Geração de aplicações para linhas de produtos orientadas a aspectos com apoio da ferramenta Captor-AO

Carlos Alberto de Freitas Pereira Júnior

Geração de aplicações para linhas de produtos orientadas a aspectos com apoio da ferramenta Captor-AO

Carlos Alberto de Freitas Pereira Júnior

Orientadora: ***Prof. Dra. Rosana Teresinha Vaccare Braga***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC/USP como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Outubro/2008

Agradecimentos

Esta dissertação é resultado de muito esforço e dedicação, e não poderia deixar de registrar aqui minha sincera gratidão a todos que estiveram ao meu lado durante o desenvolvimento deste trabalho.

Primeiramente gostaria de agradecer a Deus por ter me dado esta oportunidade e também saúde, coragem, persistência e sabedoria para poder chegar a mais esta conquista.

À Profa. Dra. Rosana Braga, professora e orientadora, pelo seu permanente apoio e atenção dispensada no decorrer deste trabalho. À sua disponibilidade irrestrita, sua forma amiga, exigente e crítica, fundamental contribuição no meu crescimento enquanto pesquisador.

Ao Edison, criador do Captor, que respondeu prontamente os meus questionamentos e dúvidas.

À FAPESP, Fundação de Amparo à Pesquisa do Estado de São Paulo, que me concedeu uma bolsa durante a realização deste mestrado, auxílio financeiro que contribuiu para viabilização deste trabalho.

Agradeço a todos professores do mestrado, amigos e funcionários da Universidade de São Paulo, pelo auxílio e atenção ao longo destes meses.

Obrigado aos colegas do LabES e de mestrado, em especial aos amigos: André Maluquinho, Endo, Érika, Falcão, Ivan, KLB, Marcão, Otávio Chefe, Paula , Pelúcio, Stanley, Tânia, Valter, Vanessa e Vasco, pelas amizades então demonstradas, que, com uma perfeita mistura de dever e descontração, me ajudaram a cumprir minhas obrigações acadêmicas.

Aos meus pais, pela sólida formação que me foi dada, pelo incentivo e apoio incondicional que sempre me ajudaram em minhas conquistas. Às minhas irmãs, pelo amor, admiração e respeito.

À minha namorada, amiga e companheira, Cristina, pelo incansável apoio durante o desenvolvimento deste trabalho, por sua paciência e compreensão reveladas, fundamental nesta trajetória.

Por fim, deixo aqui minha sincera gratidão a todas as pessoas que, direta ou indiretamente, contribuíram para a concretização deste trabalho.

Resumo

Uma Linha de Produtos de Software (LPS) consiste de um conjunto de sistemas de software que compartilham características comuns e satisfazem às necessidades específicas de um segmento particular. Para tornar o processo de instanciação de produtos mais rápido e menos suscetível a erros, o projeto de uma LPS pode adotar a utilização de geradores de aplicação, que podem gerar os artefatos da LPS utilizando uma especificação das variabilidades de um certo produto. Adicionalmente, nota-se que determinadas características transversais de uma linha de produtos têm potencial de reúso em diferentes domínios, podendo ser implementadas usando a programação orientada a aspectos (POA). Neste trabalho é proposto um processo para o desenvolvimento de LPS e geração automatizada de produtos levando em consideração os interesses transversais existentes em cada domínio de aplicação. Os interesses transversais são as características comuns espalhadas pelas divisões ou módulos do programa de diferentes domínios. O processo aqui proposto tem a finalidade de aumentar o reúso de características de linhas de produtos por meio da POA, permitindo que as LPS's sejam projetadas de forma mais coesa e, consequentemente, facilitando sua manutenção e evolução. Visando diminuir o esforço necessário para a instanciação dos produtos provenientes dessas linhas de produtos, neste trabalho também é apresentada uma extensão do gerador Captor, denominada Captor-AO. Esse gerador fornece suporte ao processo proposto, permitindo a criação de produtos formados por características de diferentes domínios. Por fim, é apresentado um estudo de caso em que é realizada a configuração de um domínio transversal para o interesse de persistência, a definição de um domínio-base compatível com esse domínio transversal e a geração de produtos formados pelas características de ambos os domínios utilizando o gerador estendido Captor-AO.

Abstract

A Software Product Line (SPL) consists of a set of software systems that share common features and fulfill the specific requirements of a particular domain. In order to make the products instantiation process faster and less prone to errors, the project of a SPL can adopt the utilization of application generators, which can can automatically generate the SPL artifacts based on the specification of the variabilities of a particular product. Additionally, it can be noticed that certain crosscutting features of a product line have potencial to be reused in different domains, so they can be implemented using aspect oriented programming (AOP). In this work, a process is proposed for the development of SPLs and automatic generation of products, considering the crosscutting concerns present in each application domain. The crosscutting concerns are related to the common features that are scattered around program divisions or modules of different domains. The process proposed here has the goal of enhancing the reuse of SPL features using AOP, allowing the design of SPL in a more cohesive way and, thus, easing its maintenance and evolution. Aiming at decreasing the effort needed to instantiate products from these SPL, this work also presents an extension to the Captor application generator, named Captor-AO. This generator supports the proposed process, allowing the creation of products composed by features of different domains. Finally, a case study is presented in which Captor-AO is configured with two domains: a crosscutting domain for the persistence concern and a base domain compatible with this crosscutting domain, such that the generation of products can be done by composing features of both domains.

Sumário

Resumo **i**

Abstract **iii**

1 Introdução **1**
1.1 Contextualização 1
1.2 Motivação 2
1.3 Objetivos 3
1.4 Organização do Trabalho 4

2 Reúso de Software **5**
2.1 Considerações Iniciais 5
2.2 Reúso de Software 6
2.3 Padrões e Linguagens de Padrões 7
2.4 Frameworks de Software Orientados a Objetos 7
2.5 Linha de Produtos de Software 8
2.5.1 Modelagem de Variabilidades em Linhas de Produtos de Software 10
2.5.2 *Product Line Practice* 10
2.5.3 FAST 11
2.5.4 Método PLUS 13
2.5.5 Programação Orientada a Features 15
2.6 Geradores de Aplicações 17
2.6.1 Tipos de Transformações 18
2.6.2 Arquiteturas de Geradores de Aplicação 18
2.7 Considerações Finais 20

3 Linhas de Produtos e Aspectos **21**
3.1 Considerações Iniciais 21
3.2 Programação Orientada a Aspectos 22
3.3 Linguagem AspectJ 24
3.3.1 Adendos 25
3.3.2 Conjunto de Pontos de Junção 25
3.3.3 Declarações inter-tipos 25
3.3.4 Processo de Combinação 25
3.4 Abordagens Ad-hoc 26

3.5 Abordagens Sistemáticas . . . 27
3.6 Abordagens baseadas no Desenvolvimento Incremental . . . 28
3.6.1 Modelo Caesar . . . 28
3.6.2 Modelo AHEAD . . . 29
3.6.3 Método de Pacios . . . 31
3.7 Ferramentas . . . 32
3.8 Considerações Finais . . . 34

4 Gerador de Aplicações Configurável Captor **37**
4.1 Considerações Iniciais . . . 37
4.2 Utilização de Geradores Configuráveis em Linhas de Produtos . . . 38
4.3 Ferramenta Captor . . . 39
4.4 Engenharia de Domínio . . . 39
4.5 Engenharia de Aplicações . . . 41
4.6 Exemplo de Uso: LPS GestorPsi . . . 43
4.6.1 Análise e Implementação do Domínio . . . 44
4.6.2 Configuração do Domínio . . . 46
4.6.3 Geração de Produtos . . . 49
4.7 Considerações finais . . . 51

5 Desenvolvimento de LPS apoiado por geradores de aplicações e aspectos **53**
5.1 Considerações Iniciais . . . 53
5.2 Linhas de Produtos Orientadas a Aspectos e Domínios Transversais . . . 54
5.2.1 Pontos de Junção Abstratos . . . 55
5.2.2 Pontos de Junção Pré-Definidos . . . 55
5.2.3 Combinação entre Domínios . . . 56
5.2.4 Variabilidades Funcionais e Variabilidades de Junção . . . 56
5.2.5 Conjuntos de Extensão . . . 57
5.3 Processo de desenvolvimento de LPS para Domínios-Base e Domínios Transversais 57
5.3.1 Engenharia de Domínio . . . 58
5.3.2 Engenharia de Aplicações . . . 62
5.4 Considerações Finais . . . 64

6 Captor-AO **65**
6.1 Considerações Iniciais . . . 65
6.2 Captor-AO: Extensão do gerador de aplicações Captor . . . 65
6.3 Análise de Requisitos . . . 67
6.4 Projeto . . . 70
6.5 Implementação . . . 72
6.5.1 Gerenciamento de Domínios . . . 74
6.5.2 Definição da LMA no Gerador Captor-AO . . . 74
6.5.3 Extensão da Linguagem de Gabaritos . . . 76
6.5.4 Especificação Transversal . . . 80
6.5.5 Geração de Produtos . . . 81
6.6 Considerações Finais . . . 85

7 Estudo de Caso **87**
7.1 Considerações Iniciais . . . 87
7.2 Domínio Transversal de Persistência . . . 87
7.2.1 Análise e Implementação do Domínio . . . 88
7.2.2 Criação do projeto do domínio transversal . . . 91
7.2.3 Identificação dos Pontos de Junção Abstratos . . . 92
7.2.4 Criação da linguagem de modelagem de aplicações . . . 93
7.2.5 Implementação dos Gabaritos . . . 96
7.2.6 Mapeamento entre LMA e Gabaritos . . . 98
7.3 Domínio-Base de Bóias Náuticas . . . 99
7.3.1 Informação do Domínio Compatível e Definição de PJP's . . . 100
7.3.2 Inclusão dos Conjuntos de Extensão . . . 101
7.4 Geração de Produtos . . . 102
7.5 Considerações Finais . . . 105

8 Conclusão **107**
8.1 Considerações Iniciais . . . 107
8.2 Contribuições . . . 108
8.3 Trabalhos Futuros . . . 109

A Manual de Instalação do Gerador Captor-AO **119**
A.1 Executando o Arquivo Binário . . . 119
A.2 Obtendo o Código-Fonte . . . 119

Lista de Figuras

2.1 Três níveis de reúso (Lajoie e Keller, 1994) 6
2.2 As três atividades essenciais para linhas de produtos de software (SEI, 2008) 11
2.3 Evolutionary Software Product Line Engineering Process (ESPLEP) (Gomaa, 2004) 14
2.4 Atividades do processo ESPLEP (Gomaa, 2004) 15
2.5 Pilha de *mixin layers* (Apel et al., 2005c) 16
2.6 Modelo para um gerador de aplicações (Cleaveland, 1988) 17
2.7 Transformações vertical, horizontal e oblíqua (Czarnecki e Eisenercker, 2002) . . . 19

3.1 Implementação de *tracing* OO convencional com espalhamento e entrelaçamento de código. (Eler, 2006) 23
3.2 Implementação de *tracing* com aspectos. (Eler, 2006) 24
3.3 Implementação da feature de logging usando Aspectual Mixins Layers (Apel et al., 2005a) 30
3.4 Dados estatísticos do estudo de caso (Apel e Batory, 2006) 31
3.5 Abordagem orientada a aspectos para desenvolvimento de linhas de produtos de software (Pacios, 2006) 32
3.6 Arquitetura da ferramenta GenArch (Cirilo et al., 2007) 34

4.1 Configuração e utilização de geradores configuráveis (Shimabukuro, 2006) 38
4.2 Arquitetura do Captor (Shimabukuro et al., 2006) 40
4.3 Interface utilizada para configuração de um novo domínio 41
4.4 Interface utilizada para a criação de um novo projeto no gerador Captor 42
4.5 Interface utilizada durante a especificação de uma nova aplicação 43
4.6 Modelo de Features do domínio de Clínicas de Psicologia (Pacios, 2006) 45
4.7 Árvore de formulários para o domínio de clínicas de psicologia 47
4.8 Interface do Captor utilizada para a criação de formulários 48
4.9 Configuração de regras de validação 48
4.10 Exemplo de gabarito do domínio de clínicas de psicologia 49
4.11 Interface utilizada para instanciação de membros do domínio de clínicas de psicologia 50
4.12 Janela principal do produto gerado pelo Captor 50

5.1 Relacionamento entre domínios-base e domínios transversais 55
5.2 Diagrama de atividades da engenharia de domínio 58
5.3 Diagrama de atividades da engenharia de domínios-base 59
5.4 Exemplo de Compatibilidade entre domínios-base e domínios transversais 60

5.5 Inclusão do domínio-base C . . . 60
5.6 Diagrama de atividades da engenharia de domínios transversais . . . 61
5.7 Inclusão do domínio transversal T4 . . . 62
5.8 Diagrama de atividades da engenharia de aplicações . . . 63
5.9 Diagrama de atividades da engenharia de aplicações com combinação de domínios 63

6.1 Processo de geração de aplicações com utilização de múltiplos domínios . . . 66
6.2 Diagrama de casos de uso da ferramenta Captor-AO . . . 69
6.3 Geração de aplicações individuais e composição manual de pontos de junção (Captor versão original) . . . 69
6.4 Instanciação de produtos com combinação automática de features base e transversais (Captor-AO) . . . 70
6.5 Modelo conceitual do sistema Captor-AO . . . 71
6.6 Arquitetura do Captor-AO . . . 72
6.7 Diagrama de classes de projeto do gerador Captor-AO . . . 73
6.8 Classes de gerenciamento de domínios . . . 75
6.9 Interface do Captor-AO para configuração de um novo domínio . . . 76
6.10 Modelagem de Pontos de Junção Abstratos . . . 76
6.11 Ponto de Junção Abstrato com lista de valores . . . 77
6.12 Exemplo de gabarito com utilização de PJA's . . . 77
6.13 Gabarito contendo a declaração de um conjunto de extensão . . . 79
6.14 Hierarquia da especificação transversal . . . 80
6.15 Exemplo de especificação transversal . . . 82
6.16 Classes utilizadas para combinação de domínios . . . 83
6.17 Interface utilizada para o preenchimento de variabilidades de diferentes domínios . 83
6.18 Árvore de diretórios para os arquivos gerados . . . 84

7.1 Estrutura do framework de persistência (Camargo, 2006) . . . 89
7.2 Exemplo de instanciação do aspecto `PersistentEntities` . . . 90
7.3 Criação do projeto para o domínio de persistência de dados . . . 92
7.4 Escolhendo o tipo do domínio . . . 92
7.5 Árvore de formulários do domínio de persistência . . . 94
7.6 Interface utilizada para criação da LMA do domínio de persistência . . . 95
7.7 Elemento de formulário da variabilidade Banco de Dados . . . 95
7.8 Elemento de formulário do PJA: "AbrirConexão" . . . 96
7.9 Elemento de formulário do PJA: "ClassesPersistentes" . . . 96
7.10 Caixa de seleção utilizada para escolha do tipo de consciência . . . 96
7.11 Gabarito da classe `MyODBCConnection` . . . 97
7.12 Gabarito da classe `MyODBCConnection` . . . 98
7.13 Mapeamento MTL para os gabaritos do domínio de persistência . . . 99
7.14 Instalação do domínio de persistência no gerador Captor-AO . . . 100
7.15 Especificação transversal do domínio FWS . . . 101
7.16 Conjunto de extensão do domínio FWS . . . 102
7.17 Definição dos domínios do novo produto . . . 103
7.18 Interface utilizada durante a engenharia de aplicações do estudo de caso . . . 103
7.19 Progresso da geração . . . 104
7.20 Compilação do produto utilizando o ambiente Eclipse . . . 105

A.1 Interface do gerador Captor-AO . . . 120

A.2 Compilando o gerador Captor-AO 120

Lista de Tabelas

4.1 Descrição das features implementadas para a linha de produtos 46

6.1 Requisitos do gerador estendido Captor-AO 68
6.2 Elementos de uma especificação transversal 81

7.1 Operações do framework de persistência 88
7.2 Pontos de Junção Abstratos do domínio de persistência 93
7.3 Variabilidades funcionais do domínio de persistência 93
7.4 Variabilidades de junção do domínio de persistência 94

CAPÍTULO

1

Introdução

1.1 Contextualização

O crescimento da demanda e da complexidade dos sistemas de software nas últimas décadas contribuiu para o surgimento de vários problemas inerentes ao desenvolvimento de software, tais como: dificuldades do software ficar pronto a tempo; aumento exponencial da dificuldade de desenvolvimento em decorrência do aumento da complexidade; a qualidade do processo e do produto ficam abaixo do esperado; a manutenção e evolução do software tornam-se muito difíceis; e, principalmente, o software não evolui como esperado pelo mercado, não acompanhando a evolução de outras engenharias, como o hardware (Pressman, 2002). Nesse cenário, pode-se notar a importância da aplicação de técnicas de reúso de software no processo de desenvolvimento.

Pode-se definir reúso de software como a reutilização de qualquer tipo de conhecimento sobre um sistema em outros sistemas similares, com o objetivo de reduzir o esforço de desenvolvimento e a manutenção nesses novos sistemas (Biggerstaff e Perlis, 1989). Várias técnicas para reúso de software têm sido propostas na literatura, com o objetivo de aproveitar esforços despendidos quando se iniciam novos desenvolvimentos. Dentre essas técnicas pode-se citar orientação a objetos, componentes, frameworks, linguagens de padrões, geradores de aplicação e linhas de produtos.

Uma Linha de Produtos de Software (LPS) consiste de um conjunto de sistemas de software que compartilham características comuns e que satisfazem às necessidades específicas de um segmento particular. Esses sistemas são desenvolvidos a partir de um conjunto comum de artefatos (Clements e Northrop, 2001) e são diferenciados entre si em termos das suas características variantes (Parnas, 1979). A utilização de LPS se torna mais vantajosa quando diversos sistemas são

produzidos em série e em quantidade. Nesse sentido, o projeto de uma LPS pode adotar a utilização de geradores de aplicação, tornando o processo de instanciação mais rápido e menos suscetível a erros.

Um gerador de aplicação é uma ferramenta que aceita como entrada uma especificação de uma tarefa ou problema a ser solucionado e gera automaticamente os artefatos necessários para solucionar o problema (Czarnecki e Eisenercker, 2002). Com o uso de geradores, os sistemas de uma LPS podem ser criados rapidamente sem a necessidade de codificação manual, exceto para os requisitos não previstos pelo gerador. Embora as ferramentas de automação facilitem o trabalho que deve ser realizado, a construção dessas ferramentas é uma tarefa complexa e que também requer um grande esforço para o seu desenvolvimento (Braga, 2003; Smaragdakis e Batory, 2000).

Visando diminuir o esforço necessário para o desenvolvimento de geradores de aplicações, no trabalho de Shimabukuro (2006) é apresentado um estudo sobre geradores de aplicações configuráveis. Um gerador de aplicação configurável é um gerador de aplicação que pode ser configurado para realizar a geração de artefatos em domínios diferentes. Com essa abordagem, para realizar a geração de artefatos, o engenheiro troca o processo de desenvolvimento de geradores de aplicação por um processo de configuração de um gerador de aplicação configurável.

Diversos artigos enfatizam a dificuldade de representar e implementar variabilidades de uma LPS (Bachmann et al., 2003; Becker e Kaiserslautern, 2003; Bosch et al., 2002; Anastasopoulos e Gacek, 2001). Características variantes são difíceis de implementar pois elas podem estar espalhadas e entrelaçadas em diversas unidades de decomposição, como classes e componentes, dependendo da tecnologia utilizada. Nesse sentido, têm surgido pesquisas sobre a separação de interesses de LPS utilizando a Programação Orientada a Aspectos (POA) (Anastasopoulos e Muthig, 2004; Lee et al., 2006; Heo e Choi, 2006; Mezini e Ostermann, 2004; Apel e Batory, 2006; Lohmann et al., 2006; Lesaint e Papamargaritis, 2004; Camargo, 2006; Silva, 2004; Kulesza et al., 2007).

A separação de interesses (*concerns*) é um princípio fundamental que cuida das limitações da cognição humana ao lidar com um ambiente complexo. Esse princípio defende que, para contornar a complexidade, deve-se resolver uma questão importante (ou interesse) por vez (Dijkstra, 1976). Interesses transversais (do inglês, *crosscutting concerns*) são interesses cujo código espalha-se pelas divisões ou módulos do programa. A Programação Orientada a Aspectos é uma metodologia de programação que promove a separação avançada de interesses ao introduzir uma nova unidade modular, chamada aspecto (Kiczales et al., 1997), criada para prover uma separação mais eficiente dos interesses transversais contidos em um software.

1.2 Motivação

Por ser um tema de pesquisa muito recente, os pesquisadores primeiramente se preocuparam em estabelecer os conceitos e desafios da integração entre as tecnologias de linhas de produtos e

aspectos, bem como os processos e ferramentas que oferecessem mecanismos adequados para a implementação dos interesses transversais existentes em linhas de produtos. Trabalhos nessa linha já estão em andamento e alguns resultados estão disponíveis (Anastasopoulos e Muthig, 2004; Camargo, 2006; Kulesza et al., 2007; Cirilo et al., 2007).

Percebe-se a necessidade de criação de métodos para apoiar o desenvolvimento de linhas de produtos baseadas em aspectos, denominadas nesta pesquisa como linhas de produtos orientadas a aspectos. Como há na literatura poucas propostas de métodos para o desenvolvimento de LPS e geração de produtos que explorem as vantagens da POA, há muito espaço para pesquisas e experimentos. É nesse contexto que este trabalho está inserido.

Um método para o desenvolvimento de linhas de produtos que leve em consideração os interesses transversais existentes em cada domínio[1], permitiria que as LPS's fossem projetadas de forma mais legível, ou seja, sem haver interesses transversais espalhados e entrelaçados pelo sistema. Isso possibilitaria a diminuição do acoplamento entre as classes e o aumento da coesão, facilitando, conseqüentemente, sua manutenção, evolução e reúso.

