP/Page" xmlns:ui="http://java.sun.com/blueprints/ui">
3   <jsp:output doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN" doctype-root-element="HTML"
4     doctype-system="http://www.w3.org/TR/html4/loose.dtd" omit-xml-declaration="true"/>
5   <jsp:directive.page contentType="text/html; charset=utf-8"/>
6   <html xmlns="http://www.w3.org/1999/xhtml">
7     <head>
8       <script src="common.js" type="text/javascript"></script>
9       <script src="scroller.js" type="text/javascript"></script>
10      <link href="scroller.css" rel="stylesheet" type="text/css"/>
11      <script src="accordion.js" type="text/javascript"></script>
12      <link href="accordion.css" rel="stylesheet" type="text/css"/>
13      <script src="catalog.js" type="text/javascript"></script>
14    </head>
15    <body>
16      <jsp:include page="banner.jsp"/>
17      <script type="text/javascript">
18        dojo.event.connect(window, "onload", function(){initCatalog();});
19      </script>
20      <center>
21        <table border="0">
22          <tr>
23            <td>
24              <table id="accordion">
25                <tr class="accordionTitle" id="accordionTitle">
26                  <td>Pets</td>
27                </tr>
28                <tr>
29                  <td>
30                    <table border="0" class="accordionBody" id="accordionBody"></table>
31                  </td>
32                </tr>
33              </table>
34            </td>
35            <td>
36              <div id="CatalogBrowser">
37                <table border="0" class="slider" width="500px">
38                  <tr height="415" valign="top">
39                    <td align="center" border="0" id="bodySpace"></td>
40                  </tr>
41                  <tr id="targetRow">
42                    <td align="top" height="70" width="500">
43                      <div class="nav" id="right_button">
44                        
45                      </div>
46                      <div class="nav" id="left_button">
47                        
48                      </div>
49                      <div class="infopane" id="infopane">
50                        <table class="infopaneTable">
51                          <tr>
52                            <td class="infopaneTitle" id="infopaneName"></td>
53                            <td class="infopaneRating" id="infopaneRating">
54                              <f:view>
55                                <ui:rating clearHoverText="Clear Rating" grade="{RatingBean.grade}" hoverTexts="{RatingBean.ratingText}"
56                                  id="rating" includeClear="false" includeNotInterested="false" maxGrade="5"
57                                  notInterestedHoverText="Not Interested"/>

```

```

58         <ui:buyNow amount="100.00" business="donate@animalfoundation.com" id="buyNow1" itemName="Buy Item One"
59             postData="#{PayPalBean.postData}" quantity="1" target="paypal" type="BuyNow"/>
60     </f:view>
61 </td>
62 <td class="infopaneIndicator" id="infopaneIndicator"></td>
63 <td id="infopaneDetailsIcon"></td>
64 </tr>
65 <tr>
66     <td class="infopaneShortDescription" colspan="7" id="infopaneShortDescription"></td>
67 </tr>
68 <tr style="height:20px">
69     <td></td>
70 </tr>
71 <tr>
72     <td class="infopaneDescription" colspan="6" id="infopaneDescription"></td>
73 </tr>
74 </table>
75 </div>
76 </td>
77 </tr>
78 </table>
79 </div>
80 </td>
81 <td style="vertical-align:top; width:300px; text-align:right;">
82     <div id="downloadAds">
83         <jsp:include page="download.jsp"/>
84     </div>
85 </td>
86 </tr>
87 </table>
88 <div id="status" style="text-align:left"></div> <div id="status_2"></div><div id="dstatus"></div><div id="injection_point"></div></center>
89 <jsp:include page="footer.jsp"/>
90 </body>
91 </html>
92 </jsp:root>

```

B.1.2 Javascript

```

1  var ac;
2  var is;
3  var controller;
4
5  function getApplicationContextRoot() {
6      var urlArray=window.location.toString().split("/", 4);
7      return "/" + urlArray[3];
8  }
9
10 var applicationContextRoot=getApplicationContextRoot();
11
12 function initCatalog() {
13     ac = new AccordionMenu();
14     is = new ImageScroller();
15     is.load();
16     controller = new CatalogController();
17     dojo.event.connect("before", bpui.rating, "doClick", controller, "modifyState");
18     controller.initialize();
19 }
20
21 function CatalogController() {
22     dojo.event.topic.subscribe("/catalog", this, handleEvent);
23     var pList = new ProductList();
24     var CHUNK_SIZE=7;
25     var initialRating;
26     var initialItem;
27     var originalURL;
28     var useDOMInjection = false;
29
30     var infoName = document.getElementById("infopaneName");
31     var infoRating = document.getElementById("infopaneRating");
32     var infoPrice = document.getElementById("infopanePrice");
33     var infoBuyNow = document.getElementById("infopaneBuyNow");
34     var infoShortDescription = document.getElementById("infopaneShortDescription");
35     var infoDescription = document.getElementById("infopaneDescription");
36     var buyNowAmount = document.getElementById("buyNow1_amount");
37     var buyNowItemName = document.getElementById("buyNow1_item_name");
38
39     function handleEvent(args) {
40         if (args.type == "showingItem") {
41             if (typeof bpui != 'undefined') {
42                 var groupId = is.getGroupId();
43                 window.location.href= originalURL + "#" + groupId + "," + args.id;
44                 if (typeof bpui.rating != 'undefined') {
45                     bpui.rating.state["rating"].bindings["itemId"]=args.id;
46                     bpui.rating.modifyDisplay("rating", args.rating, true);
47                     showItemDetails(args.id);
48                 } else {
49                     initialItem = args.id;
50                     initialRating = args.rating;
51                 }
52             } else {
53                 initialItem = args.id;
54                 initialRating = args.rating;
55             }
56         } else if (args.type == "getChunk") {
57             populateItems(args.id, args.index, args.currentChunk, false);
58         }
59     }

```