1.3 Objetivos

O objetivo geral desta dissertação é investigar como projetar linhas de produtos que considerem a tecnologia de desenvolvimento com aspectos. Neste trabalho são mostrados estudos realizados com a intenção de beneficiar o desenvolvimento de linhas de produtos utilizando as vantagens inerentes da POA. Somado a isso, também existe o objetivo de investigar como generalizar os interesses transversais de uma LPS de tal forma que eles possam ser reusados em outras linhas de produtos.

O objetivo específico é apresentar um processo de desenvolvimento de LPS e geração de produtos, baseado na composição de interesses transversais provenientes de diferentes domínios. Para tal, são propostas novas atividades nos processos de engenharia de domínios e engenharia de aplicações visando o reúso dos interesses transversais de uma ou mais LPS's.

Outro objetivo específico é estender o gerador de aplicações configurável Captor com a utilização de aspectos. Essa extensão, denominada Captor-AO, possibilita a geração de produtos sobre combinações entre *features*[2] de diferentes domínios.

Adicionalmente, é um objetivo deste trabalho registrar a forma de condução do processo proposto, o que é feito por meio de um estudo de caso em que é realizada a demonstração prática desse processo utilizando o gerador Captor-AO. Esse estudo consiste no projeto de duas linhas de produtos, uma do domínio de bóias náuticas e outra do domínio de persistência, e na geração de produtos formados pelas features de ambas as linhas de produtos.

[1] Neste trabalho, o termo domínio será utilizado para denotar uma área de conhecimento ou atividade caracterizada por um conjunto de conceitos e terminologia compreendidos pelos participantes dessa área (Booch et al., 2000).

[2] O termo *feature* não será traduzido neste trabalho pois a expressão correspondente em português, característica, não expressa tecnicamente o que o conceito significa

1.4 Organização do Trabalho

No Capítulo 2 é apresentada uma revisão bibliográfica sobre reúso de software. A noção de reúso é definida e são apresentadas e discutidas diferentes abordagens, técnicas e processos de reúso. É apresentada também uma introdução sobre os conceitos de linguagens de padrões, frameworks de software orientados a objetos, linhas de produtos e geradores de aplicação.

No Capítulo 3 são abordados conceitos relativos à programação orientada a aspectos e também são apresentadas pesquisas que propõem a integração entre esses conceitos e a engenharia de linha de produtos. Trabalhos relevantes publicados sobre esse assunto são revisados e discutidos.

No Capítulo 4 é apresentado o gerador de aplicação configurável Captor. Esse gerador foi criado durante a pesquisa de mestrado de Shimabukuro (2006) e foi utilizado como base para o desenvolvimento da ferramenta Captor-AO. Nesse Capítulo são detalhados a sua arquitetura e o seu processo de configuração e utilização.

No Capítulo 5 é descrito um processo de desenvolvimento da LPS apoiado por geradores de aplicações e aspectos. Nesse Capítulo são apresentandas em detalhes as atividades realizadas durante as etapas de engenharia de domínio e engenharia de aplicações. Essas atividades podem variar de acordo com os interesses transversais do domínio e são baseadas nas atividades propostas no trabalho de Shimabukuro (2006).

No Capítulo 6 é apresentado, em detalhes, o gerador estendido Captor-AO, no qual foi implementado o processo de configuração de domínios seguindo a Programação Orientada a Aspectos. A apresentação dessa ferramenta é realizada de forma gradual, de acordo com os passos do ciclo de desenvolvimento, desde a criação dos casos de uso, passando pela análise de requisitos, até as etapas de projeto e implementação.

No Capítulo 7 é descrito um estudo de caso em que o gerador Captor-AO é configurado para dois domínios diferentes: um domínio de bóias náuticas, descrito anteriormente por Weiss e Lai (1999) e um domínio para persistência de dados adaptado a partir do framework de persistência desenvolvido por Camargo (2006). Também é a realizada nesse estudo de caso a geração de produtos formados pela combinação de features desses dois domínios.

No Capítulo 8 são apresentadas as conclusões deste trabalho, enfatizando as suas principais contribuições e apresentando propostas de trabalhos futuros em continuidade ao que foi realizado.

Por fim, no Apêndice A é incluída uma documentação adicional do gerador Captor-AO, que esclarece os passos para a instalação da ferramenta. Também são mostrados os passos necessários para a obtenção do código-fonte e compilação das bibliotecas.

CAPÍTULO

2

Reúso de Software

2.1 Considerações Iniciais

Com o surgimento de uma crescente demanda por sistemas de software cada vez maiores e mais complexos, e a necessidade de construir e evoluir esses sistemas em prazos curtos sem perda de qualidade, tornou-se necessária a utilização de tecnologias que promovam o reúso de artefatos existentes e facilitem a evolução de um software. Essas tecnologias podem visar o reúso de artefatos como documentação, artefatos de análise e projeto, implementação, conhecimento do domínio, estruturas arquiteturais, experiência de desenvolvimento, requisitos, casos de teste e código-fonte.

Neste capítulo são descritas as abordagens de reúso de software relacionadas com a proposta apresentadas no capítulo 1 e é organizado da seguinte forma: a Seção 2.2 apresenta a definição formal para o conceito de Reúso de Software e as diferentes escalas de reúso. A Seção 2.3 trata dos Padrões e das Linguagens de Padrões. A Seção 2.4 discute sobre Frameworks de Software Orientados a Objetos. Na seção 2.5 é apresentada a Engenharia de Linhas de Produtos de Software, juntamente com alguns dos principais processos de desenvolvimento de linhas de produtos. Na Seção 2.6 é mostrada a definição de Geradores de Aplicações e uma classificação para geradores de acordo com o tipo de arquitetura adotada. Finalmente, na Seção 2.7 são apresentadas as considerações finais sobre este capítulo.

2.2 Reúso de Software

A engenharia de software vem evoluindo e refinando técnicas e métodos para o desenvolvimento de aplicações[1] confiáveis, flexíveis, de fácil manutenção e com custo e prazos de desenvolvimento viáveis. Um dos fatores possibilita atingir essas metas é o emprego de reúso no processo de desenvolvimento.

O reúso tem como objetivo principal reaproveitar artefatos de software que já tenham sido desenvolvidos, testados e implantados anteriormente com sucesso, visando uma diminuição considerável no tempo de desenvolvimento e uma facilidade maior de depuração, que, por conseqüência, desencadeiam os demais objetivos a serem alcançados. Segundo Krueger (1992), o reúso de software "é o processo de criação de sistemas de software a partir de software existente, ao invés de construir sistemas de software a partir do zero".

A Figura 2.1 apresenta uma classificação de níveis de reutilização e abstração. No reúso em baixa escala, são tratados elementos de baixo nível, como classes, métodos e/ou fragmentos de código. No reúso em média escala, são envolvidas micro-arquiteturas[2] isoladamente, assim como as semi-aplicações denominadas frameworks. No mais alto nível, o reúso em alta escala, é considerado o reúso de aplicações, que são subsistemas independentes e que podem ser reutilizados com poucas modificações ou extensões.

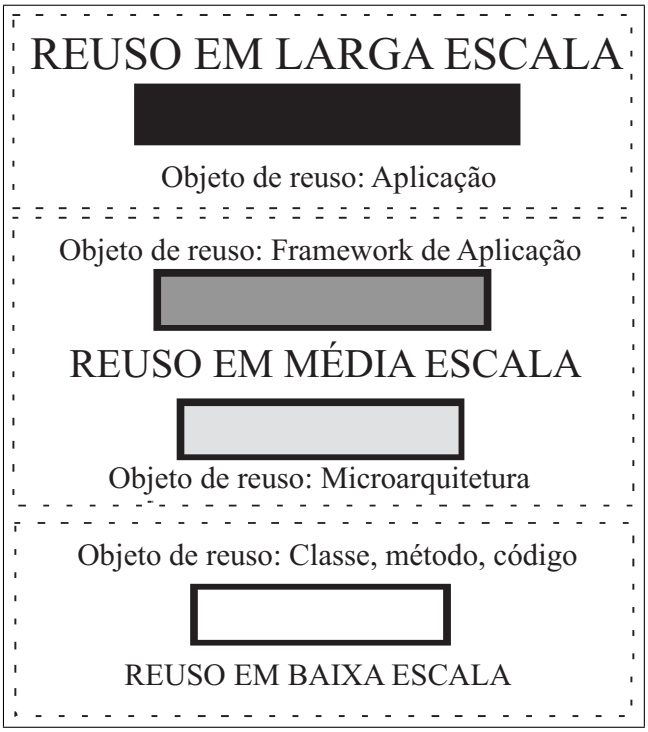

Figura 2.1: Três níveis de reúso (Lajoie e Keller, 1994)

[1] O termo aplicação é utilizado neste trabalho para descrever um sistema de software ou parte de um sistema de software.

[2] Micro-arquiteturas: termo apresentado por Erich Gamma (Gamma et al., 1995), que representa estruturas de classes e suas interações.

2.3 Padrões e Linguagens de Padrões

Padrões de software descrevem soluções para problemas que ocorrem com freqüência no desenvolvimento de software, com o intuito de captar a experiência adquirida pelos desenvolvedores durante anos de prática profissional (Gamma et al., 1995). Podem ser vistos como uma forma de nomear uma técnica e descrever seus custos e benefícios. A idéia é que os padrões descrevam soluções recorrentes que passaram pelo teste do tempo (Fayad et al., 1999).

Além de fornecerem exemplos a serem seguidos e artifícios a serem copiados e posteriormente refinados ou estendidos, os padrões garantem uniformidade na estrutura do software, aumentando a produtividade no seu desenvolvimento e manutenção. Adicionalmente, eles viabilizam o reúso do conhecimento obtido por projetistas experientes, ajudando um novato a agir como um especialista (Gamma et al., 1995). O processo de desenvolvimento de software orientado a objetos pode ser facilitado pelo uso de padrões de projeto. Eles proporcionam um vocabulário comum para a comunicação entre projetistas, criando abstrações em um nível superior ao de classes e instâncias.

Pode-se reunir padrões em diversas formas tais como: coleções, catálogos, sistemas e linguagens. Em particular, uma linguagem de padrões é uma coleção estruturada de padrões que se apóiam uns nos outros para transformar requisitos e restrições em uma arquitetura (Coplien, 1998). Seus padrões constituintes cobrem todos os aspectos importantes de um dado domínio e pelo menos um padrão deve estar disponível para cada aspecto da construção e implementação de um sistema de software: não podem haver “vazios” ou “brancos”. Uma linguagem de padrões representa a seqüência temporal de decisões que levam ao projeto completo de uma aplicação, de forma a tornar-se um método para guiar o processo de desenvolvimento (Brugali et al., 2000). O gerador de aplicações Captor, alvo deste trabalho, pode ser configurado por meio de linguagens de padrões, conforme foi apresentado no trabalho de Shimabukuro (2006). As linguagens de padrões podem ser implementadas com frameworks.

2.4 Frameworks de Software Orientados a Objetos

Um framework de software orientado a objetos tem o propósito de facilitar o reúso, não só o aproveitamento de trechos de código de programas, mas também a reutilização de esforços dispensados em todas as fases do desenvolvimento de software, desde a análise dos requisitos, passando pelo projeto do software, implementação e testes.

Do ponto vista do propósito do framework, pode-se considerá-lo como o esqueleto de uma aplicação, que pode ser instanciado por um desenvolvedor de aplicações (Johnson, 1997). Trata-se de uma aplicação semi-completa reutilizável que, quando especializada, produz aplicações personalizadas (Johnson e Foote, 1988). Pode ainda ser visto como um conjunto de blocos de software pré-fabricados que programadores podem usar, estender e adaptar para soluções computacionais específicas (Taligent, 2008). Frameworks fazem com que desenvolvedores não precisem começar

do zero sempre que constroem uma nova aplicação. Do ponto de vista da estrutura interna, um framework pode ser definido como um conjunto de classes que contêm o projeto abstrato de soluções para uma família de problemas relacionados, propiciando o reúso com granularidade maior do que classes (Johnson e Foote, 1988).

Um aspecto variável de um domínio[3] de aplicação é chamado de "ponto variável" (em inglês, *hot-spot*) (Buschmann, 1996). Diferentes aplicações de um mesmo domínio são distinguidas por um ou mais pontos variáveis. Eles representam as partes do framework de aplicação que são específicas de sistemas individuais. Os pontos variáveis são projetados para serem genéricos e podem ser adaptados às necessidades da aplicação. Em contrapartida, os "Pontos fixos" (em inglês, *frozen-spots*) definem a arquitetura geral de um sistema de software - seus componentes básicos e os relacionamentos entre eles. Os pontos fixos são usados sem nenhuma modificação em todas as instanciações de um framework de aplicação.

O comportamento específico de um framework de aplicação é geralmente definido adicionando-se métodos às subclasses de uma ou mais de suas classes. Assim, existem três tipos de framework: caixa branca (*white box*), caixa cinza (*gray box*) e caixa preta (*black box*) (Yassin e Fayad, 2000). No framework caixa branca o reúso ocorre por herança, ou seja, o usuário deve criar subclasses das classes abstratas contidas no framework para produzir aplicações específicas, devendo, para tal, entender detalhes de como o framework funciona. No framework caixa preta, o reúso é por composição, ou seja, o usuário combina diversas classes concretas existentes no framework para obter a aplicação desejada, devendo apenas conhecer a interface para poder usá-lo. Já no framework caixa cinza, há uma mistura entre os frameworks caixa branca e caixa preta. Portanto, o reúso é obtido por meio de herança, por ligação dinâmica e também pelas interfaces de definição.

A presença de frameworks influencia o processo de desenvolvimento de software. Por exemplo, desenvolver um sistema financeiro tendo em mãos um framework para construção de aplicações financeiras é diferente de desenvolver o mesmo sistema partindo do zero. O desenvolvimento de software baseado em frameworks possui três fases (Bosch et al., 1999): a fase de desenvolvimento, na qual o framework é construído e testado; a fase de uso, na qual o framework é instanciado inúmeras vezes para aplicações específicas; e a fase de evolução e manutenção, na qual o framework evolui para incorporar novas funcionalidades do domínio. A instanciação de frameworks pode ser feita manualmente ou com apoio de geradores de aplicações, como o Captor (Ver Capítulo 4).

2.5 Linha de Produtos de Software

Uma linha de produtos é um conjunto de produtos que compartilham conjuntos de requisitos em comum, mas ao mesmo tempo exibem variabilidade significativa nos requisitos (Griss, 2000). Uma arquitetura do sistema é desenvolvida, de modo a expandir as necessidades de todo o conjunto

[3]Neste trabalho, o termo domínio será utilizado para denotar uma área de conhecimento ou atividade caracterizada por um conjunto de conceitos e terminologia compreendidos pelos participantes dessa área (Booch et al., 2000).

de produtos e fornecer um contexto no qual outros recursos, tais como código e testes, podem ser desenvolvidos com flexibilidade suficiente para satisfazer os produtos da linha.

Uma organização pode utilizar métodos voltados para o desenvolvimento de uma família de produtos, ou seja, um conjunto de itens que possuem aspectos comuns e variabilidades previsíveis (Weiss e Lai, 1999). Um membro de uma família pode ser um produto completo ou pode ser apenas parte de um produto maior.

Ao contrário dos métodos convencionais de engenharia de software que desenvolvem seus produtos de forma manual como uma arte, a engenharia de software de linhas de produtos se preocupa com a industrialização do processo de desenvolvimento. Essa industrialização inclui o aprendizado da configuração e da montagem de componentes para a produção de produtos similares, integração e automação do processo de produção, desenvolvimento de ferramentas que configuram e automatizam tarefas repetitivas, melhoramento das relações entre clientes e fornecedores para a redução de riscos, automação da produção de variantes dos produtos, distribuição da produção entre os diferentes fornecedores e padronização de processos (Greenfield e Short, 2004; Weiss e Lai, 1999).

Vários artefatos são disponibilizados para implementação das possíveis funcionalidades de um produto da linha. Um determinado produto é composto utilizando-se artefatos com diversos tipos de funcionalidades. As diferentes escolhas de artefatos podem fazer com que dois produtos tenham funcionalidades diferentes ou as mesmas funcionalidades com características diferentes. A confiabilidade é aumentada porque as partes reusadas já foram usadas em sistemas anteriores. Custo e tempo de desenvolvimento são diminuídos, já que o esforço de desenvolvimento para qualquer produto sozinho é largamente distribuído para vários recursos reusáveis, cujo custo é distribuído a todos os membros da linha de produtos. Com essas técnicas, uma organização pode atingir níveis mais elevados de produtividade e construir sistemas sob demanda de forma mais eficaz (Shimabukuro, 2006).

Muitas vantagens são obtidas ao utilizar linhas de produtos. Artefatos comuns são reusados muitas vezes e, além disso, reparos e correções de defeitos em um produto podem rapidamente ser propagados para outros membros da linha de produtos, pelo uso de artefatos compartilhados (Griss, 2000).

Segundo Griss (2001) as principais vantagens relacionadas ao uso de linhas de produto são:

- Redução no custo de desenvolvimento de um produto;
- Redução da mão de obra de desenvolvimento de software (dos projetos de uma família);
- Redução no tempo de entrega (*time-to-market*);
- Aumento na qualidade dos sistemas desenvolvidos.

2.5.1 Modelagem de Variabilidades em Linhas de Produtos de Software

A modelagem de similaridades e de variabilidades é uma atividade essencial em linhas de produtos de software. No trabalho de Jacobson et al. (1997) é apresentado o conceito de pontos de variação como uma forma de modelar a variabilidade em casos de uso e em artefatos da UML. Segundo Gomaa (2004), a abordagem amplamente difundida pela comunidade científica para modelagem da variabilidade de requisitos de software é o modelo de *features*.

Features são um conceito importante em linhas de produtos do software pois indicam requisitos ou características reusáveis de uma linha de produtos. Um requisito é usado para representar uma necessidade da linha de produtos de software, e portanto, ao menos um membro da linha de produtos deve ser capaz de satisfazê-lo. Uma vez que a LPS capturou esse requisito, ele é identificado como uma feature fornecida pela LPS (Gomaa, 2004).

O método FODA (*Fast Oriented Domain Analysis*) (Kang et al., 1990) utiliza features, que são organizadas em uma árvore. A árvore de features possui uma hierarquia de composição em que as features dos níveis mais altos da árvore são compostas pelas features dos níveis inferiores. Embora o modelo de features auxilie na identificação dos pontos variáveis da linha de produtos, ele por si só não fornece diretivas suficientes para as etapas de projeto e implemantação da linha de produtos. A seguir são apresentados alguns métodos e processos voltados para o desenvolvimento de linhas de produtos.

2.5.2 *Product Line Practice*

Desenvolvido pelo *Software Engineering Institute*[4] (SEI), o processo *Product Line Practice* (PLP) (SEI, 2008) envolve atividades relacionadas ao desenvolvimento de artefatos centrais e desenvolvimento/gerenciamento de produtos utilizando esses artefatos centrais. Para a execução dessas tarefas é necessária a definição de áreas de trabalho mais gerenciáveis, com conjuntos menores de atividades. Cada área de trabalho possui um plano de trabalho e métricas associadas que permitem acompanhar sua execução e avaliar o sucesso dos trabalhos realizados. Usualmente, essas áreas de trabalho produzem artefatos concretos que levam à criação de artefatos centrais que serão utilizados na linha de produtos.

A abordagem PLP agrupa as áreas de trabalho em três áreas principais:

- Desenvolvimento dos Produtos: necessária para aplicar a tecnologia apropriada à criação e evolução dos produtos;
- Desenvolvimento do Núcleo de Artefatos: relacionada à criação e evolução dos artefatos ou componentes centrais;

[4] O *Software Engineering Institute* (SEI) é sediado na *Carnegie Mellon University* (CMU)

- Gerenciamento: utilizada para o gerenciamento dos esforços relacionados a toda a linha de produtos.

Na Figura 2.2 é ilustrado o relacionamento entre as três atividades essenciais da abordagem da SEI para linhas de produtos.

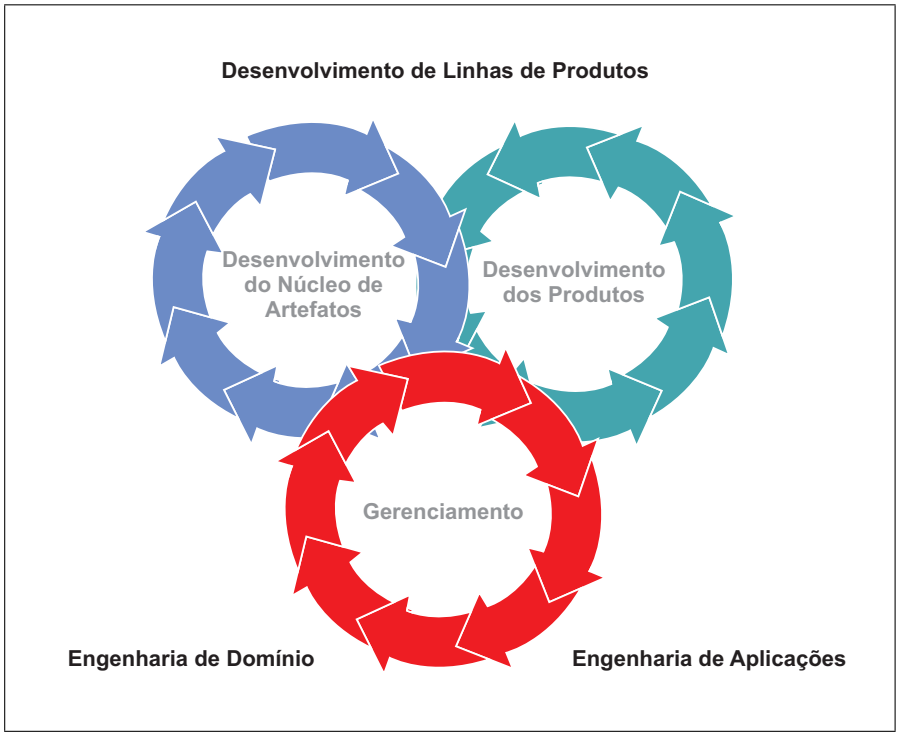

Figura 2.2: As três atividades essenciais para linhas de produtos de software (SEI, 2008)

A PLP está em constante evolução graças às experiências obtidas em diferentes estudos de caso realizados, à colaboração existente entre o SEI e às organizações que estudam a utilização de linha de produtos e as contribuições da comunidade de software.

2.5.3 FAST

A análise de características comuns é um aspecto marcante da utilização do FAST (*Family-oriented Abstraction, Specification and Translation*)(Weiss e Lai, 1999). O FAST é um padrão de processo de produção de software que objetiva resolver os problemas entre uma produção rápida e uma engenharia cuidadosa. Ele está organizado em três sub-processos (Weiss e Lai, 1999):

- Qualificação do Domínio: cujo propósito é identificar a família merecedora de investimento.
- Engenharia de Domínio: cujo propósito é investir em facilidades para produção de membros da família.
- Engenharia de Aplicações: que tem por objetivo fazer uso das facilidades desenvolvidas para uma rápida produção dos membros da família.

Cada um desses subprocessos possui padrões característicos para as atividades. A Qualificação do Domínio consiste de uma análise econômica da família, uma vez que será necessário um alto investimento para sua produção, analisando-se qual família possui um grande número de membros instanciáveis. A Engenharia de Domínio é o processo que exige o maior investimento, pois nessa fase será desenvolvida a arquitetura que facilitará o processo de instanciação dos membros da família. A Engenharia de Aplicações se beneficia dos produtos e artefatos gerados pela Engenharia de Domínio para uma rápida produção dos membros da família.

Uma tarefa importante da engenharia de domínio é a análise de similaridades, que identifica as abstrações comuns a todos os membros da família. A análise de similaridades é usada posteriormente para projetar uma Linguagem de Modelagem de Aplicações, que é utilizada para gerar um membro da família de produtos a partir de uma especificação (Mernik et al., 2005).

Linguagens de Modelagem de Aplicações

Uma Linguagem de Modelagem de Aplicações (LMA) é uma linguagem de alto nível de abstração utilizada para representar aplicações. O ambiente da engenharia de aplicações deve permitir a análise de especificações escritas segundo alguma LMA. Também devem ser fornecidas maneiras de gerar código a partir dessas especificações, ou seja, realizar o mapeamento de uma abstração para uma implementação.

Uma LMA pode ser representada de várias formas, desde um texto relativamente informal, que pode ser processado manualmente, passando por formulários gráficos, até uma linguagem formal com um compilador próprio. A decisão sobre qual tipo de representação escolher depende da natureza do domínio, do ambiente de desenvolvimento e também das habilidades dos colaboradores envolvidos (Beuche, 2003).

O processo FAST possui duas abordagens para a geração de membros da família de produtos a partir de uma LMA: composição e compilação (Weiss e Lai, 1999). A abordagem por composição cria um projeto modular para a família de produtos, em que cada módulo é implementado como um gabarito. Um componente de coposição, do inglês *composer*, é utilizado para gerar membros da família de produtos compondo gabaritos implementados. A especificação da LMA deverá determinar quais gabaritos serão usados para gerar um membro particular da família de produtos (Harsu, 2002). A abordagem por composição é utilizada pelo gerador Captor (Ver Capítulo 4).

A abordagem por compilação requer a construção de um compilador, um parser e um analisador semântico para a LMA, juntamente com um gerador de código. O código é gerado diretamente a partir da especificação LMA. O código gerado pode ser código de alto nível, em nível de máquina, ou qualquer outro. Essa abordagem é utilizada por geradores como o DRACO (Neighbors, 1984).

Uma LMA é projetada e implementada por engenheiros de domínio e usada por engenheiros de aplicação. Entretanto, os engenheiros de aplicação não necessitam ter conhecimento sobre qual abordagem de geração foi utilizada na LMA, nem sobre outras decisões internas que foram tomadas pelos engenheiros de domínio (Weiss e Lai, 1999).

2.5.4 Método PLUS

O método PLUS (do inglês *Product Line UML Based Software Engineering*) (Gomaa, 2004) é um método para projeto de linhas de produtos de software baseado na UML. A ênfase do reúso é dada aos requisitos e arquitetura, pois considera-se que a codificação representa uma porcentagem pequena do custo de um projeto. Assim, as técnicas de linhas de produtos são utilizadas para gerenciar esses artefatos.

No método PLUS é feita distinção entre casos de uso e features. Casos de uso são orientados ao desenvolvimento, pois determinam os requisitos funcionais das linhas de produtos. Features são orientadas ao reúso, pois organizam os resultados de uma análise de similaridades e variabilidades em uma linha de produtos.

O método PLUS é formado pelo processo ESPLEP (do inglês *Evolutionary Software Product Line Engineering Process*). O ESPLEP é um modelo de processo de software que elimina a distinção tradicional entre desenvolvimento e manutenção de software (Gomaa, 2004). Ao invés disso, o software evolui por meio de várias iterações. Assim, os sistemas desenvolvidos são capazes de se adaptar a mudanças nos requisitos durante cada iteração. Para o desenvolvimento de linhas de produtos de software, as atividades de modelagem de requisitos, análise e projeto são substituídas por atividades de desenvolvimento que aumentam a linha de produtos.

O ESPLEP é constituído de duas fases principais:

- Engenharia de Linhas de Produtos: Durante essa fase, as semelhanças e variabilidades na linha de produtos são analisadas do ponto de vista geral dos requisitos. Essa atividade consiste do desenvolvimento de um modelo de casos de uso da linha de produtos, modelo de análise da linha de produtos, arquitetura da linha de produtos e componentes reutilizáveis. Os testes são feitos nos componentes, bem como nas configurações da linha de produtos. Os artefatos produzidos são armazenados em um repositório;

- Engenharia de Aplicações: Durante essa fase, um membro da linha de produtos é desenvolvido. Ao invés de iniciar do zero, é feito uso dos artefatos do repositório. Dados os requisitos da aplicação individual, o modelo de casos de uso da linha de produtos é adaptado para derivar o modelo de caso de uso da aplicação. A arquitetura da linha de produtos é adaptada para derivar a arquitetura da aplicação. Tendo a arquitetura da aplicação e os componentes apropriados do repositório, pode-se instanciar a aplicação desejada.

Na Figura 2.3 é mostrado como as fases de Engenharia de Linhas de Produtos e Engenharia de Aplicações estão relacionadas com o processo ESPLEP.

As atividades principais do processo ESPLEP, conforme exibido na Figura 2.4 (Gomaa, 2004), são:

- Modelagem de requisitos da linha de produtos de software: Um modelo de requisitos consistindo de um modelo de casos de uso e um modelo de features é desenvolvido. O modelo

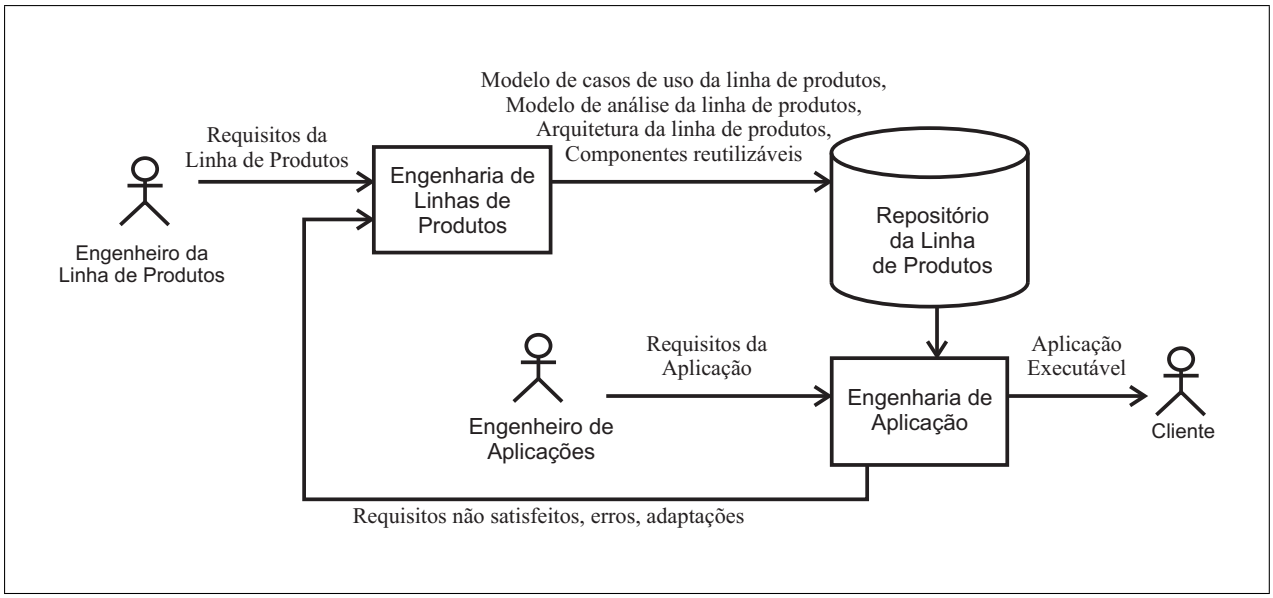

Figura 2.3: Evolutionary Software Product Line Engineering Process (ESPLEP) (Gomaa, 2004)

de casos de uso define os requisitos funcionais em termos de atores e os requisitos não-funcionais. Esse modelo é estendido para indicar as semelhanças e variabilidades da linha de produtos e então os casos de uso são classificados como obrigatórios, opcionais ou alternativos;

- Modelo de análise da linha de produtos de software: São feitos modelos estáticos e dinâmicos. O modelo estático define o relacionamento estrutural entre as classes do domínio. Ele é constituído de um diagrama de classes em que as semelhanças e variabilidades são mostradas com a classificação das classes em obrigatórias, opcionais ou variantes. Devem ser anotadas dependências entre as classes, já que elas podem estar em features diferentes e pode haver dependência entre classes; também é constituído do modelo de casos de uso, que mostra os objetos que participam em cada caso de uso. Os modelos dinâmicos são constituídos por diagramas de estado e diagramas de interação;

- Modelo de projeto da linha de produtos de software: A arquitetura da linha de produtos de software é projetada e o modelo da análise é mapeado para um ambiente operacional;

- Implementação dos componentes de software incremental: Um subconjunto da linha de produtos é selecionado para ser implementado em cada incremento. Deve-se começar com os casos de uso obrigatórios, seguidos pelos opcionais e alternativos. Para cada caso de uso desenvolvido é feito o projeto, implementação e testes;

- Teste da linha de produtos: Consiste de teste de unidade e testes de integração.

No método PLUS as features são classificadas como comuns, opcionais ou alternativas. As features opcionais e alternativas determinam o grau de variabilidade de uma família e, por isso, também são conhecidas como features variantes.

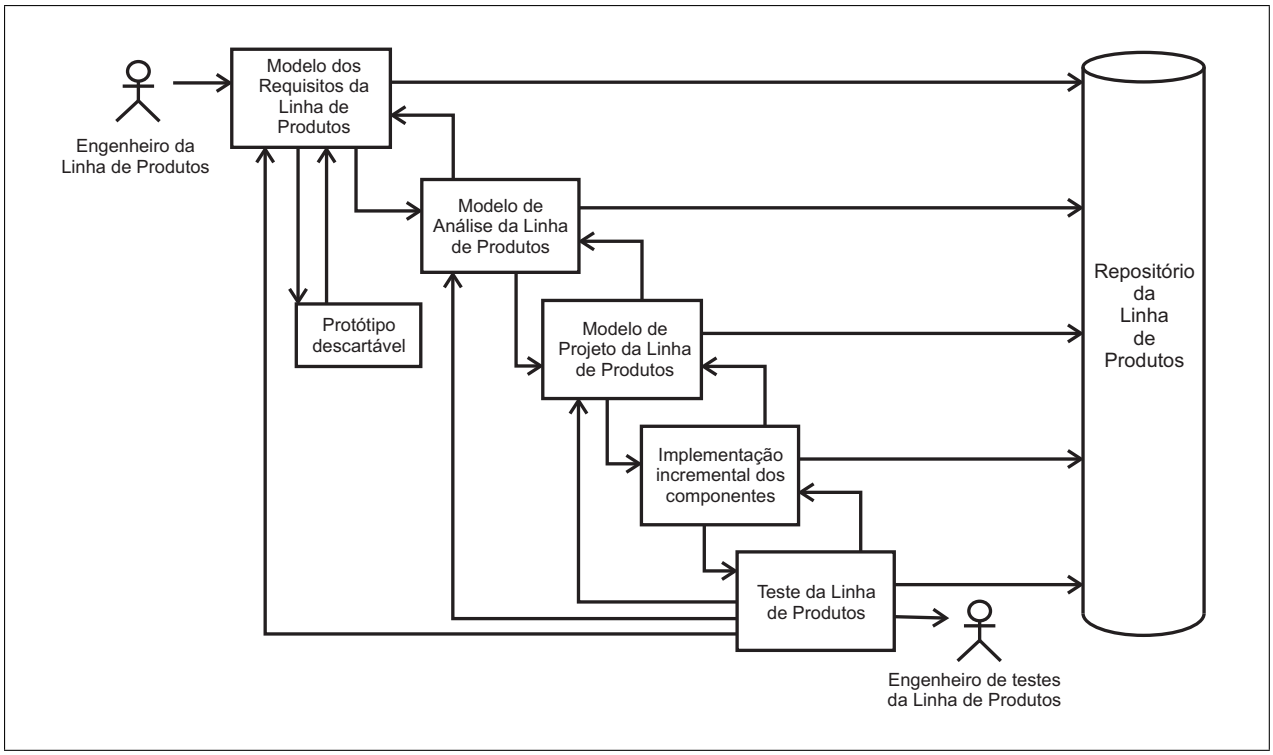

Figura 2.4: Atividades do processo ESPLEP (Gomaa, 2004)

- Features Comuns. São aquelas features que estão presentes em todos os membros da linha de produtos.
- Features Opcionais. São as features que necessitam ser fornecidas somente para alguns membros da linha de produtos. As features opcionais podem supor que as features comuns são fornecidas e depender de sua presença.
- Features Alternativas. Duas ou mais features podem ser alternativas, ou seja, somente uma delas pode estar presente em um membro da linha de produtos de software. Logo, essas features são mutuamente exclusivas.

2.5.5 Programação Orientada a Features

Uma feature pode ser definida como uma característica essencial das aplicações de um domínio (Kang et al., 1998). A principal virtude do uso das features está no fato de elas capturarem e organizarem a terminologia usada por especialistas e usuários de um domínio de aplicação, facilitando assim a comunicação entre os especialistas em software e os usuários.

A Programação Orientada a Features (POF) estuda a modularidade das features nas linhas de produtos, em que uma feature é vista como um incremento na funcionalidade do programa (Smaragdakis, 1999). A idéia central da POF é a síntese de softwares (programas individuais) por meio da composição de várias features.

Uma feature pode "refinar" o conteúdo de outras features. Um refinamento é uma adição de funcionalidade em um projeto de software que pode afetar várias entidades da implementação do projeto (funções, classes, etc.). Deste ponto em diante, o termo refinamento será usado para fazer referência ao conjunto de mudanças que uma feature aplica sobre o código-base[5] ou sobre outras features.

Na POF existe um conceito denominado módulo de camada. Uma camada pode encapsular múltiplas abstrações, que juntas implementam a funcionalidade de uma feature. As abstrações individuais e as camadas que encapsulam essas abstrações são definidas como "*mixins*" e "*mixin layers*" (Smaragdakis e Batory, 2002).

Classes *Mixin* e *Mixin Layers*

Uma classe *mixin* (ou apenas *mixin*) é uma subclasse abstrata que pode ser usada para especializar o comportamento de várias classes pai (Bracha e Cook, 1990). A idéia principal dos *mixins* é a seguinte: nas linguagens orientadas a objetos (OO), as classes pai podem ser definidas sem especificar suas subclasses, entretanto, esta propriedade não é simétrica. Uma subclasse é definida apenas quando existe uma classe pai especificada. *Mixins* representam um mecanismo para especificar classes que herdam eventualmente de classes pai, entretanto, não é necessário que essas classes pai sejam especificadas no local da definição dos *mixins*. Assim, um único *mixin* pode ser instanciado para classes pai diferentes. Esta propriedade faz com que *mixins* sejam apropriados para definir extensões incrementais uniformes para uma variedade de features (Smaragdakis e Batory, 2002).

Uma *mixin layer* (Smaragdakis e Batory, 1998) é um módulo que encapsula vários *mixins* diferentes que representem um interesse do sistema quando agrupados. Na Figura 2.5 é mostrada uma pilha de três *mixin layers* (`layer 1`, `layer 2` e `layer 3`) (Apel et al., 2005c). A pilha cresce de cima para baixo. Essas *mixin layers* entrecortam várias classes (`A`, `B` e `C`). As caixas brancas representam os *mixins* e as linhas contínuas representam heranças ou refinamentos.

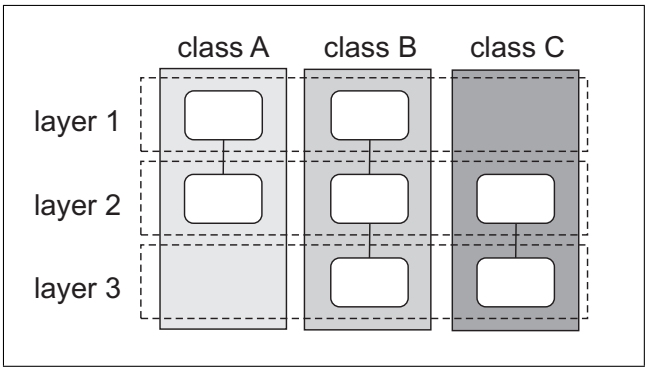

Figura 2.5: Pilha de *mixin layers* (Apel et al., 2005c)

[5]Neste trabalho o adjetivo "base" será utilizado para denotar artefatos que são comuns a todos os membros da linha de produtos.

2.6 Geradores de Aplicações

Geradores de Aplicações são sistemas de software que transformam especificações em uma aplicação. As especificações descrevem o problema ou a tarefa que deve ser realizada pelo gerador. Essas especificações podem ser feitas de forma interativa, em que o usuário seleciona as características desejadas por meio de escolhas em uma seqüência de formulários ou menus, ou também podem ser modeladas de forma gráfica ou escritas em alguma linguagem (Cleaveland, 1988).

O termo gerador de aplicações pode assumir diferentes significados, tais como: compiladores, pré-processadores, meta-funções que geram classes e procedimentos, *wizards* e geradores de código (Czarnecki e Eisenercker, 2002). Um *wizard*, por exemplo, é um programa gráfico que recebe uma especificação em alto nível de abstração e transforma essa informação em software. Algumas ferramentas CASE (do inglês *Computer Aided Software Engineering*) realizam algum tipo de geração de código, utilizando como entrada os diagramas e especificações que são inseridos na ferramenta. Os compiladores são geradores de código, que, uma vez que recebem uma informação em um nível de abstração, tais como a linguagem Java ou C++, transformam essa informação em uma linguagem de mais baixo nível, tais como código objeto, *bytecode* ou código de máquina (Czarnecki e Eisenercker, 2002).

A Figura 2.6 mostra o processo básico necessário para desenvolver um programa utilizando um gerador de aplicações (Cleaveland, 1988). Começa com uma especificação que descreve o que o programa deve fazer. Ela serve como entrada para o gerador, o qual produz o programa. Finalmente o programa é compilado para produzir a aplicação executável.

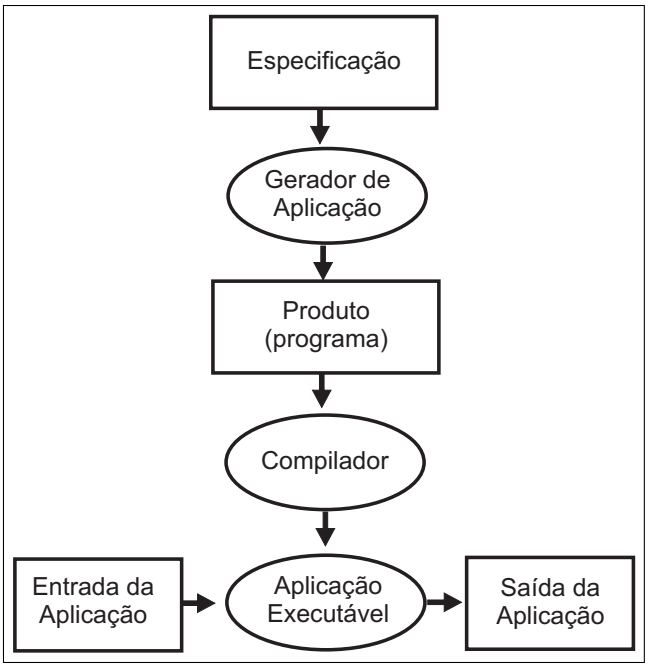

Figura 2.6: Modelo para um gerador de aplicações (Cleaveland, 1988)

Os geradores de aplicações podem ajudar na construção de múltiplos produtos de uma família[6] com mais facilidade do que a maneira de implementação tradicional. Além de código, os geradores podem produzir documentação do usuário e do software, casos de teste, diagramas e figuras (Cleaveland, 2001).

O processo de desenvolvimento que utiliza geradores durante a engenharia de aplicações pode ser mais rápido e menos suscetível a erros humanos do que o processo tradicional de codificação, ou seja, os geradores podem produzir código de forma sistemática e mais segura em relação aos métodos tradicionais de programação (Cleaveland, 2001; Czarnecki e Eisenercker, 2002; Smaragdakis e Batory, 2000).