```

59         } else if (args.type == "showItemDetails") {
60             showProductDetails(args.productId, args.itemId);
61
62         } else if (args.type == "showProducts") {
63             is.reset();
64             populateItems(args.productId, 0, 0, true);
65         }
66     }
67
68     function showItemDetails(id) {
69         var i = is.getItems().get(id);
70         setNodeText(infoName, i.name + "<br/><a href='javascript:controller.disableItem(&quot;" + id +
71             "&quot;,&quot;" + i.name + "&quot;)'><font size='-1' color='white'><i>Flag as inappropriate</i></font></a>");
72         setNodeText(infoPrice, i.price);
73         setNodeText(infoShortDescription, i.shortDescription);
74         setNodeText(infoDescription, i.description);
75         buyNowAmount.value = i.price;
76         buyNowItemName.value = i.name;
77     }
78
79     function setNodeText(t, text) {
80         if (useDOMInjection) {
81             t.lastChild.nodeValue = text;
82         } else {
83             t.innerHTML = text;
84         }
85     }
86
87     this.initialize = function() {
88         infoName.innerHTML = "&nbsp;";
89         if (!useDOMInjection && infoName.innerHTML != "&nbsp;") {
90             useDOMInjection = true;
91
92             infoName.appendChild(document.createTextNode("Name"));
93             infoPrice.appendChild(document.createTextNode("$0.00"));
94             infoShortDescription.appendChild(document.createTextNode("<description>"));
95             infoDescription.appendChild(document.createTextNode("<description>"));
96         }
97         var ratingInstance = bpui.rating.state["rating"];
98         ratingInstance.grade = initialRating;
99         bpui.rating.state["rating"].bindings["itemId"]=initialItem;
100         bpui.rating.modifyDisplay("rating", initialRating, true);
101         loadAccordion();
102     }
103
104
105     this.modifyState = function(arg, rating) {
106         var itemId = initialItem;
107         if (typeof bpui.rating.state["rating"].bindings["itemId"] != 'undefined') {
108             itemId = bpui.rating.state["rating"].bindings["itemId"];
109         }
110         is.getItems().get(itemId).rating = rating;
111     }
112
113
114     function loadAccordion () {
115         var bindArgs = {
116             url: applicationContextRoot + "/catalog?command=categories&format=json",
117             mimetype: "text/json",
118             load: function(type, json) {
119                 ac.load(json);
120                 processURLParameters();
121             },
122             error: ajaxBindError
123         };
124         dojo.io.bind(bindArgs);
125     }
126
127     function processURLParameters () {
128         originalURL = decodeURIComponent(window.location.href);
129         var params = {};
130         if (originalURL.indexOf("#") != -1) {
131             var qString = originalURL.split('#')[1];
132             var args = qString.split(',');
133             originalURL = originalURL.split('#')[0];
134             ac.loadCategoryItem(args[0], args[1]);
135             return;
136         } else if (originalURL.indexOf("?") != -1) {
137             var qString = originalURL.split('?')[1];
138             if (qString.indexOf("#") != -1) {
139                 qString = qString.split('#')[0];
140                 originalURL = originalURL.split('#')[0];
141                 window.location.href = originalURL;
142             }
143             ps = qString.split('&');
144             for (var i in ps) {
145                 var t = ps[i].split('=');
146                 params[t[0]] = t[1];
147             }
148             if (typeof params.itemId != 'undefined' && typeof params.pid != 'undefined') {
149                 ac.loadCategoryItem(params.pid, params.itemId);
150             } else if (typeof params.catid != 'undefined') {
151                 ac.showCategory(params.catid);
152             }
153         } else {
154             ac.showFirstCategory();
155         }
156     }
157

```

```

158 function showProductDetails(pid, itemId) {
159     is.reset();
160     is.showProgressIndicator();
161     var bindArgs = {
162         url: applicationContextRoot + "/catalog?command=itemInChunk&pid=" + pid +
163             "&itemId=" + itemId + "&length=" + CHUNK_SIZE,
164         mimeType: "text/xml",
165         load: function(type,data,postProcessHandler) {
166             processProductData(data,true, pid, itemId);
167             showItemDetails(itemId);
168             is.doMaximize();
169         },
170         error: ajaxBindError
171     };
172     dojo.io.bind(bindArgs);
173 }
174
175 function populateItems(pid, index, neededChunk, showImage) {
176     is.showProgressIndicator();
177     is.setGroupId(pid);
178     printDebug("populateItems - need to make sure displaying - pid=" + pid + " Chunk=" + neededChunk);
179     if (!is.containsChunk(pid + "-" + neededChunk)) {
180         if (pList.hasChunk(pid, neededChunk)) {
181             printDebug("**** adding chunk from cache - pid=" + pid + " Chunk=" + neededChunk);
182             is.addChunk(pid + "-" + neededChunk);
183             is.addItem(pList.getChunk(pid, neededChunk));
184             if (showImage && is.getScrollerItems().length > 0) {
185                 is.showImage(is.getScrollerItems()[0].id);
186             }
187         } else {
188             startRetIndex=(neededChunk * CHUNK_SIZE);
189
190             printDebug("**** retrieving chunk from server - pid=" + pid + " currentIndex=" + index +
191                 " startIndex=" + startRetIndex + " Chunk=" + neededChunk);
192             var bindArgs = {
193                 url: applicationContextRoot + "/catalog?command=items&pid=" + pid + "&start=" +
194                     startRetIndex + "&length=" + CHUNK_SIZE,
195                 mimeType: "text/xml",
196                 load: function(type,data,postProcessHandler) {
197                     processProductData(data, showImage, pid, null, neededChunk);
198                 },
199                 error: ajaxBindError
200             };
201             dojo.io.bind(bindArgs);
202         }
203     } else {
204         printDebug("*** items already showing");
205     }
206 }
207
208 function processProductData(responseXML, showImage, pid, iId, chunkId) {
209     var items = [];
210     var count = responseXML.getElementsByTagName("item").length;
211     for (var loop=0; loop < count ; loop++) {
212
213         var item = responseXML.getElementsByTagName("item")[loop];
214         var itemId = item.getElementsByTagName("id")[0].firstChild.nodeValue;
215         var name = item.getElementsByTagName("name")[0].firstChild.nodeValue;
216         var thumbURL = item.getElementsByTagName("image-tb-url")[0].firstChild.nodeValue;
217         var imageURL = item.getElementsByTagName("image-url")[0].firstChild.nodeValue;
218         var description = item.getElementsByTagName("description")[0].firstChild.nodeValue;
219         var price = item.getElementsByTagName("price")[0].firstChild.nodeValue;
220         var rating = item.getElementsByTagName("rating")[0].firstChild.nodeValue;
221         var shortDescription;
222         if (description.length > 71) {
223             shortDescription = description.substring(0,71) + "...";
224         } else {
225             shortDescription = description;
226         }
227         var i = {id: itemId, name: name, image: imageURL, thumbnail: thumbURL,
228             shortDescription: shortDescription, description: description, price:price, rating: rating};
229         items.push(i);
230     }
231
232     pList.addChunk(pid, chunkId, items);
233     is.addItem(items);
234     is.addChunk(pid + "-" + chunkId);
235
236     if (showImage && iId == null) {
237         is.setGroupId(pid);
238         is.showImage(items[0].id);
239     } else {
240         is.setGroupId(pid);
241         is.showImage(iId);
242     }
243     is.hideProgressIndicator();
244 }
245
246 function ProductList() {
247     var _pList = this;
248     var map = new Map();
249
250     this.addChunk = function(pid, chunkNumber, items) {
251         map.put(pid + "-" + chunkNumber, items, true);
252     }
253
254     this.getChunk = function(pid, chunkNumber) {
255         return map.get(pid + "-" + chunkNumber);
256     }

```

```

257
258     this.hasChunk = function(pid, chunkNumber) {
259         return (map.get(pid + "_" + chunkNumber) != null);
260     }
261
262     this.contents = function() {
263         return map.contents();
264     }
265 }
266
267
268 this.disableItem=function(itemId, itemName) {
269     if (confirm("Are you sure you want to effectively remove this item from Petstore?")) {
270         var bindArgs = {
271             url: applicationContextRoot + "/catalog?command=disable&id=" + itemId,
272             mimetype: "text/xml",
273             load: function(type,json) {
274                 pList = new ProductList();
275                 is.reset();
276                 populateItems(is.getGroupId(), 0, 0, true);
277             },
278             error: ajaxBindError
279         };
280         dojo.io.bind(bindArgs);
281     }
282 }
283 }

```

B.1.3 Java (Controlador MVC)

```

1 package com.sun.javaee.blueprints.petstore.controller;
2
3 import com.sun.javaee.blueprints.petstore.controller.actions.CaptchaAction;
4 import com.sun.javaee.blueprints.petstore.controller.actions.CatalogXmlAction;
5 import com.sun.javaee.blueprints.petstore.controller.actions.DefaultControllerAction;
6 import com.sun.javaee.blueprints.petstore.controller.actions.ImageAction;
7 import com.sun.javaee.blueprints.petstore.controller.actions.TagXmlAction;
8 import com.sun.javaee.blueprints.petstore.model.CatalogFacade;
9 import java.io.IOException;
10 import java.io.PrintWriter;
11 import java.util.HashMap;
12 import java.util.logging.Level;
13 import javax.servlet.ServletConfig;
14 import javax.servlet.ServletContext;
15 import javax.servlet.ServletException;
16 import javax.servlet.http.HttpServlet;
17 import javax.servlet.http.HttpServletRequest;
18 import javax.servlet.http.HttpServletResponse;
19 import com.sun.javaee.blueprints.petstore.util.PetstoreUtil;
20
21 public class ControllerServlet extends HttpServlet {
22
23     private static final boolean bDebug=false;
24     private HashMap<String, ControllerAction> actionMap = new HashMap<String, ControllerAction>();
25
26     @Override
27     public void init(ServletConfig config) throws ServletException {
28         super.init(config);
29         ServletContext context = config.getServletContext();
30         CatalogFacade cf = (CatalogFacade) context.getAttribute("CatalogFacade");
31         actionMap.put("/ImageServlet", new ImageAction(context));
32         actionMap.put("/controller", new DefaultControllerAction(context));
33         actionMap.put("/faces/CaptchaServlet", new CaptchaAction());
34         actionMap.put("/TagServlet", new TagXmlAction(cf));
35         actionMap.put("/catalog", new CatalogXmlAction(cf));
36     }
37
38     public ControllerAction findAction(String servletPath) {
39         return actionMap.get(servletPath);
40     }
41
42     @Override
43     public void destroy() {
44         actionMap = null;
45     }
46
47     @Override
48     public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
49         String servletPath = request.getServletPath();
50         if(bDebug) System.out.println(" ServletPath: " + servletPath + ", pathinfo: " + request.getPathInfo());
51         ControllerAction action = actionMap.get(servletPath);
52         if (action != null) {
53             if(bDebug) System.out.println(" Found action " + action.getClass().getName());
54             action.service(request, response);
55         } else {
56             PetstoreUtil.getLogger().log(Level.SEVERE, "Servlet '" + request.getServletPath() + "' not registered in ControllerServlet!!");
57             HttpServletResponse httpResponse=(HttpServletResponse)response;
58             httpResponse.sendError(HttpServletResponse.SC_NOT_FOUND);
59         }
60     }
61 }

```

B.1.4 Java (Modelo MVC)

```

1 package com.sun.javaee.blueprints.petstore.model;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5 import javax.persistence.Table;
6
7 @Entity
8 @Table(name="PRODUCT")
9
10 public class Product implements java.io.Serializable {
11
12     private String productID;
13     private String categoryID;
14     private String name;
15     private String description;
16     private String imageURL;
17
18     public Product( String productID,
19                   String categoryID,
20                   String name,
21                   String description,
22                   String imageURL ) {
23
24         this.productID = productID;
25         this.categoryID = categoryID;
26         this.name = name;
27         this.description = description;
28         this.imageURL = imageURL;
29     }
30
31     public Product() {}
32
33     @Id
34     public String getProductID() {
35         return this.productID;
36     }
37
38     public String getCategoryID() {
39         return categoryID;
40     }
41
42     public String getName() {
43         return name;
44     }
45
46     public String getDescription() {
47         return description;
48     }
49
50     public String getImageURL() {
51         return imageURL;
52     }
53
54     public void setCategoryID( String categoryID ) {
55         this.categoryID = categoryID;
56     }
57
58     public void setProductID( String productID ) {
59         this.productID = productID;
60     }
61
62     public void setName( String name ) {
63         this.name = name;
64     }
65
66     public void setDescription( String description ) {
67         this.description = description;
68     }
69
70     public void setImageURL( String imageURL ) {
71         this.imageURL = imageURL;
72     }
73 }

```

B.2 Artefatos de Geração de Código

B.2.1 *Template* para base JSP (Via A)