2.6.1 Tipos de Transformações

Existem dois tipos de transformações sobre artefatos que um gerador pode executar: transformações verticais e transformações horizontais. Uma transformação vertical é uma transformação que recebe como entrada uma representação de alto nível e fornece como saída uma representação de baixo nível, porém deixando a modularidade de alto nível preservada. Cada transformação vertical implementa um módulo de alto nível em um ou mais módulos de baixo nível, dentro dos limites estabelecidos pela representação de alto nível. Esses geradores são chamados de geradores de composição (Czarnecki e Eisenercker, 2002).

As transformações horizontais realizam transformações que redefinem a estrutura modular das representações de alto nível. Uma transformação horizontal modifica a estrutura modular da representação de alto nível, por meio da inserção, remoção ou integração dos módulos (Czarnecki e Eisenercker, 2002).

Existem transformações que são tanto horizontais quanto verticais. Essas transformações são denominadas transformações oblíquas. Um diagrama esquemático das transformações horizontais, verticais e oblíquas é apresentado na Figura 2.7.

2.6.2 Arquiteturas de Geradores de Aplicação

Segundo Masiero e Meira (1993), pode-se classificar diferentes arquiteturas de geradores de aplicações de acordo com a capacidade de adptação de um gerador para múltiplos domínios e/ou múltiplas aplicações, são elas: SDSA (*Single Domain Single Application*), SDMA (*Single Domain Multiple Application*), MDSA (*Multiple Domain Single Application*) e MDMA (*Multiple Domains Multiple Applications*). A seguir são apresentadas em detalhes apenas as arquiteturas SDSA e MDMA por estarem relacionadas com a proposta deste trabalho e também porque as arquiteturas SDMA e MDSA possuem pouca aplicação prática.

[6]O termo família de aplicação pode ser definido como conjunto de itens que possuem aspectos comuns e variabilidades previsíveis (Weiss e Lai, 1999).

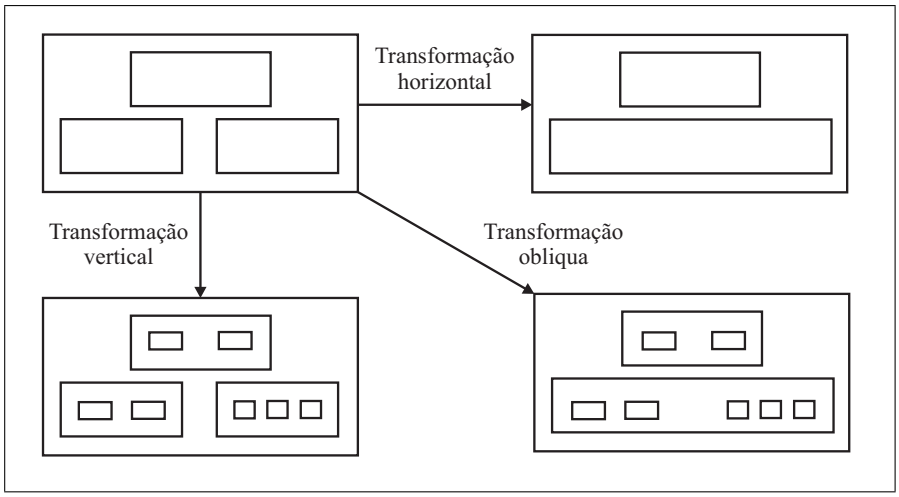

Figura 2.7: Transformações vertical, horizontal e oblíqua (Czarnecki e Eisenercker, 2002)

Os geradores de SDSA são específicos e geram aplicações de único domínio, produzindo sempre o mesmo tipo de produto. Pode-se citar como exemplos de geradores SDSA os utilitários LEX e YACC, que geram analisadores léxicos e sintáticos, respectivamente. Os geradores MDMA são mais gerais e com eles o usuário tem liberdade para definir o domínio da aplicação e as diferentes aplicações dentro de cada domínio.

Geradores MDMA podem ser adaptados para gerar aplicações de diferentes domínios. A abordagem DRACO (Neighbors, 1984) pode ser vista como um gerador MDMA, ela enfatiza o reúso de componentes de software. Esses componentes são organizados em domínios, que por sua vez, são representados por linguagens com sintaxe e semântica bem definidas. Essa abordagem também possui um mecanismo para transformação e reúso de especificações de domínio.

Ao construir um gerador de aplicações SDSA, deve-se especificar o domínio onde o gerador atuará. Caso houvessem vários domínios, seria necessário desenvolver vários geradores, o que tornaria inviável a sua utilização, visto que geradores de aplicações são ferramentas custosas e complexas de desenvolver. Nesse contexto, Shimabukuro (2006) apresentou um estudo sobre geradores MDMA, ressaltando como principal vantagem dessa abordagem o fato de que as atividades de configuração de um gerador MDMA podem ser menos custosas do que a construção completa de um gerador de aplicação específico, diminuindo o custo de uso do gerador e permitindo a utilização das técnicas de geração para domínios que normalmente não apresentam viabilidade econômica para utilização dessas ferramentas (Shimabukuro, 2006).

No trabalho de Shimabukuro (2006) os geradores MDMA são denominados como Geradores de Aplicação Configuráveis (GAC's) e é apresentado o gerador Captor (do inglês *Configurable APplication generaTOR*), que é uma ferramenta que pode ser configurada utilizando linguagens de modelagem de aplicações definidas para domínios específicos. O Captor será apresentado detalhadamente no Capítulo 4 por ser objeto de interesse deste trabalho.

2.7 Considerações Finais

Este capítulo apresentou várias metodologias utilizadas para promover o reúso de artefatos de software que formaram a base para o desenvolvimento deste trabalho de mestrado. Tais metodologias buscam a redução do tempo de desenvolvimento de software e o aumento da qualidade do produto final. As metodologias de padrões de software, linguagens de padrões e frameworks de software buscam mais do que o reaproveitamento de código objeto e classes, o objetivo central dessas técnicas é o reúso do conhecimento adquirido e a consolidação de boas práticas de desenvolvimento dentro de uma arquitetura que possa ser aplicada em diferentes contextos para a solução de um mesmo problema.

Linhas de produtos de software podem incorporar várias dessas metodologias, entretanto, são utilizadas para a produção em larga escala. Por se tratarem de sistemas de software complexos e que necessitam de uma grande quantidade de tempo e recursos para serem criados, é importante seguir um método de desenvolvimento bem definido, como os métodos PLUS e FAST, por exemplo. O método PLUS pode ser integrado com o processo unificado, que por sua vez é um processo consolidado e amplamente difundido pela comunidade de engenharia de software. No método FAST o objetivo principal é criar uma LPS onde o tempo de instanciação de produtos seja o menor possível.

Um fator importante a ser levado em conta ao se verificar a viabilidade de uma LPS é que o custo de criação da arquiterura da linha de produtos deve ser compensado pela redução do custo de produção de cada um dos membros da família de produtos. A utilização de geradores de aplicações na etapa de produção dos membros pode reduzir custos, sem afetar a qualidade do produto final. Pode-se utilizar de um gerador MDMA (gerador configurável) existente, visto que o processo de configuração de um novo domínio em gerador configurável é menos custoso do que a construção de um novo gerador específico.

A utilização de geradores configuráveis ainda é pouco difundida devido à complexidade e aos altos custos envolvidos no desenvolvimento de um gerador configurável a partir do zero. Uma vez que não é possível encontrar um gerador configurável pronto que atenda aos requisitos de uma LPS e a criação de um novo gerador configurável para um domínio apenas é inviável, acaba-se optando pela utilização de geradores tradicionais. Constata-se assim, a necessidade de realizar melhorias que tornem os geradores configuráveis mais reusáveis.

CAPÍTULO

3

Linhas de Produtos e Aspectos

3.1 Considerações Iniciais

Neste capítulo são apresentados os conceitos relativos à programação orientada a aspectos e uma revisão da literatura sobre trabalhos que relacionam linhas de produtos e aspectos. Foi traçada uma linha de evolução desses trabalhos desde o surgimento do conceito de aspectos e as primeiras propostas de modularização de features transversais[1] até as ferramentas mais atuais de geração de produtos. Essas pesquisas foram agrupadas em quatro categorias principais: abordagens ad-hoc, abordagens sistemáticas, abordagens baseadas no desenvolvimento incremental e ferramentas.

Na Seção 3.2 é apresentada uma visão geral da programação orientada a aspectos e na Seção 3.3 é descrita a linguagem AspectJ, que é uma extensão da linguagem Java que oferece suporte para a utilização de aspectos. Na Seção 3.4 são apresentadas as primeiras abordagens em que a programação orientada a aspectos foi utilizada no projeto de linhas de produtos, as abordagens ad-hoc. Nesses trabalhos as decisões relacionadas à implementação de features e aspectos são baseadas no conhecimento de domínio do engenheiro da linha de produtos. Na Seção 3.5 são mostrados os trabalhos classificados como abordagens sistemáticas. Os trabalhos dessa categoria são caracterizados por apresentarem um processo bem definido de apoio à implementação de features transversais.

A Seção 3.6 discute as abordagens baseadas no desenvolvimento incremental. Essas abordagens tratam features como camadas sobrepostas ao sistema base, onde novas features são inseridas por meio de refinamentos (incrementos) nas features já existentes. A última categoria identificada,

[1]Neste trabalho será adotado o termo feature transversal para denotar uma feature que implementa um interesse transversal da linha de produtos

ferramentas, é descrita na Seção 3.7. Os trabalhos dessa categoria descrevem ferramentas de automatização para a criação de membros da linha de produtos e que sejam apoiadas pela POA. Por fim, na Seção 3.8 são apresentadas as considerações finais deste capítulo.

3.2 Programação Orientada a Aspectos

A separação de interesses (*concerns* em inglês) é um princípio fundamental que cuida das limitações da cognição humana ao lidar com um ambiente complexo. Esse princípio defende que, para contornar a complexidade, deve-se resolver uma questão importante (ou interesse) por vez (Dijkstra, 1976). O termo interesse é utilizado dentro do contexto da Engenharia de Software para representar requisitos que precisam estar presentes nos sistemas. Esses interesses podem ser tanto funcionais, tais como regras de negócios, quanto não-funcionais, como gerenciamento de transação e persistência (Groher e Baumgarth, 2004). Pode-se considerar como interesses desde noções de alto nível, tais como segurança e qualidade de serviço, a noções de baixo nível, como *caching* ou *buffering* (Elrad et al., 2001a).

Interesses transversais (do inglês, *crosscutting concerns*) são interesses cujo código espalha-se pelas divisões ou módulos do programa. Por exemplo, na programação orientada a objetos tem-se a divisão em classes, e o código de certos interesses pode ficar espalhado por diversos componentes do sistema (*scattering*), afetando o desempenho ou a semântica da aplicação. Por "atravessar" ou "entrecortar" várias classes, o interesse é dito transversal (ou ortogonal).

Embora possam ser visualizados e analisados relativamente em separado, a implementação de interesses transversais em linguagens orientadas a objetos ou estruturadas torna-se confusa. Além disso, seu código encontra-se espalhado pelo código da aplicação, dificultando a separação entre a funcionalidade básica do sistema e essas propriedades. Esse fenômeno, denominado entrelaçamento de código (*tangling*), é responsável por parte da complexidade encontrada nos sistemas atuais. Ele favorece o aumento da dependência entre os componentes funcionais, desviando-os da sua finalidade principal e os tornando menos reusáveis e mais propensos a erros (Kiczales et al., 1997).

A Programação Orientada a Aspectos (POA) (Kiczales et al., 1997) é uma metodologia de programação que oferece suporte a mecanismos de abstração e composição com o objetivo de alcançar uma melhor separação de interesses no nível da implementação (Chavez, 2004). A POA promove a separação avançada de interesses ao introduzir uma nova unidade modular, chamada **aspecto**, para a modularização dos interesses transversais (Kiczales et al., 2001).

Aspectos são utilizados para implementar interesses transversais e são capazes de entrecortar pontos bem definidos no fluxo de execução de um programa. No entrecorte, são capazes de executar algum comportamento antes, depois ou no lugar do ponto entrecortado, como chamadas de métodos, construtores, modificação de valores de atributos, etc. No último caso, quando o aspecto executa um comportamento no lugar do ponto entrecortado, o aspecto pode ou não devolver o fluxo

de controle para que o ponto entrecortado seja executado. Os pontos bem definidos na execução de um programa capazes de serem entrecortados são chamados de pontos de junção (*join points*) e são determinados pelo aspecto de acordo com as regras da linguagem utilizada.

Um exemplo de interesse transversal é o requisito não-funcional de "rastreamento" (*tracing*), em que se deseja saber, por exemplo, quais métodos foram acessados durante a execução de um determinado programa. Uma maneira simples de implementar o *tracing* com linguagens orientadas a objetos é introduzir uma linha de código que imprime uma indicação de que o método foi executado. Essa implementação faz com que o interesse *tracing* fique entrelaçado no sistema, pois os módulos, além de implementar o interesse desejado, também contêm o código do *tracing*. O interesse também fica espalhado pelo sistema, pois está implementado em vários módulos.

Uma implementação do interesse *tracing* pode ser vista na Figura 3.1 (Eler, 2006), em que a linha de código `System.out.println("...")` é responsável por implementar o interesse em cada método das classes do modelo. No modelo de classes à direita do código-fonte, as faixas pretas dentro das classes indicam a implementação do *tracing* espalhado pelas classes do sistema (dentro de cada método).

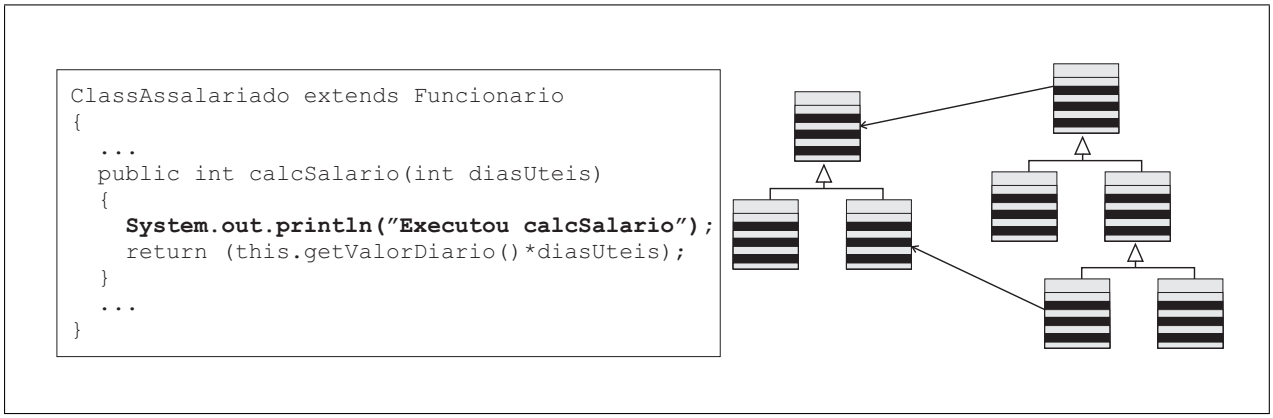

Figura 3.1: Implementação de *tracing* OO convencional com espalhamento e entrelaçamento de código. (Eler, 2006)

Na Figura 3.2 (Eler, 2006) é mostrada uma implementação do interesse de *tracing*, mostrado anteriormente na Figura 3.1, com a programação orientada a aspectos. Pode-se notar que o interesse que estava espalhado pelas classes do sistema está agora localizado em uma unidade denominada de aspecto. Maiores detalhes sobre a sintaxe são mostrados na Subseção 3.3.

O desenvolvimento na POA é apoiado por meio de linguagens de programação específicas, tais como AspectJ[2] e Hyper/J[3]. A abordagem POA introduz novos construtores de linguagens por intermédio da definição de alguns conceitos básicos, para que o código do aspecto possa ser combinado com o código da aplicação. A partir daí, de acordo com Elrad et al. (2001b), uma linguagem orientada a aspectos deve determinar:

[2]The AspectJ Project: `http://eclipse.org/aspectj`.
[3]Hyper/J: `http://www.alphaworks.ibm.com/tech/hyperj`.

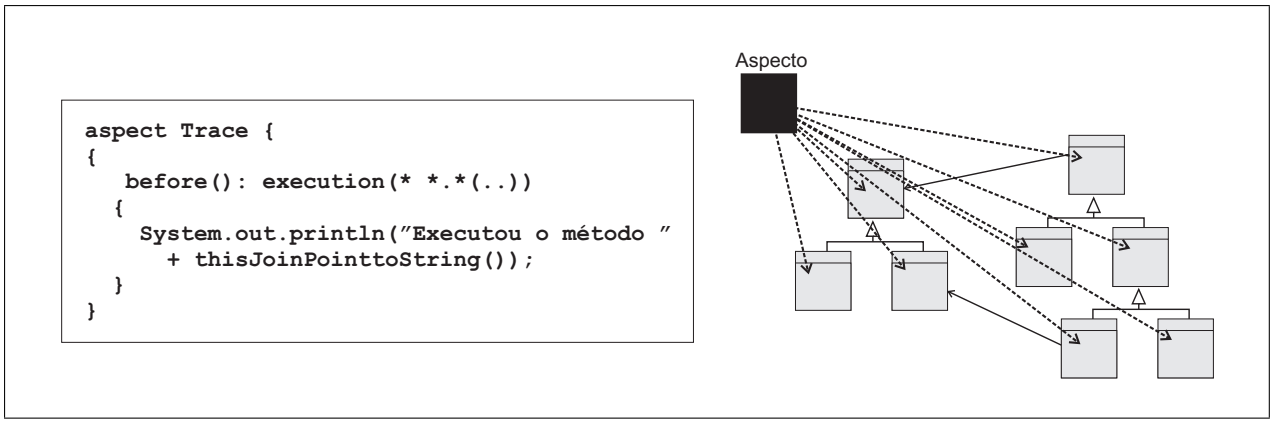

Figura 3.2: Implementação de *tracing* com aspectos. (Eler, 2006)

- Um modelo de pontos de junção que descreva os ganchos onde os entrecortes podem ocorrer;
- Um mecanismo de identificação dos pontos de junção;
- Unidades que encapsulem, tanto especificações de pontos de junção, quanto mudanças de comportamento desejadas;
- Um processo para combinar as unidades em um programa (*weaver*).

Na Seção a seguir, são apresentadas as características, mecanismos, sintaxe e alguns detalhes da linguagem AspectJ. Essa linguagem tem sido amplamente utilizada em diversas pesquisas relacionadas com a POA.

3.3 Linguagem AspectJ

A linguagem AspectJ é uma extensão da linguagem Java para apoiar a programação orientada a aspectos. É uma linguagem de propósitos gerais e foi desenvolvida no centro de Pesquisa da Xerox (Kiczales et al., 1997, 2001), em Palo Alto. Posteriormente foi agregada ao projeto Eclipse[4] da IBM.

Na programação orientada a aspectos é introduzido o conceito de aspectos (*aspect*), que implementam os interesses transversais do sistema e adicionam comportamento ao código-base (classes, por exemplo) do sistema. Para permitir a implementação de aspectos, o AspectJ introduz novas construções, tais como: conjuntos de pontos de junção (*pointcut*) que identificam pontos de junção (*join points*) e os adendos (*advices*) que implementam o comportamento a ser adicionado no ponto de junção. Além disso, pode-se realizar declarações de atributos e métodos que alteram a estrutura estática das classes afetadas pelo aspecto, chamadas de declarações inter-tipos (*intertype declarations*). A linguagem AspectJ também permite a alteração da hierarquia das classes afetadas pelo aspecto e a declaração de hierarquias entre os aspectos, por meio de aspectos especializados.

[4]Eclipse Project: `http://www.eclipse.org/`.

3.3.1 Adendos

Os adendos são implementações de comportamento que são executados no momento em que uma classe é entrecortada pelo aspecto. Os adendos podem ser de três tipos básicos: anteriores (*before*), que são executados antes do ponto de junção ser executado; posteriores (*after*), que são executados depois do ponto de junção ser executado; e os de contorno (*around*), que são executados no lugar do ponto de junção, sendo que após serem executados podem ou não retornar a execução do ponto de junção utilizando um método denominado `proceed()`.

Os adendos são construções semelhantes aos métodos, entretanto não podem ser chamados diretamente pela aplicação base e nem pelo próprio aspecto, pois sua execução é feita automaticamente após o entrecorte no ponto de junção. Uma outra diferença em relação aos métodos é que os adendos não possuem nome, não têm especificadores de visibilidade (`public`, `private`, etc) e têm acesso às informações do contexto das execuções dos pontos de junção, como a assinatura e os argumentos dos métodos entrecortados.

3.3.2 Conjunto de Pontos de Junção

Os conjuntos de ponto de junção são utilizados para identificar um grupo de pontos bem definidos no fluxo de execução de um programa, em que os comportamentos adicionais inseridos pelos aspectos devem ser executados, tais como construtores de classes, chamadas de métodos, leitura e escrita de atributos, etc.

Os conjuntos de pontos de junção podem identificar um único ponto de junção ou a composição de vários deles, usando operadores lógicos como `&&` (e), `||` (ou), além de operadores unários como o operador `!` (negação). Os conjuntos de pontos de junção podem ser criados na própria declaração dos adendos ou serem declarados no corpo do aspecto.

3.3.3 Declarações inter-tipos

As declarações inter-tipo permitem a introdução de novos atributos e métodos nas classes básicas do programa. Isso é feito simplesmente pela declaração de atributos e métodos no próprio aspecto, com esses novos métodos podendo ser acessados diretamente pelas classes afetadas.

3.3.4 Processo de Combinação

Como os aspectos são desenvolvidos em módulos separados dos módulos da aplicação base, há a necessidade de um processo de combinação dos aspectos com o programa base. O processo de combinação pode ser realizado tanto em tempo de compilação, também chamada de combinação estática, quanto em tempo de execução, também chamada de combinação dinâmica, de acordo com a terminologia adotada pela linguagem AspectJ (Kiczales et al., 2001).