```

1 <%@ jet package="cartridge.A.jee.jsf.web" class="JSPBase"
2 imports="cartridge.A.jee.jsf.* java.util.* java.io.* core.mvc.view.* core.fieldtype.* java.nio.channels.*"
3 %>
4 <%
5 Screen s = (Screen)((Object[])argument)[0];
6 Vector<Element> elements = (Vector<Element>)((Object[])argument)[1];
7 Vector<String> allElementTypes = (Vector<String>)((Object[])argument)[2];
8 String repositoryPath = (String)((Object[])argument)[3];
9 String featureTarget = (String)((Object[])argument)[4];
10
11 Vector<String> folders = new Vector<String>();

```

```

12  HashMap<String,String> toCopy = new HashMap<String,String>();
13
14  if (allElementTypes.contains("Captcha")){
15      folders.add(featureTarget + "/Base/src/captcha/");
16      folders.add(featureTarget + "/Base/src/controller/");
17      folders.add(featureTarget + "/Base/src/controller/actions/");
18
19      toCopy.put(repositoryPath + "src/captcha/BlueFilter.java",featureTarget + "/Base/src/captcha/BlueFilter.java");
20      toCopy.put(repositoryPath + "src/captcha/CaptchaSingleton.java",featureTarget + "/Base/src/captcha/CaptchaSingleton.java");
21      toCopy.put(repositoryPath + "src/captcha/RandomString.java",featureTarget + "/Base/src/captcha/RandomString.java");
22      toCopy.put(repositoryPath + "src/captcha/SimpleCaptcha.java",featureTarget + "/Base/src/captcha/SimpleCaptcha.java");
23      toCopy.put(repositoryPath + "src/controller/CaptchaValidateFilter.java",featureTarget + "/Base/src/controller/CaptchaValidateFilter.java");
24      toCopy.put(repositoryPath + "src/controller/actions/CaptchaAction.java",featureTarget + "/Base/src/controller/actions/CaptchaAction.java");
25  }
26
27  if (allElementTypes.contains("MapView")){
28      folders.add(featureTarget + "/Base/src/mapviewer/");
29      folders.add(featureTarget + "/Base/src/proxy/");
30
31      toCopy.put(repositoryPath + "src/mapviewer/MapBean.java",featureTarget + "/Base/src/mapviewer/MapBean.java");
32      toCopy.put(repositoryPath + "src/proxy/GeoCoder.java",featureTarget + "/Base/src/proxy/GeoCoder.java");
33      toCopy.put(repositoryPath + "src/proxy/GeoPoint.java",featureTarget + "/Base/src/proxy/GeoPoint.java");
34  }
35
36  for(String folder : folders){
37      try{
38          File f = new File(folder);
39          f.mkdirs();
40      }
41      catch(Exception e){
42          System.out.println(folder);
43          e.printStackTrace();
44      }
45  }
46
47  for(String st:toCopy.keySet()){
48      File sourceFile = new File(st);
49      File targetFile = new File(toCopy.get(st));
50      FileChannel source = null;
51      FileChannel destination = null;
52
53      try {
54          if(! targetFile.exists()) {
55              targetFile.createNewFile();
56          }
57          JSFArtifactModel.addCopiedFile(targetFile.getPath());
58
59          source = new FileInputStream(sourceFile).getChannel();
60          destination = new FileOutputStream(targetFile).getChannel();
61          destination.transferFrom(source, 0, source.size());
62
63          if(source != null) {
64              source.close();
65          }
66          if(destination != null) {
67              destination.close();
68          }
69      }
70      catch(Exception e){
71          e.printStackTrace();
72      }
73  }
74  %>
75  <?xml version="1.0" encoding="utf-8"?>
76  <document>
77      <top/>
78      <main>
79  <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
80          xmlns:h="http://java.sun.com/jsf/html"
81          xmlns:f="http://java.sun.com/jsf/core"
82          xmlns:ui="http://java.sun.com/blueprints/ui">
83      <jsp:output omit-xml-declaration="true" doctype-root-element="HTML"
84          doctype-system="http://www.w3.org/TR/html4/loose.dtd"
85          doctype-public="-//W3C//DTD HTML 4.01 Transitional/EN" />
86      <jsp:directive.page contentType="text/html;charset=utf-8" />
87      <f:view>
88  <html xmlns="http://www.w3.org/1999/xhtml">
89  <head>
90      <title>:<%s.getName() %>:</title>
91      <script src="<%s.getName().toLowerCase() %>.js" type="text/javascript"></script>
92  </head>
93  <body>
94  <%
95  Vector<String> inserted = new Vector<String>();
96
97  for(String type:allElementTypes){
98      if(! inserted.contains(type)){
99          if(type.compareTo("FishEyeList")==0){
100              inserted.add("FishEyeList");
101          %>
102          <script type="text/javascript" src="<${pageContext.request.contextPath}/faces/static/META-INF/dojo/bpcatalog/dojo.js"></script>
103          <script>dojo.require("dojo.widget.FisheyeList");</script>
104          <%
105              }
106              else if(type.compareTo("RSSFeed")==0){
107                  inserted.add("RSSFeed");
108              %>
109          <script type="text/javascript" src="<${pageContext.request.contextPath}/faces/static/META-INF/rss/rssbar.js"></script>
110          <link type="text/css" rel="stylesheet" href="<${pageContext.request.contextPath}/faces/static/META-INF/rss/rssbar.css" />

```

```

111 <%
112     }
113 }
114 }
115 %>
116 <%
117 HashMap hm = new HashMap<String,String>();
118 hm.put("1Column", "/home/rreisp/workspace/FOMDD/models/domain/WebStores/analysis/layout/1Column.html");
119 hm.put("1ColumnA", "/home/rreisp/workspace/FOMDD/models/domain/WebStores/analysis/layout/1ColumnA.html");
120 hm.put("2Columns", "/home/rreisp/workspace/FOMDD/models/domain/WebStores/analysis/layout/2Columns.html");
121 hm.put("3Columns", "/home/rreisp/workspace/FOMDD/models/domain/WebStores/analysis/layout/3Columns.html");
122 hm.put("3ColumnsA", "/home/rreisp/workspace/FOMDD/models/domain/WebStores/analysis/layout/3ColumnsA.html");
123 hm.put("4Columns", "/home/rreisp/workspace/FOMDD/models/domain/WebStores/analysis/layout/4Columns.html");
124
125 File source = new File((String) hm.get(s.getLayout()));
126
127 ElementTranslator et = new HTMLTranslator();
128 et.setRepository("/home/rreisp/workspace/FOMDD/transformations/repository/domain/WebStores/jsf/");
129 Layout l = new Layout(source,et);
130 l.setElementTranslator(et);
131
132 for(Element e:elements){
133     l.append(e.getAt(),e);
134 }
135 l.generate();
136 %>
137 <%= l.getText() %>
138 </body>
139 </html>
140 </f:view>
141 </jsp:root>
142 </main>
143 <bottom/>
144 </document>

```

B.2.2 *Template* para refinamento JSP (Via A)

```

1 <%@ jet package="cartridge.A.jee.jsf.web" class="JSPRefinement" imports="java.util.Vector core.mvc.view.* core.fieldtype.*"%>
2 <?xml version="1.0" encoding="utf-8"?>
3 <xr:refine xmlns:xr="http://www.atarix.org/xmlRefinement" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4 xsi:schemaLocation="http://www.atarix.org/xmlRefinement" xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/core"
5 xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">
6 <%
7 Vector<Element> els = (Vector<Element>)((Object[])argument)[1];
8 ElementTranslator et = new HTMLTranslator();
9 et.setRepository("/home/rreisp/workspace/FOMDD/transformations/repository/domain/WebStores/jsf/");
10
11 for(Element e:els){
12     if(e.getDepth() > 0){
13         %>
14         <xr:at select="//*[@id='<%=e.getAt() %>']">
15         <xr:append>
16         <%= et.translateBegin(e) + et.translateEnd(e) %>
17         </xr:append>
18         </xr:at>
19         <%
20         }
21         else {
22         %>
23         <xr:at select="//*[@id='<%=e.getAt() %>']">
24         <xr:append>
25         <%= et.translateBegin(e) + et.translateEnd(e) %>
26         </xr:append>
27         </xr:at>
28         <%
29         }
30     }
31     %>
32 </xr:refine>

```

B.2.3 *Template* para base Javascript (Via A)

```

1 <%@ jet package="cartridge.A.jee.jsf.web" class="ScriptBase" imports="java.util.* core.mvc.controller.*" %>
2 <%
3     Screen s = (Screen) ((Object[])argument)[0];
4     Vector<Handler> handlers = (Vector<Handler>)((Object[])argument)[1];
5 %>
6 <?xml version="1.0" encoding="utf-8"?>
7 <scripts>
8 <%
9     for(Handler h:handlers){
10         String name = h.getName();
11
12         String event = "";
13         String guard = "";
14         String action = "";
15
16         if(name.contains("/")){
17             String eventAndGuard = name.substring(0,name.indexOf("/"));
18             if(eventAndGuard.contains("[")){
19                 event = eventAndGuard.substring(0,eventAndGuard.indexOf("["));

```

```

20         guard = eventAndGuard.substring(eventAndGuard.indexOf("/") + 1, eventAndGuard.length() - 1);
21     }
22     else {
23         event = eventAndGuard;
24     }
25     action = name.substring(name.indexOf("/") + 1, name.length());
26 }
27 else {
28     if (name.contains("["){
29         event = name.substring(0, name.indexOf("[");
30         guard = name.substring(name.indexOf("[") + 1, name.length() - 1);
31     }
32     else {
33         event = name;
34     }
35 }
36 String source = h.getSource();
37 String target = h.getTarget();
38
39 if (event.compareTo("") == 0) {
40     event = "onclick";
41 }
42 if (action.length() > 0 && ! action.contains("(")) {
43     action += "(";
44 }
45
46 if (guard.compareTo("") == 0) {
47     if (action.compareTo("") == 0) {
48         if (target.compareTo("ExternalURL") != 0) {
49             String page = target.substring(0, target.indexOf("."));
50 %>
51 <function name="goTo"><%=target %>() ">
52     window.open("<%=page.toLowerCase() %>.jsp");
53 </function>
54 <%
55         }
56     }
57     else {
58 %>
59 <function name="<%=action %> ">
60 </function>
61 <%
62         }
63     }
64     else {
65 %>
66 <function name="<%=action %> ">
67     if (<%=guard %>){
68     }
69 </function>
70 <%
71     }
72 }
73 %>
74 </scripts>

```

B.2.4 Template para refinamento Javascript (Via A)

```

1  <%@ jet package="cartridge.A.jee.jsf.web" class="ScriptRefinement" imports="java.util.* core.mvc.controller.*"%>
2  <?xml version="1.0" encoding="utf-8"?>
3  <%
4      Screen s = (Screen) ((Object[])argument)[0];
5      Vector<Handler> handlers = (Vector<Handler>)((Object[])argument)[1];
6  %>
7  <xr:refine xmlns:xr="http://www.atarix.org/xmlRefinement" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8  xsi:schemaLocation="http://www.atarix.org/xmlRefinement">
9  <%
10      for (Handler h:handlers){
11          String name = h.getName();
12
13          String event = "";
14          String guard = "";
15          String action = "";
16
17          if (name.contains("/"){
18              String eventAndGuard = name.substring(0, name.indexOf("/"));
19              if (eventAndGuard.contains("["){
20                  event = eventAndGuard.substring(0, eventAndGuard.indexOf("[");
21                  guard = eventAndGuard.substring(eventAndGuard.indexOf("/") + 1, eventAndGuard.length() - 1);
22              }
23              else {
24                  event = eventAndGuard;
25              }
26              action = name.substring(name.indexOf("/") + 1, name.length());
27          }
28          else {
29              if (name.contains("["){
30                  event = name.substring(0, name.indexOf("[");
31                  guard = name.substring(name.indexOf("[") + 1, name.length() - 1);
32              }
33              else {
34                  event = name;
35              }
36          }
37          String source = h.getSource();
38          String target = h.getTarget();

```