No processo de combinação entre aspectos e classes, o compilador AspectJ transforma os aspectos em implementações semelhantes às classes e os adendos se tornam semelhantes aos métodos. Diante disso, o combinador identifica os possíveis pontos de junção das classes afetadas pelos aspectos e insere uma chamada ao método do aspecto (adendo) antes, depois ou no lugar do ponto de junção, de acordo com o adendo. Esse processo é feito diretamente no *bytecode*.

Após essa breve revisão sobre os conceitos da POA, são apresentados nas próximas Seções diversos estudos sobre a inclusão da POA no processo de engenharia de linha de produtos. Tais pesquisas têm o objetivo de aproveitar a facilidade de modularização de features proporcionada pela separação avançada de interesses.

3.4 Abordagens Ad-hoc

Foram classificadas como abordagens ad-hoc, as abordagens em que as decisões relacionadas à implementação de features e aspectos são baseadas no conhecimento de domínio do engenheiro da linha de produtos. O trabalho de Anastasopoulos e Muthig (2004) reflete os primeiros passos em direção à combinação de aspectos e famílias de produtos. Nesse trabalho é apresentado um estudo de caso sobre um framework de aplicações para telefone celular em que uma feature opcional é implementada utilizando aspectos e os pontos de variação são representados como conjuntos de pontos de junção da linguagem AspectJ.

Com os dados obtidos a partir do estudo de caso, foram identificados alguns fatores positivos resultantes da utilização da POA, mais especificamente, da linguagem AspectJ no desenvolvimento de features. Entre eles pode-se citar:

- Facilidade de reúso: O esforço da engenharia de aplicações para integrar funcionalidades comuns e variantes foi reduzido, visto que o combinador de aspectos da linguagem AspectJ faz isso automaticamente.
- Melhora na manutenibilidade: A modularização das features transversais torna a manutenção dessas features mais fácil.
- Facilidade na geração de membros: O processo de geração e de combinação de features foi automatizado devido à utilização de um *script* de compilação que é capaz de acoplar ou desacoplar features transversais em tempo de compilação.

No trabalho de Lohmann et al. (2006) é apresentada uma abordagem na qual a POA é utilizada para a implementação de propriedades não-funcionais. Esse trabalho apresenta a família PURE, uma família de sistemas operacionais embarcados, e as atividades realizadas durante a implementação de propriedades não-funcionais dessa família utilizando aspectos.

Mesmo sem definir um método para implementação de features como aspectos, as abordagens ad-hoc apontam para a viabilidade de integração da POA em linha de produtos de software.

3.5 Abordagens Sistemáticas

São definidas como abordagens sistemáticas as pesquisas que possuem um conjunto de *guidelines* ou um processo voltado para o desenvolvimento de features transversais de acordo com o seu comportamento dentro da linha de produtos. Segundo (Lee et al., 2006), o comportamento das features transversais pode ser classificado em dois tipos: homogêneo e heterogêneo. Features que possuem comportamento transversal homogêneo são aquelas que adicionam o mesmo comportamento em diferentes partes do sistema (*logging*, por exemplo). Similarmente, features que possuem comportamento transversal heterogêneo, são aquelas que adicionam diferentes comportamentos em diferentes partes do sistema.

Em (Lee et al., 2006) são propostas algumas *guidelines* para a escolha da forma de implementação das features de uma linha de produtos:

1. Implementar similaridades como componentes base. Não é necessário que todas as features comuns sejam implementadas como componentes base. Features comuns que possuam comportamento transversal homogêneo podem ser implementadas como aspectos. Por outro lado, as features comuns que possuem comportamento transversal heterogêneo podem ser incorporadas aos componentes base ou podem ser distribuídas em vários componentes aspectuais.

2. Implementar features variantes (features opcionais ou alternativas) que possuem comportamento modular como componentes variantes tradicionais.

3. Implementar features variantes que possuem comportamento transversal homogêneo e heterogêneo como componentes aspectuais. Dessa forma as variabilidades contidas nessas features estarão modularizadas dentro do componente aspectual, facilitando assim o processo de composição dos membros.

4. Implementar as dependências entre features em módulos aspectuais separados. Os conjuntos de diferentes dependências entre features podem ser implementados como aspectos concretos que estendem diferentes aspectos abstratos. Essa técnica torna possível a adição de features variantes sem afetar os artefatos existentes, melhorando a adaptabilidade e a reusabilidade dos artefatos da linha de produtos.

Segundo Heo e Choi (2006), a separação das dependências entre features por meio de aspectos aumenta a adaptabilidade dos componentes da linha de produtos. A composição entre features base e variantes da linha de produtos pode ser feita por meio de aspectos, de tal forma que não seja necessário alterar o código-base das features. Aspectos ficam encarregados de fazer a combinação dos componentes base com os componentes variantes, melhorando dessa forma, a modularidade dos artefatos, além de facilitar o processo de geração de membros, aumentando assim a produtividade da linha de produtos (Heo e Choi, 2006).

3.6 Abordagens baseadas no Desenvolvimento Incremental

A POA permite isolar as variabilidades de uma linha de produto dentro de aspectos e integrá-las de maneira aditiva, isto é, as features comuns são usadas para implementar uma estrutura modular base e as features variantes são modeladas como aspectos. Dessa forma, um combinador de aspectos pode gerar produtos por meio da combinação da estrutura modular base com os aspectos (Lee et al., 2006).

Algumas abordagens propõem um processo de modularização de interesses transversais existentes em linhas de produtos, tal como as abordagens sistemáticas, mas além disso, se baseiam no desenvolvimento de features transversais como incrementos sobre o programa base existente. Essas abordagens foram classificadas como abordagens baseadas no desenvolvimento incremental. A maioria dos trabalhos pesquisados mostram formas de integrar a POA com a Programação Orientada a Features (POF) (Ver Seção 2.5.5), utilizando modelos de arquitetura de dois grupos de pesquisa, o grupo de pesquisa do modelo de arquitetura Caesar (Mezini e Ostermann, 2003) e o grupo do modelo AHEAD (*Algebraic Hierarchical Equations for Application Design*) (Batory et al., 2003).

Nesta seção também é apresentado o Método de Pacios (2006). Esse método não é baseado em nenhum modelo de arquitetura específico, mas também propõe um processo para o desenvolvimento incremental de linhas de produtos baseado em aspectos.

3.6.1 Modelo Caesar

O modelo Caesar é um modelo de arquitetura para linhas de produtos que suporta a componentização de aspectos por meio do encapsulamento de classes virtuais, pontos de junção e adendos em componentes aspectuais. Os componentes aspectuais podem ser compostos de forma incremental (Mezini e Ostermann, 2003).

Segundo Mezini e Ostermann (2004), na POF tradicional o engenheiro da linha de produtos é forçado a expressar as features de acordo com a estrutura hierárquica básica, isto é, uma nova feature deve ser representada por meio de refinamentos nas classes base existentes. Entretanto, algumas features podem entrecortar a estrutura modular das classes base.

A integração do código implementado das features variantes com o código do programa base geralmente é feita por meio de sobrecarga de métodos e herança de interfaces. Graças às noções de pontos de junção e de adendos da POA, a interação das features variantes com o programa base pode ser expressa de forma mais clara. Contudo, a POA, mais especificamente a linguagem AspectJ, possui duas deficiências com relação à POF: a primeira é a impossibilidade de coexistência de múltiplas variantes de uma feature no mesmo sistema e a segunda é o fato de que o código de um aspecto é muito específico, portanto, pouco reusável (Mezini e Ostermann, 2004).

O modelo Caesar tem o objetivo de combinar as qualidades das abordagens orientadas a features e orientadas a aspectos. Os conceitos de pontos de junção e adendos do Caesar possuem um mecanismo de implantação capaz de especificar em qual contexto as definições do adendo podem ser ativadas de acordo com a instância da classe, criando dessa forma diversas variantes da feature.

O modelo Caesar introduz o conceito de Interface de Colaboração Aspectual (ICA) - uma definição de interfaces para aspectos com tipos aninhados - para tratar dos problemas relacionados com a reusabilidade e tornar as features menos dependentes de contexto. A finalidade da ICA é relizar o desacoplamento entre a implementação do aspecto e a definição dos tipos utilizados pelo aspecto, os quais são definidos em módulos independentes, mas indiretamente conectados. A idéia é que esses módulos independentes implementem partes disjuntas de uma ICA, que por sua vez é responsável pela ligação desses módulos (Mezini e Ostermann, 2004).

3.6.2 Modelo AHEAD

O modelo AHEAD é um modelo arquitetural para implementação de linhas de produtos (Batory et al., 2003). O foco do AHEAD é a decomposição de programas abstratos em features básicas e a composição de pilhas de features para derivar programas concretos. Os produtos gerados pela linha de produtos podem tanto reusar features existentes como adicionar novas features à linha de produtos. Esse processo iterativo é denominado de refinamento *step-wise*.

Nesse modelo é utilizada a programação composicional em grande escala, generalizando os conceito de features e refinamentos de features. As features não são compostas apenas por código, mas também por diversos tipos de artefatos, por exemplo, *makefiles*, diagramas UML e documentação. Cada artefato dentro de uma feature pode refinar artefatos de features localizadas nas camadas inferiores.

Para a implementação de linhas de produtos no modelo AHEAD pode ser utilizada a linguagem FEATUREC++ (Apel et al., 2005b), que é uma extensão proprietária do C++ apoiada pela POF. No trabalho Apel et al. (2005a) são identificadas algumas deficiências relacionadas à representatividade da POF e à capacidade de evolução das linhas de produtos. São elas:

1. Falta de apoio da indústria. Atualmente, a POF ainda é um conceito predominantemente acadêmico, visto que a linguagem utilizada nas implementações dos conceitos, a linguagem Java, ainda não é aceita em vários domínios, como sistemas operacionais, bancos de dados, sistemas embarcados, entre outros. Percebe-se uma demanda por soluções baseadas em C++.

2. Falta de suporte à modularidade transversal. Durante a sua evolução, o software precisa ser adaptado para atender requisitos e circunstâncias imprevistas. Isto resulta em modificações e em extensões que podem entrecortar muitas unidades existentes.

No sentido de suprir essas deficiências, são apresentadas em Apel et al. (2005a) uma extensão da linguagem FEATUREC++ capaz de modularizar interesses transversais e também uma técnica

de implementação de features transversais chamada *Aspectual Mixins Layers* (AML). Como visto na Seção 2.5.5, um *mixin layer* é um conjunto de classes que representam um determinado interesse do sistema. Um *Aspectual Mixin Layer* (AML) é definido como sendo um *mixin layer* com aspectos embutidos.

A Figura 3.3 (Apel et al., 2005a) mostra uma pilha de *mixin layers* que implementa uma feature de *buffer*. As três primeiras camadas (*buffer* básico, alocação e sincronização) são implementadas como *mixin layers* comuns e a camada *logging* é implementada como um *Aspectual Mixin Layer* (`Log`). O AML `Log` é composto por um aspecto que captura o conjunto de métodos que devem ter suas atividades registradas. Esse conjunto é indicado pelas setas tracejadas.

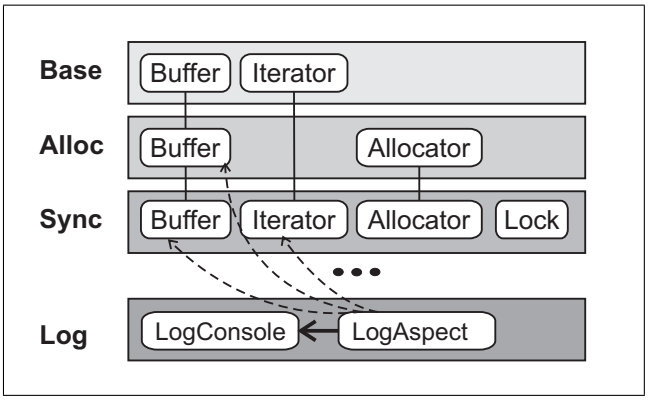

Figura 3.3: Implementação da feature de logging usando Aspectual Mixins Layers (Apel et al., 2005a)

No trabalho de Apel et al. (2005c) é proposta a criação de um mecanismo de segurança capaz de assegurar que os aspectos da LPS afetem somente artefatos provenientes de incrementos anteriores. Caso contrário, podem ocorrer situações onde determinados aspectos comecem a atuar sobre features integradas em etapas posteriores do desenvolvimento. Visto que esses aspectos foram implementados sem consciência dessas novas features, podem surgir efeitos colaterais e erros não previstos.

Essa restrição no escopo dos aspectos pode não ser desejada, logo é necessário que o programador possa mudar a política de limitação e deixar que os aspectos introduzidos afetem todos os estágios do desenvolvimento. Entretanto, o mecanismo deve apontar todos os possíveis problemas e assim ajudar a impedir erros (Apel et al., 2005c).

Em Apel e Batory (2006) é apresentado um estudo de caso em que foram colocados em prática todos os conceitos teóricos acerca do modelo AHEAD, da linguagem FEATUREC++ e da utilização de *Aspectual Mixin Layers* no desenvolvimento de LPS. O estudo de caso consistiu na criação de uma linha de produtos para aplicações de redes P2P. Por meio desse estudo foram levantadas algumas informações estatísticas, conforme mostrado na Figura 3.4.

De acordo com esse estudo de caso os autores concluem que aspectos são mecanismos úteis de modularização, pois auxiliam nas situações em que as técnicas de orientação a objetos tradicionais e os *mixins* falham. Esse estudo observou que (i) a utilização de aspectos ao invés de

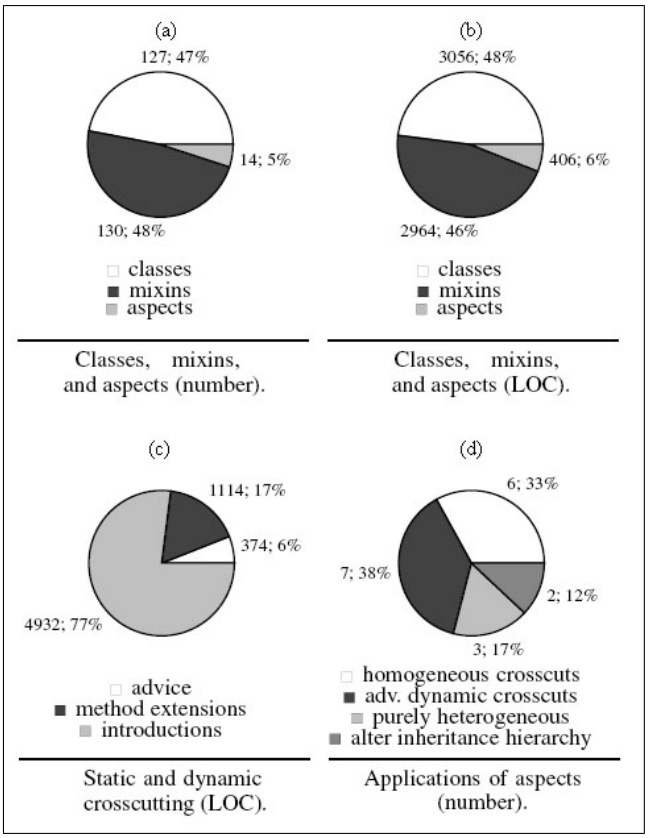

Figura 3.4: Dados estatísticos do estudo de caso (Apel e Batory, 2006)

classes tradicionais reduz a quantidade de código replicado ao implementar interesses transversais homogêneos, (ii) melhora a modularização de interesses transversais, e (iii) possibilita alterações subseqüentes nos relacionamentos de herança (Apel e Batory, 2006).

3.6.3 Método de Pacios

Esse método foi apresentado por Pacios (2006) em sua dissertação de mestrado. É um método voltado para a construção incremental de linhas de produtos de sistemas de informação, utilizando aspectos. O método está dividido em duas fases: Desenvolvimento da Base e Criação dos Produtos, conforme ilustrado na Figura 3.5.

A primeira fase da abordagem é o Desenvolvimento da Base. Nessa fase é construída a base da linha de produtos, que é a implementação do conjunto de todas as features básicas da linha de produtos. Nessa fase é executada uma análise de domínio da LPS, essa análise tem a finalidade de proporcionar o entendimento necessário para a construção da linha de produtos. A análise de domínio irá propiciar a descoberta do escopo da linha de produtos. Após a análise de domínio, inicia-se o desenvolvimento das features básicas. Esse desenvolvimento deve permitir que features posteriormente desenvolvidas também possam ser compostas com a base.

Na segunda fase, Criação dos Produtos, diversas iterações podem ocorrer para desenvolver as features necessárias para a instanciação dos produtos concretos da LPS. Cada iteração resultará

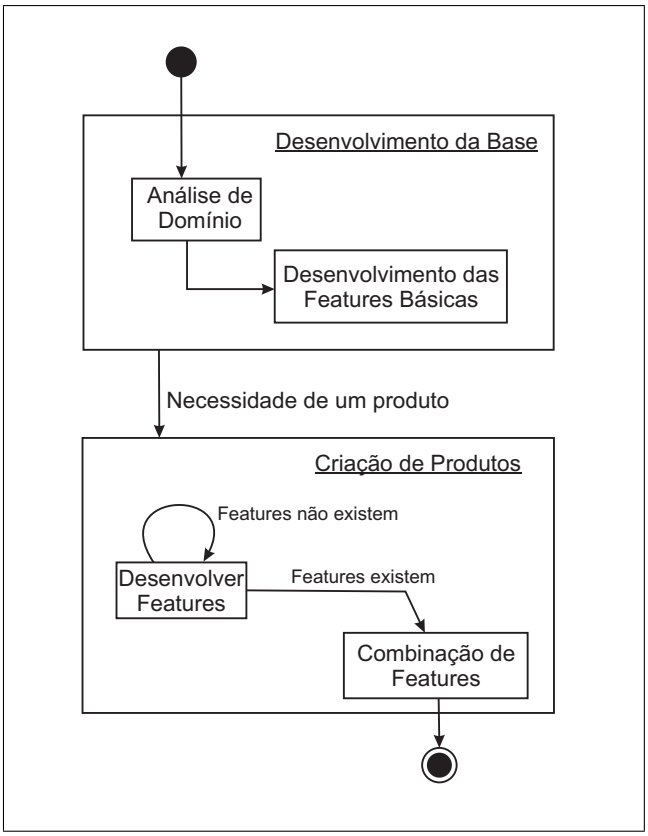

Figura 3.5: Abordagem orientada a aspectos para desenvolvimento de linhas de produtos de software (Pacios, 2006)

em um conjunto de features que são específicas de um produto em particular, mas que podem ser reutilizadas em outros produtos. Podem ser usadas técnicas orientadas a aspectos sempre que for necessário isolar features transversais. Os produtos são obtidos por meio da combinação entre aspectos e código-base de acordo com os requisitos. A utilização de aspectos nesse contexto proporciona facilidade para adição de novas features, evitando ter que projetar e desenvolver novamente partes da base da LPS.

3.7 Ferramentas

Atualmente, a maioria dos estudos relacionados com a modularização de interesses tranversais existentes em LPS utilizam arquiteturas baseadas na instanciação de membros por meio de herança. Entretanto, determinadas etapas do processo podem ser agilizadas utilizando ferramentas de automatização, de preferência, ferramentas capazes de lidar com features transversais.

O trabalho de Lesaint e Papamargaritis (2004) apresenta um *toolkit* para o desenvolvimento de linhas de produtos baseado no modelo de arquitetura GenVoca (Batory et al., 2000). Esse *toolkit* é composto por:

1. Uma linguagem para geração de gabaritos da arquitetura da linha de produtos. Essa linguagem de geração implementa os componentes do núcleo da linha de produtos como classes

modulares tradicionais e os refinamentos nesses componentes são implementados por meio de componentes aspectuais.

2. Uma linguagem para configuração de aplicações. Essa linguagem é utilizada para criar especificações de alto nível que posteriormente serão transformadas em aplicações membro.

3. Uma ferramenta gráfica de auxílio e um gerador de aplicações por compilação.

No trabalho de Kulesza et al. (2007) é apresentada uma abordagem baseada em modelos para criação de arquiteturas de LPS orientadas a aspectos de fácil acoplagem/desacoplagem de features. Nessa abordagem, a etapa da engenharia de domínio é baseada na construção de três modelos (Kulesza et al., 2005):

- Modelo de Features: Representa as features da família de produtos, como definido em Czarnecki e Eisenercker (2002). Entretanto, na abordagem proposta por Kulesza et al. (2007), o modelo de features é estendido para permitir a representação de features transversais e a inclusão de *Extension Join Points* (Kulesza et al., 2006a,b). *Extension Join Points* (EJP's) são pontos de entrecorte bem definidos na LPS que permitem não só a composição de features opcionais e alternativas, mas também a evolução da arquitetura utilizando aspectos de extensão.

- Modelo de Arquitetura: É formado por um conjunto de componentes que agregam diferentes elementos de implementação da LPS, tais como, classes, interfaces, aspectos e gabaritos. O objetivo principal do modelo de arquitetura é criar uma representação desses elementos de implementação e relacioná-los com as variabilidades da linha de produtos.

- Modelo de Configuração: Define um conjunto de regras para o mapeamento entre as features da LPS e seus respectivos elementos de implementação. O modelo de configuração é utilizado para decidir quais componentes devem ser instanciados e quais adaptações devem ser realizadas considerando um determinado membro da LPS.

Essa arquitetura baseada em modelos visa facilitar a automatização da etapa de instanciação de produtos. A ferramenta GenArch (Cirilo et al., 2007) é um gerador de produtos para arquiteturas de LPS baseadas em modelos. Esse gerador possui um módulo de importação responsável pela geração automática dos modelos de features, de arquitetura e de configuração. A geração dos modelos de features e de configuração é feita através da leitura de marcações especiais que identificam as features existentes e seus respectivos elementos de implementação. A geração do modelo de arquitetura é baseada no *parsing* da árvore de diretórios do local onde estão armazenados os elementos de implementação da linha de produtos.