```

38
39     StringTokenizer st = new StringTokenizer(source, ".");
40
41     int level = st.countTokens();
42     st.nextToken();
43     name = "";
44
45     for(int i=0; i<level-1; i++){
46         name += st.nextToken();
47     }
48
49     if(event.compareTo("") !=0 && guard.compareTo("")==0 && action.compareTo("")==0){
50         // do nothing
51         // this is a simple transition, view should handle the task.
52     }
53     else {
54 %>
55 <xr:at select="//scripts">
56 <xr:append>
57 <%
58         if(event.compareTo("")==0){
59             System.out.println("baga");
60         }
61         else {
62             if(guard.compareTo("")==0){
63                 if(action.compareTo("")==0){
64                     // do nothing
65                     // this is a simple transition, view should handle the task.
66                 }
67                 else {
68 %>
69 <function name="<%=event + name %>" action="<%=action %>" target="<%= target %>" />
70 <%
71                 }
72             }
73             else {
74                 if(action.compareTo("")==0){
75 %>
76 <function name="<%=event + name %>" guard="<%=guard %>" target="<%= target %>" />
77 <%
78                 }
79                 else {
80 %>
81 <function name="<%=event + name %>" guard="<%=guard %>" action="<%=action %>" target="<%= target %>" />
82 <%
83                 }
84             }
85 %>
86 </xr:append>
87 </xr:at>
88 <%
89         }
90     }
91 %>
92 <%
93     }
94 %>
95 </xr:refine>

```

B.2.5 *Template* para base JAK (Modelo MVC - Via A)

```

1 <%@ jet package="cartridge.A.jee.jsf.src.model" class="ModelBeanBase" imports="java.util.Vector core.mvc.model.*" %>
2 <%
3     Model m = (Model)((Object[])argument)[0];
4     Vector<Attribute> atts = (Vector<Attribute>)((Object[])argument)[1];
5     Vector<Operation> ops = (Vector<Operation>)((Object[])argument)[2];
6 %>
7 layer model;
8
9 public class <%=m.getName() %> implements java.io.Serializable {
10
11     <%
12         for(Attribute a: atts){
13 %>
14         private <%=a.getType() %> <%=a.getName() %>;
15     <%
16         }
17 %>
18
19     public <%=m.getName() %>(){ }
20
21     public <%=m.getName() %>(Object[] o){
22         <%=m.getName() %>Constructor(o);
23     }
24     <%
25     if(atts.size()==0){
26 %>
27
28     public <%=m.getName() %>Constructor(Object[] o){
29     <%
30         }
31     else {
32 %>
33
34     public void <%=m.getName() %>Constructor(Object[] o){

```

```

35  <%
36      int i = 0;
37      for(Attribute a: atts){
38  %>
39          this.<%=a.getName() %> = (<%=a.getType() %>) o[<%=i %>];
40  <%
41
42          i++;
43      }
44  %>
45      }
46  <%
47      }
48      for(Attribute a: atts){
49          String attrName = (""+ a.getName().charAt(0)).toUpperCase() + a.getName().substring(1,a.getName().length());
50  %>
51
52          public <%=a.getType() %> get<%=attrName %>(){
53              return this.<%=a.getName() %>;
54          }
55
56          public void set<%=attrName %>(<%=a.getType() %> <%=a.getName() %>){
57              this.<%=a.getName() %> = <%=a.getName() %>;
58          }
59  <%
60      }
61
62      for(Operation o: ops){
63          Vector<Parameter> parameters = o.getParameters();
64  %>
65
66          public <%=o.getType() %> <%=o.getName() %>(<% for(Parameter p: parameters){ %><%=p.getType() + " "+ p.getName() %>
67  <%if(p.getName() != (Parameter) parameters.lastElement().getName()){ %>, <%} %> }%> {
68              }
69  <%
70      }
71  %>
72  }

```

B.2.6 *Template* para refinamento JAK (Modelo MVC - Via A)

```

1  <%@ jet package="cartridge.A.jee.jsf.src.model" class="ModelBeanRefinement" imports="java.util.Vector core.mvc.model.*"%>
2  <%
3      Model m = (Model)((Object[])argument)[0];
4      Vector<Attribute> atts = (Vector<Attribute>)((Object[])argument)[1];
5      Vector<Operation> ops = (Vector<Operation>)((Object[])argument)[2];
6      Vector<Attribute> parameterStack = (Vector<Attribute>)((Object[])argument)[3];
7  %>
8  layer model;
9
10 refines class <%=m.getName() %> {
11
12  <%
13      for(Attribute a: atts){
14          if(a.getFeature().compareTo("")!=0 && a.getFeature() != null){
15  %>
16          private <%=a.getType() %> <%=a.getName() %>;
17  <%
18          }
19      }
20
21      if(att.size())>0){
22  %>
23
24          public void <%=m.getName() %>Constructor(Object[] o){
25  <%
26              int i = parameterStack.size();
27              if(i != 0){
28  %>
29                  Super(Object[]).<%=m.getName() %>Constructor(o);
30  <%
31              }
32
33              for(Attribute a: atts){
34  %>
35                  this.<%=a.getName() %> = (<%=a.getType() %>) o[<%=i %>];
36  <%
37                  parameterStack.add(a);
38                  i++;
39              }
40  %>
41          }
42  <%
43          }
44
45          for(Attribute a: atts){
46              String attrName = (""+ a.getName().charAt(0)).toUpperCase() + a.getName().substring(1,a.getName().length());
47  %>
48
49              public <%=a.getType() %> get<%=attrName %>(){ return this.<%=a.getName() %>; }
50
51              public void set<%=attrName %>(<%=a.getType() %> <%=a.getName() %>){
52                  this.<%=a.getName() %> = <%=a.getName() %>;
53          }

```