A ferramenta também possui um módulo de derivação de membros (geração de produtos) que recebe como entrada uma instância do modelo de features, isto é, um conjunto específico de elementos de implementação que satisfaça os requisitos do produto. Em seguida são processados os

gabaritos relacionados pelo modelo de configuração com as features selecionadas. Finalmente, é criado um novo projeto Java/Eclipse[5] contendo todos os elementos de implementação do produto gerado. A Figura 3.6 apresenta uma visão geral da arquitetura da ferramenta GenArch.

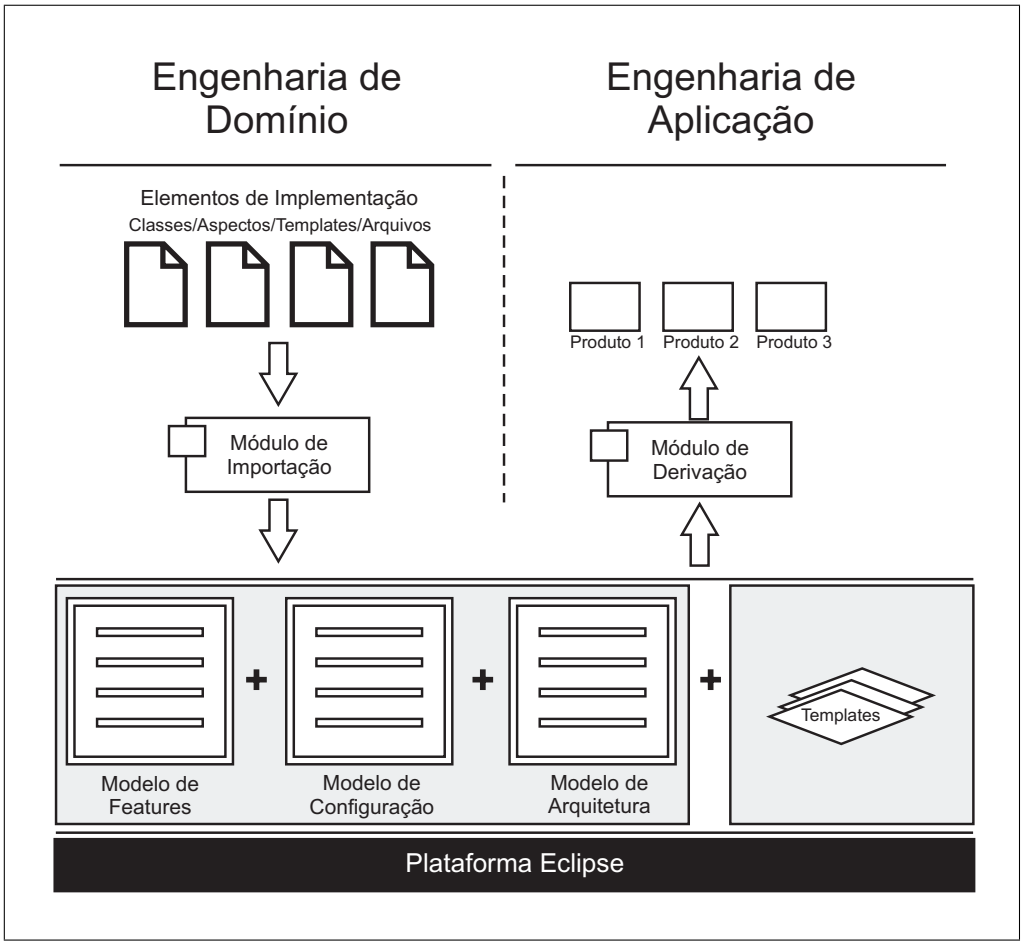

Figura 3.6: Arquitetura da ferramenta GenArch (Cirilo et al., 2007)

3.8 Considerações Finais

Neste capítulo foram abordados os principais conceitos relacionados com a programação orientada a aspectos, a linguagem AspectJ e as abordagens que buscam integrar aspectos e linhas de produtos de software.

A programação orientada a aspectos fornece mecanismos para o encapsulamento dos interesses transversais, o que ajuda a evitar problemas de entrelaçamento e espalhamento de código e aumenta a separação de interesses no sistema. Foram mostradas propostas para a integração dos conceitos provenientes da POA no processo de desenvolvimento de LPS. Essas abordagens foram analisadas e classificadas quanto às suas linhas de pesquisa.

[5]Eclipse Project: `http//:www.eclipse.org`.

Foi constatado que a aplicação de conceitos da POA no desenvolvimento de linhas de produtos é viável e útil. Não é possível afirmar que a POA é a melhor técnica a ser combinada com a engenharia de linhas de produtos, mas pode-se concluir que essa combinação com aspectos: (i) soluciona problemas existentes na modularização de features transversais, (ii) permite a inclusão, remoção e alteração de features com pouco impacto sobre o código-base da arquitetura e (iii) facilita a combinação entre features comuns e variantes durante o processo de instanciação de membros da LPS.

Comparando as pesquisas apresentadas neste capítulo, percebe-se que não são exploradas formas de reúso de features transversais em diferentes linhas de produtos. Dada a sua natureza ortogonal, uma mesma feature transversal poderia ser reusada em diversos domínios ajustando apenas o conjunto de pontos de junção, evitando assim, o desenvolvimento desnecessário dos mesmos interesses.

Nesse sentido, este trabalho de mestrado apresenta um processo de desenvolvimento de LPS e geração de produtos formados por features de um domínio-base incrementadas por features transversais já implementadas em outros domínios. Para a fase de geração, foi implementada uma extensão de um gerador já existente, o gerador de aplicações configurável Captor (Shimabukuro, 2006), que é apresentado em detalhes no próximo Capítulo.

CAPÍTULO

4

Gerador de Aplicações Configurável Captor

4.1 Considerações Iniciais

O gerador de aplicações configurável Captor (Shimabukuro, 2006) tem por objetivo facilitar a geração de artefatos de software em diferentes domínios. Para realizar a geração de uma família de produtos no Captor, o engenheiro troca o processo de desenvolvimento da LPS a partir do zero, por um processo de configuração de domínio. O Captor permite a definição de domínios utilizando um tipo próprio de especificação LMA (Ver Seção 2.5.3), baseado no preechimento de formulários. A ferramenta também possui um repositório para os artefatos fixos e variáveis (gabaritos) de cada domínio que são utilizados durante a geração dos produtos. Vários tipos de artefatos de software podem ser gerados pelo Captor, entre eles código, documentação e testes.

Neste Capítulo é apresentado em detalhes o gerador Captor, juntamente com os processos de engenharia de domínio e aplicação propostos por Shimabukuro (2006). O Capítulo está organizado da seguinte forma: na Seção 4.2 é apresentado brevemente um processo de desenvolvimento de software com apoio de geradores de aplicação configuráveis. Na Seção 4.3 é apresentado o Captor, sua arquitetura e suas principais funcionalidades.

Na Seção 4.4 são apresentadas as atividades do processo de engenharia de domínio, no qual a ferramenta Captor é configurada para novos domínios, na Seção 4.5 são apresentadas as atividades da etapa de engenharia de aplicações na qual o gerador é utilizado para produzir artefatos. Na Seção 4.6 é mostrado um exemplo passo-a-passo do processo de configuração e geração de produtos de um domínio. Por fim, na Seção 4.7, são apresentadas as considerações finais deste Capítulo.

4.2 Utilização de Geradores Configuráveis em Linhas de Produtos

O principal fator a ser considerado antes da criação de uma linha de produtos é o custo envolvido no desenvolvimento de uma arquitetura completa *versus* a economia alcançada após produção de diversos membros da família de produtos (Neighbors, 1984). É possível diminuir os custos de produção utilizando geradores de aplicações durante a etapa de instanciação e caso seja adotado um gerador configurável (Ver Seção 2.6), pode-se trocar o desenvolvimento de uma nova ferramenta a partir do zero, pelo processo de configuração de um novo domínio no gerador configurável, reduzindo o tempo total de implementação da linha de produtos.

Caso não seja possível utilizar nenhum gerador configurável disponível, pode-se analisar a viabilidade de construção de um novo gerador configurável levando em conta a reutilização desse gerador por outras equipes ou a possibilidade de configuração para novos domínios de linhas de produtos que venham a ser desenvolvidas pela organização. Se o resultado da análise aprovar o desenvolvimento de um novo gerador configurável, deve-se estudar previamente diversos geradores de aplicação específicos, isolar suas características em comum e projetar uma arquitetura flexível para incorporar diversos domínios. Muitas características estão presentes em diversos geradores específicos e podem ser reusadas independentemente do domínio. A idéia principal é implementar no gerador configurável as funções básicas necessárias para qualquer domínio e um módulo que recebe como entrada a especificação de um determinado domínio e automaticamente gera artefatos desse domínio, como código-fonte, casos de teste e documentação.

O processo de configuração e utilização de um gerador configurável é ilustrado na Figura 4.1 (Shimabukuro, 2006). Basicamente, esse processo é dividido em três etapas: configuração do domínio, definição de uma aplicação específica e geração da aplicação específica.

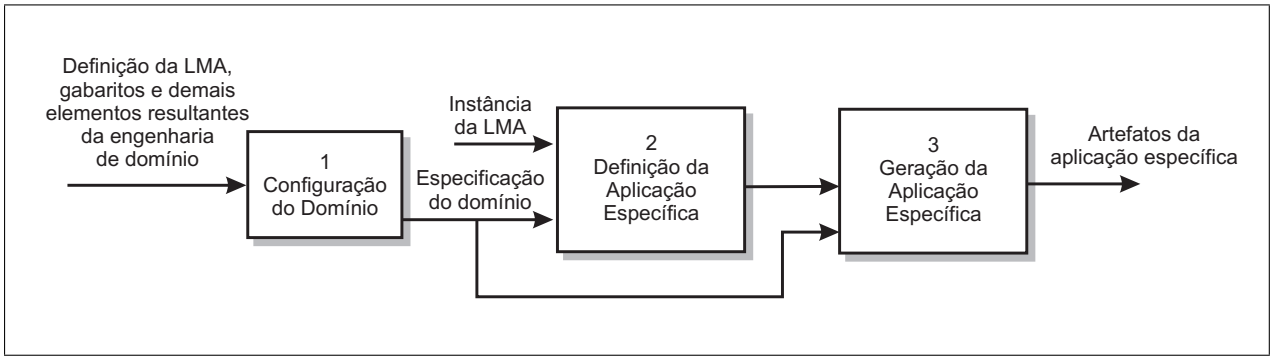

Figura 4.1: Configuração e utilização de geradores configuráveis (Shimabukuro, 2006)

Um gerador configurável deve ser projetado de forma a ser flexível o suficiente para suportar diferentes domínios. Logo, ele pode possibilitar a criação de uma LMA para cada novo domínio, bem como o armazenamento dos artefatos produzidos durante a engenharia de domínio (Passo 1 na Figura 4.1). O gerador configurável também deve permitir que o engenheiro de domínio defina

um mapeamento entre os gabaritos e seus respectivos elementos de implementação de acordo com o domínio configurado e a instância da LMA fornecida.

Na etapa de definição da aplicação específica (Passo 2), o engenheiro de aplicações deve fornecer para o gerador uma especificação da aplicação concreta desejada, isto é, uma instância da LMA do domínio configurado. Finalmente, na etapa de geração da aplicação específica (Passo 3) o gerador deve validar a consistência da instância fornecida e então gerar os artefatos desejados baseado no mapeamento entre gabaritos e features definido pelo engenheiro de domínio na primeira parte do processo.

4.3 Ferramenta Captor

A ferramenta Captor (Shimabukuro, 2006; Shimabukuro et al., 2006) é um gerador de aplicações configurável desenvolvido com a linguagem de programação Java e um conjunto de bibliotecas de componentes reutilizáveis e frameworks externos fornecidos pelo projeto Jakarta (Jakarta, 2008) e pela Sun Microsystems (Java, 2008). O Captor utiliza a abordagem de geração por composição, isto é, a geração de produtos é feita a partir da composição de gabaritos implementados. A especificação da LMA deverá determinar quais gabaritos serão usados para gerar um produto particular (Weiss e Lai, 1999). A arquitetura da ferramenta é apresentada na Figura 4.2.

A utilização da ferramenta Captor pode ser dividida em duas fases: engenharia de domínio e engenharia de aplicações, que são apresentada nas Seções 4.4 e 4.5, respectivamente. Na primeira, o Captor é alimentado com os domínios. Essa fase é realizada predominantemente pelo engenheiro de domínio. Na segunda, os produtos da LPS são gerados, sendo essa atividade realizada tipicamente por um engenheiro de aplicações.

4.4 Engenharia de Domínio

Na etapa de engenharia de domínio, um domínio de aplicação é analisado (a informação sobre o domínio é identificada, capturada e organizada com o objetivo de fazê-la reusável durante a criação de novos sistemas (Diaz, 1990)) e artefatos de software reutilizáveis são projetados e implementados para apoiar o desenvolvimento de aplicações nesse domínio. É criada uma linguagem de modelagem de aplicações (LMA) para representar o domínio (ver Seção 2.5.3) e o gerador é configurado para esse domínio da linha de produtos. O gerador Captor possui um subsistema, denominado “Configuração do Domínio” (Figura 4.2), responsável por organizar e manter todas essas informações necessárias sobre o domínio. O processo de engenharia de domínio na ferramenta Captor divide-se em:

1. Criação de uma linguagem de modelagem de aplicações. A LMA utilizada pela ferramenta pode ser criada definindo uma estrutura de formulários em forma de uma árvore. Cada formulário contém um conjunto de campos, representados por elementos gráficos (`widgets`),

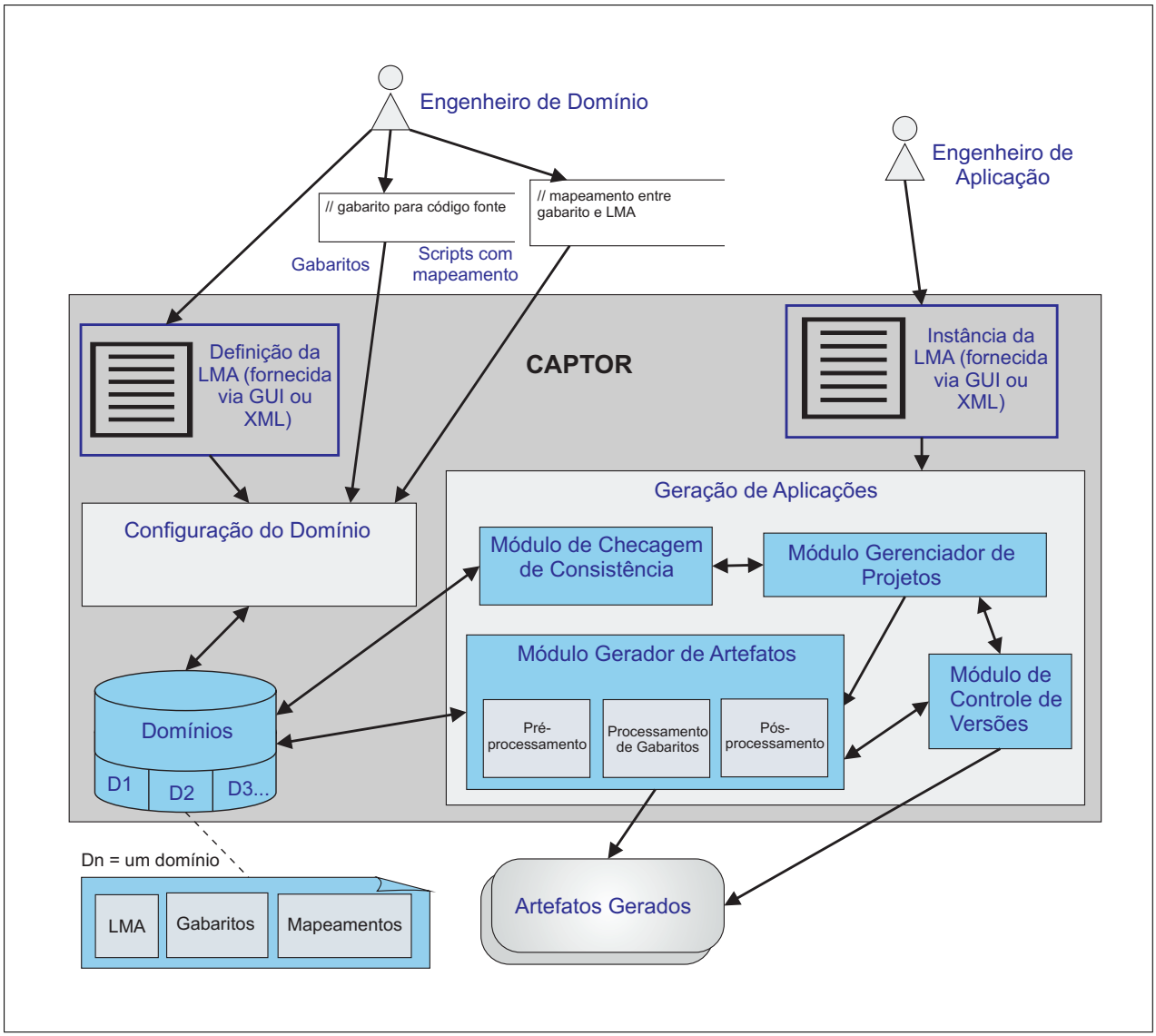

Figura 4.2: Arquitetura do Captor (Shimabukuro et al., 2006)

tais como: caixas de texto, caixas de seleção, etiquetas e tabelas que podem ser utilizadas pelo engenheiro de domínio para representar abstrações das features do domínio.

2. Implementação dos artefatos reutilizáveis do domínio. O engenheiro de domínio deve projetar e implementar todos os artefatos fixos dos domínios que são gerados automaticamente por meio de gabaritos. O gerador Captor é capaz de gerar aplicações independentemente da linguagem de programação escolhida, logo não há restrições quanto à linguagem, paradigma ou tecnologias que devem ser utilizadas para a implementação do domínio.

3. Criação dos gabaritos. Os gabaritos utilizados pelo Captor para transformar dados de uma especificação em artefatos concretos devem ser escritos utilizando a linguagem de transformação XSL (XSL, 2008). Essa linguagem de transformação de gabaritos é padronizada pela entidade W3C [1] e permite asserções condicionais, interações e composição de gabaritos. A

[1] World Wide Web Consortium (W3C): `http://www.w3.org/`.

sintaxe e a semântica detalhada dessa linguagem estão fora do escopo deste trabalho, mais instruções sobre XSL podem ser encontradas no site: `http://www.w3c.org/xslt`, livros e em manuais disponíveis.

4. Definição do mapeamento entre a LMA e os gabaritos. O engenheiro de domínio deve criar um arquivo de mapeamento que será interpretado pelo gerador durante o processamento dos gabaritos. Esse arquivo deve ser escrito na linguagem MTL (*Mapping Transformation Language*) proposta por Shimabukuro (2006) e deve conter regras que serão aplicadas para a seleção dos gabaritos de acordo com a instância LMA fornecida.

Na Figura 4.3 é mostrada a interface utilizada durante a criação de um novo domínio. O módulo de Configuração do Domínio realiza a leitura dos arquivos de configuração de um domínio e monta uma estrutura de dados que represente o domínio escolhido pelo usuário e que possa ser utilizada pelos outros módulos do Captor.

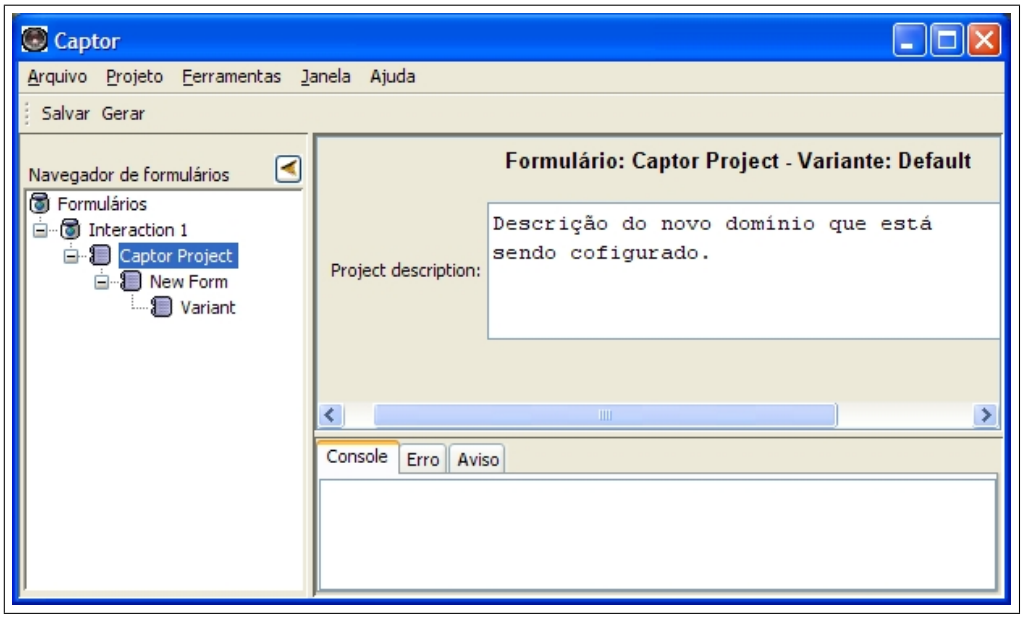

Figura 4.3: Interface utilizada para configuração de um novo domínio

4.5 Engenharia de Aplicações

Na etapa de engenharia de aplicações, o gerador é utilizado para a instanciação de produtos. Isso envolve atividades de engenharia de requisitos (coleta das necessidades dos clientes e criação do documento de requisitos do sistema), criação do modelo da aplicação (definição de uma instância da LMA que representa uma aplicação particular), geração dos artefatos (parte deles ou todos), implementação das funcionalidades não cobertas pelo gerador (se necessário), testes e documentação. O subsistema "Geração de Aplicações" (Figura 4.2) implementa todas as funcionalidades relacionadas com a especificação e geração de novas aplicações.

A ferramenta Captor utiliza o conceito de projetos para armazenar informações sobre as aplicações específicas geradas. Para cada nova aplicação, o engenheiro de aplicações deve criar um projeto. Dentro do arquivo de projeto são reunidas informações sobre a aplicação específica, como o nome e o domínio da aplicação, e o local em disco para o armazenamento dos artefatos gerados. O módulo de gerenciamento de projetos é responsável pela manutenção dos projetos criados no Captor, ou seja, criação, manutenção e remoção de projetos. Na figura Figura 4.4 é mostrada a interface utilizada para a criação de um novo projeto no gerador Captor.

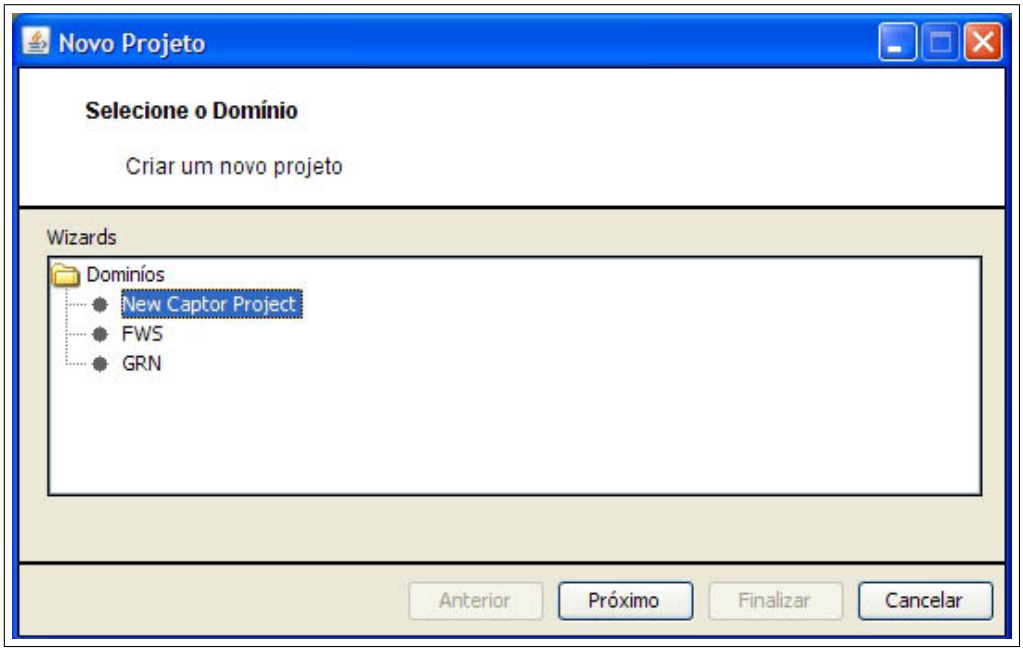

Figura 4.4: Interface utilizada para a criação de um novo projeto no gerador Captor

A interface do gerador é apresentada na forma de janelas para o usuário, onde cada janela agrupa um conjunto de funções, sendo elas: principal, criação de novo projeto, propriedades do projeto, ajuda, ajuste de preferências e validação da instância da LMA. A janela principal é dividida em dois painéis durante a engenharia de aplicações, essa janela é apresentada na Figura 4.5. No painel à esquerda (letra A da Figura 4.5) é apresentado um conjunto de formulários organizados hierarquicamente em forma de árvore representando a LMA do domínio configurado. A interação com o engenheiro de aplicações é feita através do preenchimento dos elementos de formulário (caixas de texto, caixas de seleção, listas, entre outros) existentes. Os elementos contidos dentro de cada fomulário da àrvore são mostrados no painel da direita na janela principal (letra B da Figura 4.5). A união de todos os dados fornecidos pelo engenheiro de aplicações define a instância da LMA que será utilizada posteriormente para a geração dos artefatos da nova aplicação específica.

Antes da etapa de geração, o Módulo de Checagem de Consistência deve ler o conjunto de informações, preenchidas diretamente dentro dos formulários ou contidas em arquivos de projeto no formato XML[2], e validar essas especificações de acordo com as regras de negócio do domínio.

[2]Extensible Markup Language (XML), `http://www.w3.org/XML/`

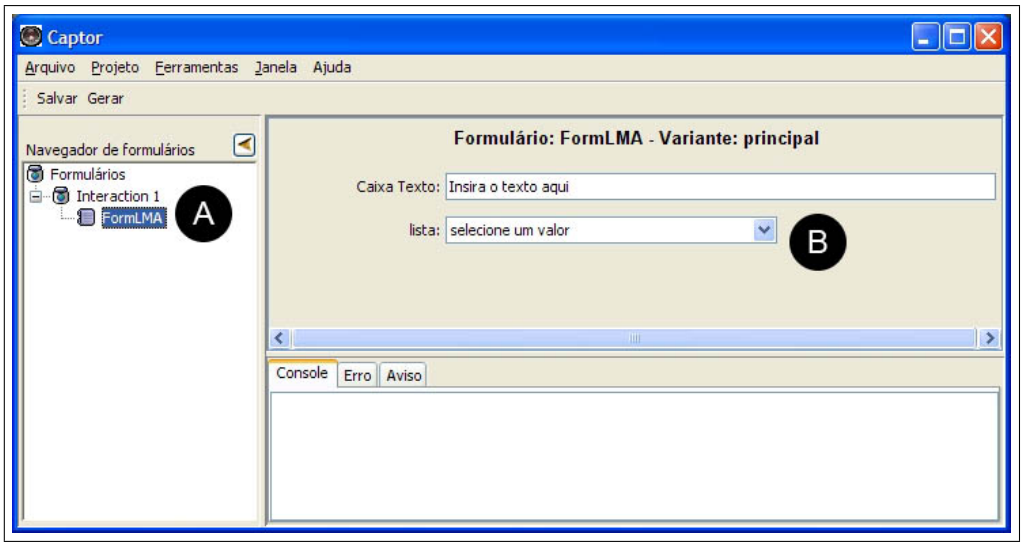

Figura 4.5: Interface utilizada durante a especificação de uma nova aplicação

Se a ferramenta detectar uma inconsistência na especificação, o módulo emitirá um aviso de erro para o usuário.

Caso a validação seja executada com sucesso, o módulo Gerenciador de Artefatos tem a responsabilidade de selecionar os gabaritos do domínio envolvido que devem ser gerados e utilizar seu sub-módulo de Processamento de Gabaritos para produzir os artefatos da aplicação. O processamento dos gabaritos é feito com base nos mapeamentos definidos durante a engenharia de domínio e no arquivo com os dados da especificação (instância LMA). Se o artefato de saída já tiver sido gerado em uma versão antiga do projeto, o Módulo de Controle de Versões deve analisar o arquivo gerado, buscar por modificações manuais feitas pelo usuário diretamente nos artefatos gerados anteriormente usando zonas de segurança e incluí-las novamente no artefato que esta sendo gerado atualmente.

Zonas de segurança ou marcações (Greenfield e Short, 2004) são seções delimitadas por comentários especiais onde o desenvolvedor pode inserir código manual sem perder as modificações com o processo de re-geração de artefatos. Durante o processo de re-geração, a ferramenta deve ler o arquivo de saída gerado e modificado anteriormente, recuperar as informações delineadas pelas marcações especiais e inserir essas informações nos arquivos re-gerados durante o novo processo de geração.

4.6 Exemplo de Uso: LPS GestorPsi

No mestrado de Shimabukuro (2006), o Captor foi configurado para três domínios diferentes: gestão de recursos de negócios (por meio do framework GREN (Braga e Masiero, 2002b) e da linguagem de padrões GRN (Braga et al., 1999a)), bóias náuticas (Weiss e Lai, 1999) e persistência de objetos. Ao longo deste trabalho o Captor foi configurado para mais um domínio: sistemas de informação para clínicas de psicologia (Pacios, 2006). Nos dois primeiros casos (GRN e bóias),

ao utilizar o Captor obtém-se aplicações completas que podem ser executadas imediatamente. No terceiro caso (persistência), o Captor gera scripts SQL para a criação de um banco de dados relacional. Esse domínio não tem relação com o framework de persistencia de dados proposto por Camargo (2006). No último caso, o Captor gera diferentes membros por meio do acoplamento/desacoplamento das features opcionais (implementadas como aspectos) sobre o código-base.

A seguir será ilustrado o processo de configuração utilizando o exemplo do domínio de clínicas de psicologia. O processo de configuração e geração usando a ferramenta Captor é o mesmo mostrado na Figura 4.1. Após a análise e implementação do domínio, uma LMA é definida e os gabaritos são criados e mapeados para a LMA. Após a configuração do domínio, o Captor está pronto para ser usado pelo engenheiro de aplicações, que deve prover a instância da LMA e invocar o gerador de artefatos para obter a aplicação específica.

4.6.1 Análise e Implementação do Domínio

A linha de produtos no domínio de sistemas de clínicas de psicologia foi desenvolvida por Pacios (2006) em sua pesquisa de mestrado. A implementação desse domínio foi realizada em paralelo com o projeto GestorPsi. O GestorPsi nasceu da demanda pela criação de um sistema informatizado que possa trazer a psicologia para a era digital, possibilitando a criação de métodos padronizados para:

- Registro de informações clínicas;
- Gerenciamento de informações administrativas e financeiras;
- Gerenciamento da prestação de serviços psicológicos;
- Registro de diagnósticos, avaliações e resultados.

A realização do projeto GestorPsi viabilizou o acesso às informações necessárias para o estudo do domínio, como material bibliográfico, sistemas existentes, especialistas e usuários. Para a compreensão do funcionamento das clínicas de psicologia, bem como dos sistemas de informação nelas utilizados, foram estudadas três clínicas existentes: Clínica Escola da PUC-SP "Ana Maria Poppovic", Clínica Escola da Faculdade Paulistana e o Instituto de Psicologia Comportamental de São Carlos.

Os artefatos produzidos nas análises individuais foram estudados e os modelos de features das três clínicas foram comparados para produzir o modelo de features do domínio. O modelo de features resultante é apresentado na Figura 4.6. Dentre todas as features identificadas, as seguintes features foram desenvolvidas para linha de produtos: `Paciente`, `Prontuario`, `Terapeuta`, `Sala`, `Agendamento` e `Serviço`.

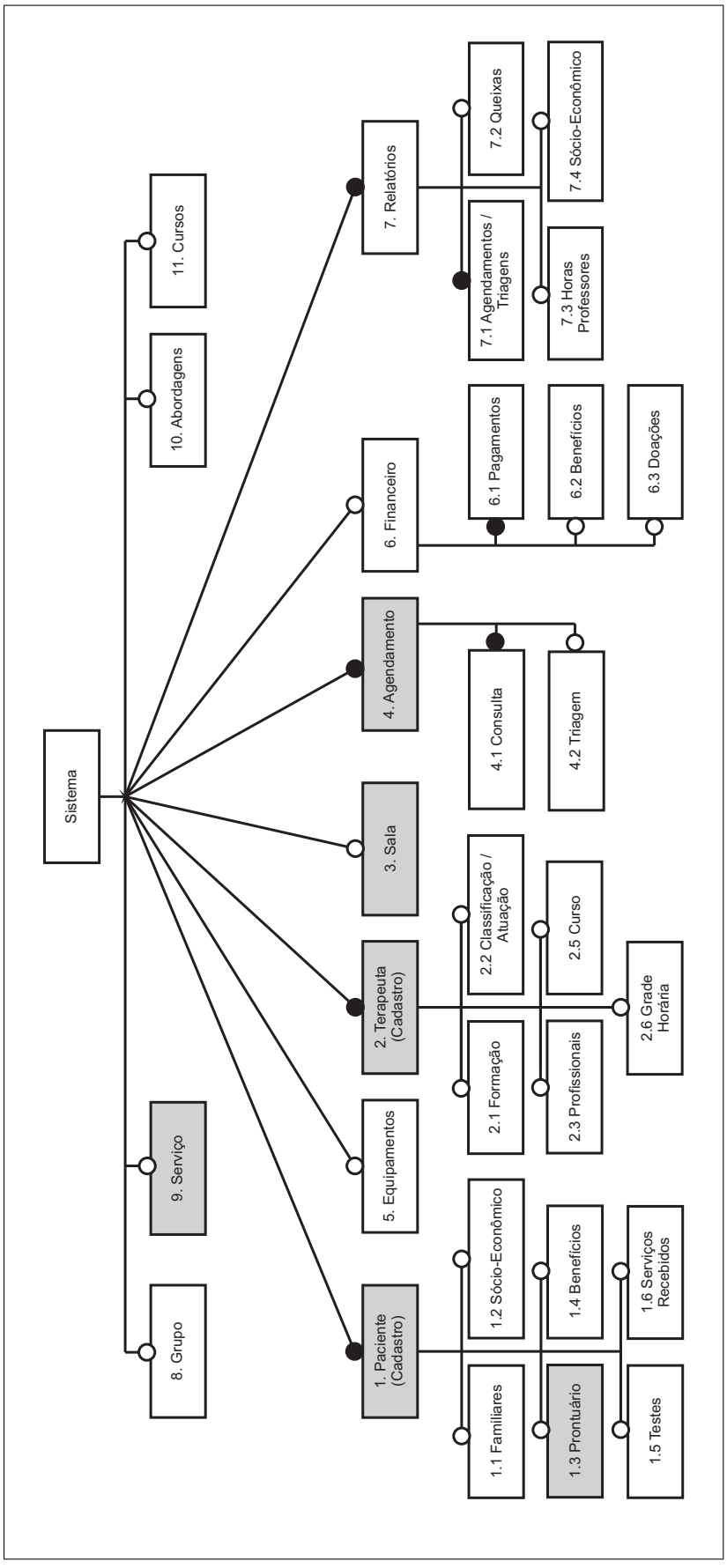

Figura 4.6: Modelo de Features do domínio de Clínicas de Psicologia (Pacios, 2006)

Não foram desenvolvidas todas as features já que, por ser apresentado como um estudo de caso, o sistema não tem o objetivo de ser aplicado na prática, o que não afeta o estudo das técnicas de desenvolvimento e composição das features. As features selecionadas aparecem em destaque na Figura 4.6 e são descritas detalhadamente na Tabela 4.1.

Nome da Feature	**Descrição**
Paciente	Representa o registro dos dados pessoais de um paciente. Abrange documentação, endereço, contatos e descrição pessoal.
Prontuário	Abstração do conjunto de documentos e informações referentes a um paciente.
Terapeuta	Representa o profissional encarregado do tratamento de um ou mais pacientes.
Sala	Local onde são realizadas as consultas dos pacientes.
Agendamento	Reserva prévia de dia e hora para consultas e atendimentos.
Serviço	Gerencia dos dados referentes ao serviço que o paciente está recebendo da clínica.

Tabela 4.1: Descrição das features implementadas para a linha de produtos

Conforme pode ser visto na Figura 4.6, as features com círculos preenchidos são as features obrigatórias que formam a base da linha de produtos e as demais features, são features opcionais. Seguindo a abordagem apresentada na seção 3.6.3 as features obrigatórias foram implementadas como classes Java tradicionais e a features opcionais como aspectos da linguagem AspectJ capazes de serem acoplados ou desacoplados dos produtos em tempo de compilação. Mais detalhes sobre a implementação dos artefatos da linha de produtos podem ser encontrados em Pacios (2006).

4.6.2 Configuração do Domínio

Como exemplo, será utilizada uma versão simplificada do domínio de clínicas de psicologia. Nesse exemplo, serão consiseradas as features obrigatórias `Paciente`, `Terapeuta` e `Agendamento`, e as features opcionais `Sala`, `Serviço` e `Prontuário`. Ao final do processo de configuração do domínio, a ferramenta Captor será capaz de gerar aplicações formadas por todas as features obrigatórias e pelas features opcionais selecionadas pelo engenheiro de aplicações.

Para configurar o domínio de clínicas de psicologia no gerador Captor, deve-se primeiro criar uma LMA que represente as suas variabilidades. As features obrigatórias, por serem fixas, não aparecem na LMA, pois elas são incluídas no sistema gerado independente de sua especificação pelo engenheiro de aplicações. Logo, apenas as features opcionais são modeladas na LMA.

Visto que a adição e remoção de features pode ser realizada pelo combinador de aspectos (*weaver*) da linguagem AspectJ, é preciso especificar como os produtos diferem uns dos outros em termos dos aspectos que serão incluídos de acordo com as variabilidades desejadas. Dessa forma, cada feature opcional pode ser mapeada como um formulário do Captor com campos de dados que

indiquem se a feature está ou não presente no produto final. Esses formulários também podem ser posteriormente usados pelo engenheiro de aplicações para informar detalhes da aplicação, como o nome da clínica que irá utilizar o sistema gerado, por exemplo.

Não existe uma solução única para a modelagem da LMA, pois um mesmo domínio pode ser coberto por diferentes LMA's. Uma possível LMA para o domínio de clínicas de psicologia consiste no conjunto de formulários representando cada feature opcional da linha de produtos, além de informações adicionais sobre precedência e dependência entre features. A organização hierárquica dos formulários para o domínio de clínicas de psicologia é apresentada na Figura 4.7.

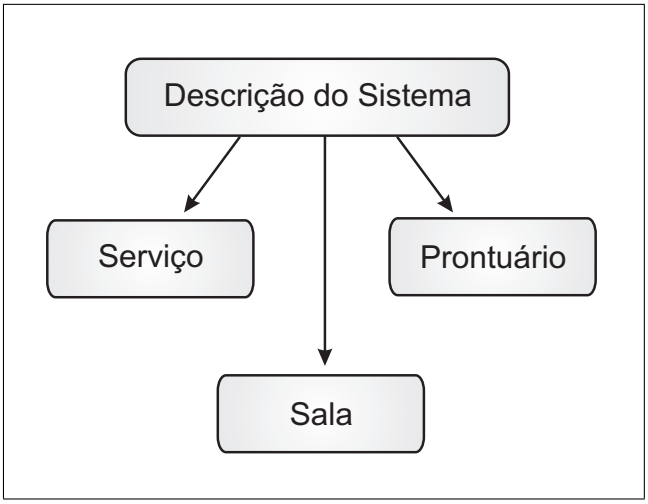

Figura 4.7: Árvore de formulários para o domínio de clínicas de psicologia

Após a definição da estrutura hierárquica, o engenheiro de domínio deve definir a forma como o formulário deve se apresentar ao usuário, ou seja, quais campos cada formulário contém e quais os valores válidos dos dados inseridos nesses campos. Cada formulário pode conter um ou mais elementos gráficos genéricos, tais como caixas de texto, tabelas, etc.

Na Figura 4.8 é mostrada a configuração do formulário utilizado para representar a feature opcional `Serviço`. Nesse formulário é incluído um elemento gráfico (Letra B na Figura 4.8) para indicar se a feature deve ou não ser incluída no produto. O usuário pode navegar pelos formulários por meio do painel de navegação de formulários (Letra A na Figura 4.8). É disponibilizado um menu *popup* para inserção de formulários filhos, remoção de formulários ou alteração da ordem de visualização.

O engenheiro de domínio também pode definir regras de validação para cada elemento gráfico, utilizando parâmetros específicos do formulário. Por exemplo, para garantir que um determinado valor seja composto somente por letras e números, pode-se utilizar a expressão regular "[a-zA-Z-0-9]+" no parâmetro `regexp` de uma caixa de texto, conforme mostrado na Figura 4.9.

O segundo passo para a configuração do domínio é a implementação dos gabaritos que serão transformados em artefatos concretos, tais como, código-fonte, casos de testes e documentos. Nesse caso, a escolha da feature prontuário, por exemplo, implica na inclusão de um conjunto de

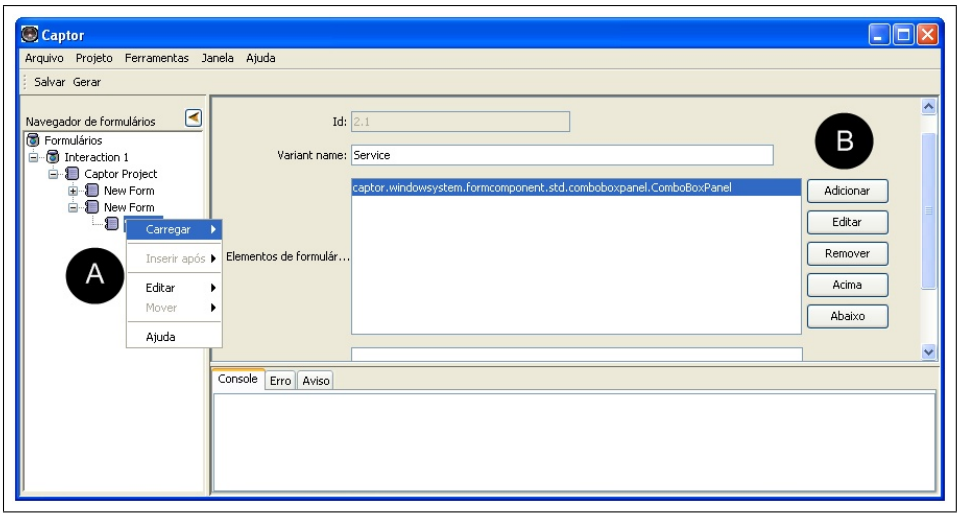

Figura 4.8: Interface do Captor utilizada para a criação de formulários

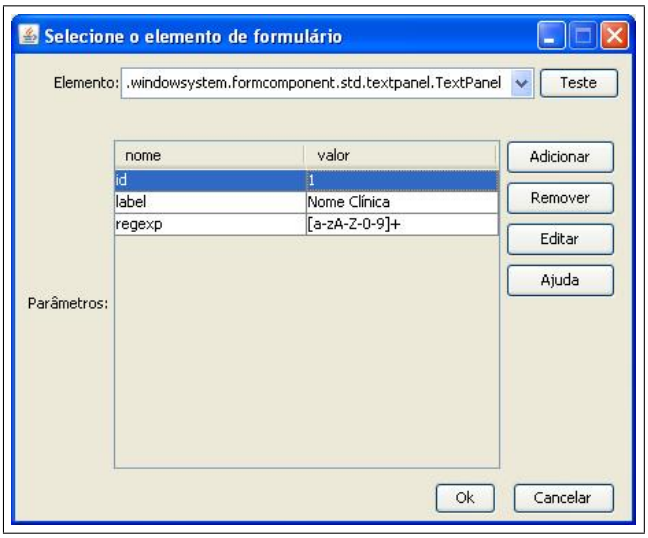

Figura 4.9: Configuração de regras de validação

aspectos responsáveis pela implementação dessa funcionalidade. Assim, o engenheiro de aplicações deverá informar quais features opcionais ele deseja incluir e essa informação será usada durante o processo de instanciação, para seleção dos aspectos necessários. O mapeamento completo entre features e seus respectivos pacotes de implementação para a linha de produtos de clínicas de psicologia pode ser encontrado em Pacios (2006).