```

54  <%
55      }
56
57      for(Operation o: ops){
58          Vector<Parameter> parameters = o.getParameters();
59  %>
60
61      public <%=o.getType() %> <%=o.getName() %>(<% for(Parameter p: parameters){ %><%=p.getType() + " "+ p.getName() %>
62  <%if(p.getName() != ((Parameter) parameters.lastElement()).getName()){ %>, <%} %>}) {
63      }
64  <%
65      }
66  %>
67  }

```

B.3 Artefatos de Código Gerados

B.3.1 Base JSP (Via A)

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <document>
3      <top/>
4      <main>
5          <jsp:root xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html"
6  xmlns:jsp="http://java.sun.com/JSP/Page" xmlns:ui="http://java.sun.com/blueprints/ui">
7              <jsp:output doctype-public="-//W3C/DTD HTML 4.01 Transitional/EN" doctype-root-element="HTML"
8  doctype-system="http://www.w3.org/TR/html4/loose.dtd" omit-xml-declaration="true"/>
9              <jsp:directive.page contentType="text/html; charset=utf-8"/>
10             <f:view>
11                 <html xmlns="http://www.w3.org/1999/xhtml">
12                     <head>
13                         <title>:Catalog: </title>
14                         <script src="catalog.js" type="text/javascript"></script>
15                     </head>
16                     <body>
17                         <script src="${pageContext.request.contextPath}/faces/static/META-INF/rss/rssbar.js" type="text/javascript"></script>
18                         <link href="${pageContext.request.contextPath}/faces/static/META-INF/rss/rssbar.css" rel="stylesheet" type="text/css"/>
19                         <div id="all">
20                             <div id="header">
21                                 <table>
22                                     <tr>
23                                         <td id="header_top_1"></td>
24                                         <td id="header_top_2"></td>
25                                         <td id="header_top_3"></td>
26                                     </tr>
27                                     <tr>
28                                         <td colspan="3" id="header_bottom"></td>
29                                     </tr>
30                                 </table>
31                             </div>
32                             <div id="main">
33                                 <div id="column1"></div>
34                                 <div id="column2">
35                                     <div id="content_1"></div>
36                                     <div id="content_2"></div>
37                                     <div id="content_3"></div>
38                                 </div>
39                                 <div id="column3"></div>
40                             </div>
41                             <div id="footer"></div>
42                         </div>
43                     </body>
44                 </html>
45             </f:view>
46         </jsp:root>
47     </main>
48     <bottom/>
49 </document>

```

B.3.2 Refinamento JSP (Via A)

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <xr:refine version="2.0" xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html"
3  xmlns:jsp="http://java.sun.com/JSP/Page" xmlns:xr="http://www.atarix.org/xmlRefinement"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.atarix.org/xmlRefinement">
5      <xr:at select="/*[id='column1']">
6          <xr:append>
7              <ul id="CatalogDefaultAccordionMenu"></ul>
8          </xr:append>
9      </xr:at>
10 </xr:refine>

```

B.3.3 Composição final JSP (Via A)

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <document>
3    <top/>
4    <main>
5      <jsp:root xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/core"
6        xmlns:jsp="http://java.sun.com/JSP/Page"
7        xmlns:ui="http://java.sun.com/blueprints/ui">
8        <jsp:output doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"
9          doctype-root-element="HTML"
10          doctype-system="http://www.w3.org/TR/html4/loose.dtd"
11          omit-xml-declaration="true"/>
12        <jsp:directive.page contentType="text/html; charset=utf-8"/>
13        <f:view>
14          <html xmlns="http://www.w3.org/1999/xhtml">
15            <head>
16              <title>:Catalog:</title>
17              <script src="catalog.js" type="text/javascript"/>
18            </head>
19            <body>
20              <script src="{pageContext.request.contextPath}/faces/static/META-INF/rss/rssbar.js"
21                type="text/javascript"/>
22              <link href="{pageContext.request.contextPath}/faces/static/META-INF/rss/rssbar.css"
23                rel="stylesheet"
24                type="text/css"/>
25              <div id="all">
26                <div id="header">
27                  <table>
28                    <tr>
29                      <td id="header_top_1">
30                        <a href="#" id="CatalogDefaultHeaderLink" onclick="goToCatalogDefault;">
31                          <img border="" src=""/>
32                        </a>
33                      </td>
34                      <td id="header_top_2">
35                        <p id="CatalogDefaultHeaderTitle">LABEL</p>
36                      </td>
37                      <td id="header_top_3">
38                        <ul id="CatalogDefaultMainMenu"/>
39                      </td>
40                    </tr>
41                    <tr>
42                      <td colspan="3" id="header_bottom">
43                        <table id="CatalogDefaultRSSChannel"/>
44                      </td>
45                    </tr>
46                  </table>
47                </div>
48                <div id="main">
49                  <div id="column1">
50                    <ul id="CatalogDefaultAccordionMenu"/>
51                  </div>
52                  <div id="column2">
53                    <div id="content_1">
54                      <img border="" src=""/>
55                    </div>
56                    <div id="content_2">
57                      <div id="CatalogDefaultInfoPanel"/>
58                    </div>
59                    <div id="content_3">
60                      <div id="CatalogDefaultSubCategoryItemSlider"/>
61                    </div>
62                  </div>
63                  <div id="column3">
64                    <ul id="CatalogDefaultAdvertisePanel1"/>
65                  </div>
66                </div>
67                <div id="footer">
68                  <p id="CatalogDefaultFooter">FREE TEXT</p>
69                </div>
70              </div>
71            </body>
72          </html>
73        </f:view>
74      </jsp:root>
75    </main>
76    <bottom/>
77  </document>

```

B.3.4 Base Javascript (Via A)

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <scripts>
3  </scripts>

```

B.3.5 Refinamento Javascript (Via A)

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <xr:refine xmlns:xr="http://www.atarix.org/xmlRefinement" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://www.atarix.org/xmlRefinement">

```

```

4 <xr:at select="//scripts">
5 <xr:append>
6 <function name="onMouseOverDefaultAccordionMenuItemStateItemCategory" action="expandCategory(categoryId)" target="Catalog.Default"/>
7 </xr:append>
8 </xr:at>
9 <xr:at select="//scripts">
10 <xr:append>
11 <function name="onClickDefaultAccordionMenuItemStateSubCategoriesSubCategory" action="loadSubCategory(subCategoryId)" target="Catalog.Default"/>
12 </xr:append>
13 </xr:at>
14 <xr:at select="//scripts">
15 <xr:append>
16 <function name="onClickDefaultSubCategoryItemSliderStateProductThumb" action="loadProduct(productId)" target="Catalog.Default"/>
17 </xr:append>
18 </xr:at>
19 </xr:refine>

```

B.3.6 Base JAK (Modelo MVC - Via A)

```

1 layer model;
2
3 public class ShopFacadeFacade implements java.io.Serializable {
4
5     private Object shopFacadeUserTransaction;
6     private Object shopFacadeEmf;
7
8     public ShopFacadeFacade() {}
9
10    public ShopFacadeFacade( Object[] o ){
11        ShopFacadeConstructor( o );
12    }
13
14    public void ShopFacadeFacadeConstructor( Object[] o ) {
15        this.shopFacadeUserTransaction = ( Object ) o[0];
16        this.shopFacadeEmf = ( Object ) o[1];
17    }
18
19    public Object getShopFacadeUserTransaction() {
20        return this.shopFacadeUserTransaction;
21    }
22
23    public void setShopFacadeUserTransaction( Object shopFacadeUserTransaction ) {
24        this.shopFacadeUserTransaction = shopFacadeUserTransaction;
25    }
26
27    public Object getShopFacadeEmf() {
28        return this.shopFacadeEmf;
29    }
30
31    public void setShopFacadeEmf( Object shopFacadeEmf ) {
32        this.shopFacadeEmf = shopFacadeEmf;
33    }
34
35    public Set shopFacadeGetProducts()
36    {
37    }
38
39    public Set shopFacadeGetItemInChunkVLH( Integer itemId, Integer productId, Integer chunkSize )
40    {
41    }
42
43    public Set shopFacadeGetItemsVLH( Integer productId, Integer start, Integer chunkSize )
44    {
45    }
46
47    public Set shopFacadeGetItemsByItemID( Set itemsId )
48    {
49    }
50
51    public Set shopFacadeGetItemsByItemIDByRadius( Set itemsId, Double fromLat, Double toLat, Double fromLong, Double toLong )
52    {
53    }
54
55    public Set shopFacadeGetItemsByCategoryVLH( Integer categoryId, Integer start, Integer chunkSize )
56    {
57    }
58
59    public Set shopFacadeGetZipCodeLocations( Integer cityId, Integer start, Integer chunkSize )
60    {
61    }
62
63    public Set shopFacadeGetProducts( Integer categoryId )
64    {
65    }
66
67    public Set shopFacadeGetItems( Integer productId )
68    {
69    }
70
71    public Item shopFacadeGetItem( Integer itemId )
72    {
73    }
74
75    public String shopFacadeAddItem( Item item )
76    {

```

```

77     }
78
79     public void shopFacadeUpdateItem( Item item )
80     {
81     }
82
83     public void shopFacadeIndexItem( Object indexDoc )
84     {
85     }
86 }

```

B.3.7 Refinamento JAK (Modelo MVC - Via A)

```

1  layer model;
2
3  refines class ShopFacadeFacade {
4
5      public Set shopFacadeGetCategories ()
6      {
7      }
8
9      public Set shopFacadeGetAllItemsFromCategory( Integer categoryId )
10     {
11     }
12
13     public Set shopFacadeGetItemsByCategoryByRadiusVLH( Integer categoryId, Integer start,
14 Integer chunkSize, Double fromLat, Double toLat, Double fromLong, Double toLong )
15     {
16     }
17
18     public Category shopFacadeGetCategory( Integer categoryId )
19     {
20     }
21 }

```

B.3.8 Composição final JAK (Modelo MVC - Via A)

```

1  layer model;
2
3  public class ShopFacadeFacade implements java.io.Serializable {
4
5      private Object shopFacadeUserTransaction;
6      private Object shopFacadeEmf;
7
8      public ShopFacadeFacade(){}
9
10     public ShopFacadeFacade( Object[] o ){
11         ShopFacadeConstructor( o );
12     }
13
14     public void ShopFacadeFacadeConstructor( Object[] o ) {
15         this.shopFacadeUserTransaction = ( Object ) o[0];
16         this.shopFacadeEmf = ( Object ) o[1];
17     }
18
19     public Object getShopFacadeUserTransaction() {
20         return this.shopFacadeUserTransaction;
21     }
22
23     public void setShopFacadeUserTransaction( Object shopFacadeUserTransaction ) {
24         this.shopFacadeUserTransaction = shopFacadeUserTransaction;
25     }
26
27     public Object getShopFacadeEmf() {
28         return this.shopFacadeEmf;
29     }
30
31     public void setShopFacadeEmf( Object shopFacadeEmf ) {
32         this.shopFacadeEmf = shopFacadeEmf;
33     }
34
35     public Set shopFacadeGetProducts ()
36     {
37     }
38
39     public Set shopFacadeGetItemInChunkVLH( Integer itemId, Integer productId, Integer chunkSize )
40     {
41     }
42
43     public Set shopFacadeGetItemsVLH( Integer productId, Integer start, Integer chunkSize )
44     {
45     }
46
47     public Set shopFacadeGetItemsByItemID( Set itemsId )
48     {
49     }
50
51     public Set shopFacadeGetItemsByItemIDByRadius( Set itemsId, Double fromLat, Double toLat, Double fromLong, Double toLong )
52     {
53     }
54 }

```

```
55     public Set shopFacadeGetItemsByCategoryVLH( Integer categoryId, Integer start, Integer chunkSize )
56     {
57     }
58
59     public Set shopFacadeGetZipCodeLocations( Integer cityId, Integer start, Integer chunkSize )
60     {
61     }
62
63     public Set shopFacadeGetProducts( Integer categoryId )
64     {
65     }
66
67     public Set shopFacadeGetItems( Integer productId )
68     {
69     }
70
71     public Item shopFacadeGetItem( Integer itemId )
72     {
73     }
74
75     public String shopFacadeAddItem( Item item )
76     {
77     }
78
79     public void shopFacadeUpdateItem( Item item )
80     {
81     }
82
83     public void shopFacadeIndexItem( Object indexDoc )
84     {
85     }
86
87     public Set shopFacadeGetCategories()
88     {
89     }
90
91     public Set shopFacadeGetAllItemsFromCategory( Integer categoryId )
92     {
93     }
94
95     public Set shopFacadeGetItemsByCategoryByRadiusVLH( Integer categoryId, Integer start,
96 Integer chunkSize, Double fromLat, Double toLat, Double fromLong, Double toLong )
97     {
98     }
99
100    public Category shopFacadeGetCategory( Integer categoryId )
101    {
102    }
103
104    public Set shopFacadeDoSearch( String queryString )
105    {
106    }
107
108    public void shopFacadeAddTagsToItemId( String sxTags, Integer itemId )
109    {
110    }
111
112    public Tag shopFacadeAddTag( String sxTag )
113    {
114    }
115
116    public Set shopFacadeGetTagsInChunk( Integer start, Integer chunkSize )
117    {
118    }
119
120    public Tag shopFacadeGetTag( String sxTag )
121    {
122    }
123 }
```