Os gabaritos utilizados pelo Captor devem ser implementados na linguagem de gabaritos XSL (*Extensible Stylesheet Language*). Os detalhes sobre a sintaxe dessa linguagem estão fora do escopo deste trabalho e podem ser encontrados em XSL (2008). Na Figura 4.10 é visto um trecho de gabarito responsável pela especificação do nome da clínica dentro do produto gerado. O gabarito cria uma variável que realiza a leitura do valor fornecido pelo engenheiro de aplicações (linhas 1 a 4). O gabarito possui uma marcação especial (*placeholder*) na linha 16. Esse marcador será substituído pelo valor concreto durante a geração da classe `FramePrincipal`.

```
<xsl:variable name="nomeClinica">
    <xsl:value-of select="/formsData/forms/form[@id='1.1']/
                                    data/textatt[@name='nomeClinica']"/>
</xsl:variable>

package principal;

public class FramePrincipal{

    public JFrame frame;

    /*
     * Create the GUI.
     */
    private void createAndShowGUI() {
        this.frame = new JFrame("<xsl:value-of select="$nomeClinica"/>");
        this.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.frame.addWindowListener(this);
        .
```

Figura 4.10: Exemplo de gabarito do domínio de clínicas de psicologia

Por último, o engenheiro de domínio deve criar um arquivo de mapeamento de transformação dos gabaritos. Esse arquivo é um *script* escrito na linguagem MTL (*Mapping Transformation Language*) proposta por Shimabukuro (2006). A linguagem MTL fornece comandos de transformação de gabaritos de acordo com especificação fornecida pelo Captor. Para o domínio de clínicas de psicologia, esse arquivo deve processar os gabaritos referentes às features opcionais aplicadas. Uma vez que as features obrigatórias estão presentes em todos os produtos, a ferramenta pode copiar diretamente o código-fonte dessas features para o diretório de saída. Essa cópia de artefatos é implementada no gerador utilizando mecanismos de pré e pós-processamento. Esses mecanismos facilitam operações de processamento dos artefatos antes e depois da geração do produto, tais como cópia, remoção, alteração e *backup* de arquivos. Além de classes Java e aspectos do domínio, também é possível criar gabaritos para a geração de *script* SQL de tabelas, casos de teste e documentação.

4.6.3 Geração de Produtos

Após a configuração do gerador Captor para o domínio de clínicas de psicologia, a ferramenta está pronta para gerar produtos nesse domínio. O processo de geração é iniciado quando o engenheiro de aplicações cria um novo projeto que irá armazenar a especificação da nova aplicação e fornecer para o Captor dados sobre as features que serão incluídas. Os formulários correspondentes à cada feature opcional devem ser preenchidos e em seguida, a ferramenta realiza a validação da especificação fornecida e, em caso de sucesso, gera automaticamente o sistema desejado. Em caso de falha na validação, a ferramenta disponibiliza um mecanismo de depuração de erros que informa para o engenheiro de aplicações detalhes dos erros encontrados.

A Figura 4.11 ilustra como os formulários criados durante a configuração de domínio são apresentados pelo Captor na interface utilizada pelo engenheiro de aplicações para a definição de uma

instância da LMA. O painel à esquerda mostra os formulários que representam as variabilidades do domínio, e o à direita representa uma feature específica sendo instanciada (feature `Sala`). O formulário `Descrição do Sistema` é definido como formulário raiz e possui relacionamento com os formulários `Sala`, `Serviço` e `Prontuário`.

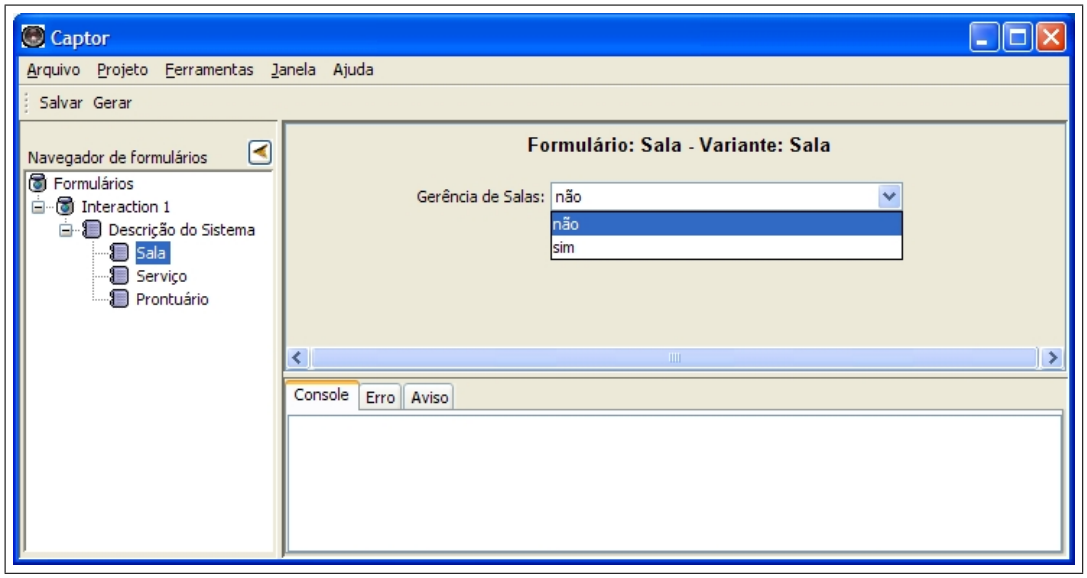

Figura 4.11: Interface utilizada para instanciação de membros do domínio de clínicas de psicologia

A informação fornecida será então combinada com *placeholders* presentes nos gabaritos. Durante fase de geração do produto, a ferramenta realiza algumas operações de pré e pós-processamento. O pré-processamento realiza a cópia dos artefatos fixos para o diretório de saída da aplicação e o pós-processamento é encarregado de compilar o código-fonte recém processado. O módulo de Controle de Versão deve verificar se código da aplicação possui zonas de segurança com modificações feitas manualmente em iterações passadas e aplicá-las novamente no código que está sendo gerado. A interface de um produto instanciado com as features opcionais `Sala` e `Serviço` é apresentada na Figura 4.12.

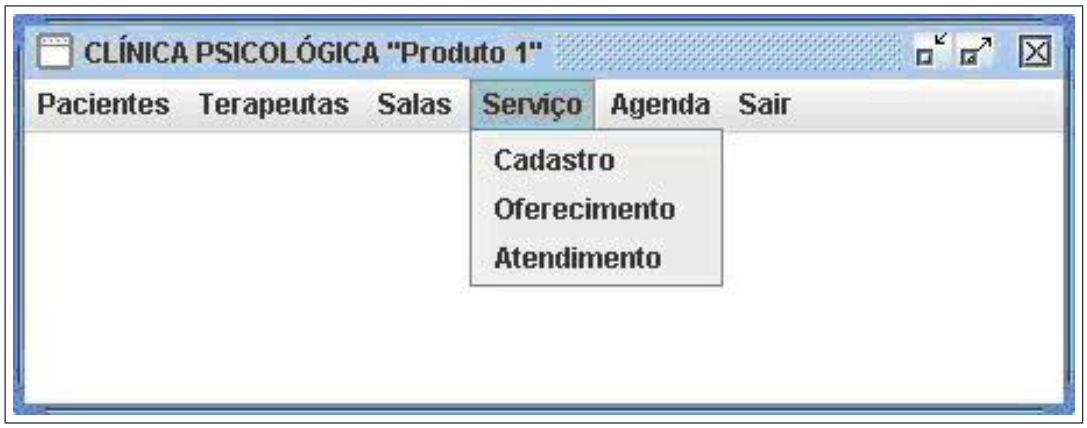

Figura 4.12: Janela principal do produto gerado pelo Captor

4.7 Considerações finais

Neste capítulo foi apresentado o gerador configurável Captor, por meio do qual é possível criar aplicações de domínios específicos. Com a utilização do Captor no desenvolvimento de software, pode-se aumentar a produtividade do desenvolvimento e a confiabilidade, pois diversos artefatos são gerados automaticamente com base na especificações de alto nível de abstração e o código produzido, diferentemente do código escrito manualmente, não é susceptível à introdução de erros humanos. O Captor é disponibilizado com licença *OpenSource* (GPL) e pode ser encontrado no endereço (URL) `http://www.labes.icmc.usp.br/captor`.

Após a configuração do Captor para o domínio GestorPsi, constatou-se que era possível utilizar a ferramenta para gerar os membros de uma linha de produtos formada por interesses transversais. Entretanto, na versão original da ferramenta, os conjuntos de pontos de junção e o entrecortes realizados pelos adendos precisam ser configurados manualmente pelo engenheiro de domínio e a partir daí só podem alterados pelo engenheiro de aplicações após a etapa de geração e diretamente no código produzido, ou seja, a abstração de alto nível disponibilizada não é suficiente para a modularização de interesses transversais existentes em um domínio de aplicação. Atualmente, existem poucas ferramentas que explorem os benefícios da utilização de interesses transversais em linhas de produtos, como a facilidade de acoplamento e desacoplamento de features, por exemplo. Essa foi uma das motivações do processo e da ferramenta apresentados nos Capítulos a seguir.

CAPÍTULO

5

Desenvolvimento de LPS apoiado por geradores de aplicações e aspectos

5.1 Considerações Iniciais

Neste capítulo é apresentado um processo de desenvolvimento de linhas de produtos apoiado pela utilização de POA e geradores de aplicações configuráveis (Ver Seção 2.6.2). Ele baseia-se nos processos de engenharia de domínio e de engenharia de aplicações apresentados no trabalho de Shimabukuro (2006), com o diferencial de explorar a facilidade de combinação de features transversais proporcionada pela POA. Nesse sentido, esta pesquisa propõe que as features transversais implementadas em um determinado domínio sejam reusadas em diversas linhas de produtos.

O Capítulo está organizado da seguinte forma. Na Seção 5.2 é discutida a utilização da programação orientada a aspectos no processo de engenharia de linha de produtos. Essa Seção discute sobre novas formas de modularização e reúso dos interesses transversais existentes em linhas de produtos. Também são apresentados os conceitos propostos para a engenharia de linha de produtos apoiada pela programação orientada a aspectos.

A Seção 5.3 descreve as atividades dos processos de engenharia de domínios e de aplicações, fazendo um paralelo entre as atividades tradicionais conhecidas e as novas atividades propostas nesta pesquisa. Por fim, na Seção 5.4 são apresentadas as considerações finais sobre o processo proposto.

5.2 Linhas de Produtos Orientadas a Aspectos e Domínios Transversais

Uma das motivações desta pesquisa é permitir o reúso ao longo de diversas linhas de produtos diferentes. Como exemplo, considere uma linha de produtos para sistemas hospitalares, outra para seguradoras e outra para bancos. Se analisarmos as features de cada uma dessas LPS's, notaremos que elas possuem diversas partes em comum, que poderiam ser isoladas e reutilizadas. Essas features podem ser tanto não-funcionais (por exemplo, controle de acesso, persistência dos objetos e concorrência), quanto funcionais (por exemplo, o sub-sistema financeiro ou de contabilidade).

Assim, propõe-se que sejam agrupadas em **domínios-base** as features (obrigatórias ou não) que definem o negócio propriamente dito (no caso do sistema hospitalar seriam as features como paciente, médico, internação, cirurgia, etc.). Domínios-base, também denominados domínios específicos, representam a parte central do produto gerado e são compostos de um núcleo de features comuns a todos os produtos da LPS e de um conjunto de features que variam de produto para produto.

Por outro lado, sugere-se que sejam agrupadas em **domínios-transversais** aquelas features que encapsulam comportamentos genéricos de um interesse transversal referentes a vários domínios-base, como persistência ou segurança, por exemplo. Assim como um domínio-base, um domínio transversal também possui features comuns e variáveis de produto para produto. As features de um domínio transversal são mais genéricas e possuem um alto grau de reúso em diferentes domínios-base.

Deve-se notar que a implementação de uma LPS para um domínio-base não é necessariamente feita apenas por classes e bibliotecas orientadas a objetos: como visto na Seção 4.5, um domínio-base pode conter aspectos em sua implementação para representar determinados comportamentos transversais internos. No entanto, no processo aqui proposto não se enfatiza esse possível uso de aspectos, mas destaca o uso de aspectos na implementação dos domínios transversais, em especial para features não-funcionais que entrecortam os domínios-base.

Assim, esta seção apresenta os novos conceitos do processo de engenharia de linha de produtos propostos neste trabalho com o objetivo de permitir um meio de composição entre features de um domínios-base e features de diversos domínios transversais.

Tanto nos domínios-base quanto nos domínios transversais há necessidade de instanciar a LPS para originar produtos específicos. Por exemplo, na Figura 5.1 o domínio transversal `T` origina os produtos `T1` e `T2` e o domínio-base `A` gera os produtos `A1` e `A2`. Uma linha de produtos de um domínio transversal deve fornecer mecanismos de composição para que as features transversais possam ser aplicadas sobre o maior número possível de domínios-base.

A instanciação de membros de um domínio transversal, por si só não produz aplicações funcionais, isto é, os produtos de um domínio transversal são dependentes da existência de um código-base. Quando compostos com domínios-base, os produtos de um domínio transversal introduzem

novas features, com pouca ou nenhuma alteração no código-base. Assim, a derivação de produtos de domínios transversais possui duas etapas distintas: instanciação e composição. A etapa de instanciação ocorre conforme o processo convencional em que é realizada a especialização de variabilidades, como descrito na Seção 2.5. A etapa de composição, por sua vez, consiste na concretização dos pontos de junção abstratos e o acoplamento dos interesses transversais com o código-base.

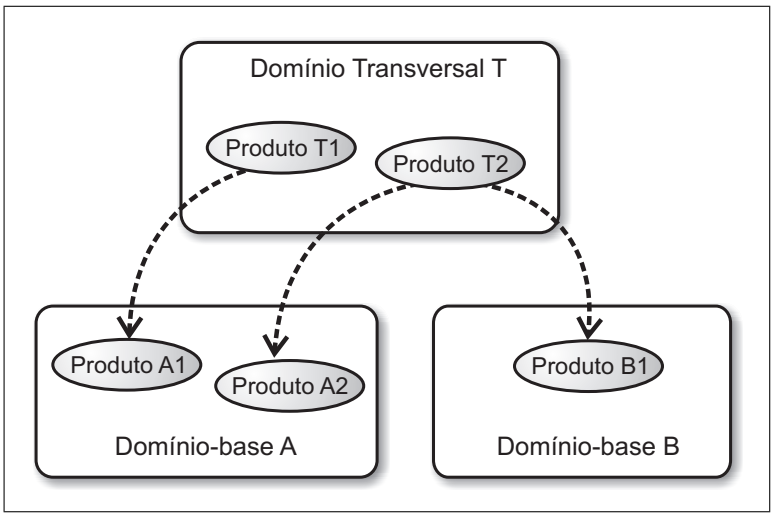

Figura 5.1: Relacionamento entre domínios-base e domínios transversais

5.2.1 Pontos de Junção Abstratos

Pontos de junção abstratos (PJA's) são os elementos responsáveis pelo acoplamento entre as features transversais de um domínio transversal com os produtos de um domínio-base. Esse conceito é baseado na noção de aspectos abstratos, que definem pelo menos um *pointcut* abstrato que deve ser sobreposto em tempo de compilação. Uma vez que os pontos a serem entrecortados podem variar de acordo com o produto, o engenheiro de um domínio transversal não pode prever os valores dos pontos de junção para todos os possíveis produtos do(s) domínio(s)-base. Por outro lado, ao implementar os aspectos do domínio transversal utilizando PJA's o engenheiro de domínio delega para o engenheiro de aplicações a responsabilidade de fornecer os valores concretos durante a instanciação de um novo produto. O objetivo dos PJA's é tornar as features transversais menos dependentes de contexto e mais reusáveis.

5.2.2 Pontos de Junção Pré-Definidos

A concretização dos PJA's não é necessariamente realizada durante a etapa de instanciação de produtos. O engenheiro de um domínio-base pode optar por configurar um valor fixo para um PJA, denominado *ponto de junção pré-definido* (PJP), de acordo com o domínio-base que será entrecortado. Em algumas situações, todos os produtos de um domínio-base compartilham o

mesmo valor concreto para algum PJA. Nesses casos, pode-se relacionar um ponto de junção pré-definido no domínio-base com o PJA do domínio transversal, evitando assim, que o engenheiro de aplicações tenha que fornecer o mesmo valor concreto para todos os produtos do domínio-base.

Dependendo do domínio transversal, a concretização de um determinado PJA pode exigir que o engenheiro de aplicações conheça detalhes de implementação do domínio-base, como as assinaturas dos métodos que serão entrecortados. É possível criar pontos de junção pré-definidos para esses PJA's ainda durante a etapa de engenharia de domínio, e assim poupar o engenheiro de aplicações do preenchimento desses PJA's. Entretanto, essa alternativa reduz a flexibilidade de composição das features transversais, visto que o valor pré-definido é utilizado invariavelmente em todos os produtos instanciados. Assim, uma solução é permitir que o engenheiro de aplicações forneça o valor concreto do PJA durante a instanciação e quando nenhum valor for preenchido, utilizar o ponto de junção pré-definido como valor padrão.

5.2.3 Combinação entre Domínios

O processo tradicional de engenharia de linhas de produtos prevê a instanciação de produtos a partir de um único domínio de aplicação isolado. Entretanto, graças ao surgimento da POA e do conceito de domínios transversais, é possível instanciar um produto formado por artefatos de múltiplos domínios. Para tal, o processo de engenharia de aplicações deve suportar a composição de artefatos provenientes de domínios distintos, denominada neste trabalho como *combinação de domínios*. O produto final de uma combinação de domínios terá funcionalidades provenientes de diferentes domínios.

Nesta pesquisa é proposta a combinação entre um domínio-base e diversos domínios transversais. Esse tipo de combinação tenta explorar a facilidade de entrecorte das features transversais sobre o código do domínio-base. Entretanto, para que as features transversais entrecortem corretamente o produto, é necessário que o engenheiro de aplicações disponibilize informações sobre as classes que serão afetadas. Logo, é possível perceber a existência de um novo tipo de variabilidade particular de domínios transversais denominada *variabilidade de junção*.

5.2.4 Variabilidades Funcionais e Variabilidades de Junção

Um domínio transversal pode possuir dois tipos de variabilidades, **variabilidades funcionais** e **variabilidades de junção**. A primeira é inerente às regras de negócios do domínio e dita as características que variam de um produto para outro da LPS. O conceito de variabilidade de junção diz respeito às variabilidades que permitirão combinar o domínio transversal com um domínio-base. As variabilidades de junção permitem que o engenheiro de aplicações concretize os PJA's do domínio transversal de acordo com o produto que está sendo instanciado. Contudo, nem todas as variabilidades de junção estão diretamente relacionadas com a concretização de PJA's, algumas variabilidades de junção podem envolver a cópia de dependências, por exemplo.

Pode-se citar como exemplo, as variabilidades do framework transversal para persistência de dados apresentado por Camargo (2006), que pode ser reusado em diversos sistemas que possuam o interesse de persistência. No domínio desse framework existem variabilidades funcionais, tais como escolha de *drivers* de banco de dados e definição de senhas de conexão, e variabilidades de junção, tal como a definição das entidades a serem persistidas.

5.2.5 Conjuntos de Extensão

De acordo com o princípio da inconsciência (*obliviousness*), a dependência entre uma feature transversal e o código-base deveria garantir que a aplicação desconheça a existência do código transversal (Filman e Friedman, 2000). Contudo, esse princípio vem sendo questionado em diversas pesquisas (Griswold et al., 2006; Elrad et al., 2001a; Clifton e Leavens, 2003), visto que modificações no código-base podem invalidar os aspectos utilizados. A adoção desse princípio também pode causar falhas de segurança, uma vez que o código-base fica totalmente suscetível a entrecortes. Nesse sentido, soluções têm sido propostas, como é o caso das *Crosscuting Programming Interfaces* (XPI) (Griswold et al., 2006). Uma XPI faz com que o código-base não seja acessível a menos que os pontos de entrecorte estejam especificados na interface XPI.

No contexto deste trabalho, considera-se que o código-base pode ter consciência dos aspectos que atuam sobre ele, pois existem situações em que se torna necessário realizar pequenos ajustes em um domínio-base para suportar a combinação com determinados domínios transversais. Por exemplo, no domínio transversal de persistência, o código-base deve possuir determinadas chamadas de métodos para que os aspectos possam definir os pontos da aplicação em que serão realizadas as operações de remoção e busca (métodos `delete()` e `find()`). Essa condição cria uma dependência nos produtos do domínio-base quando o domínio transversal é utilizado.

O conceito de *conjunto de extensão* (CE) representa um grupo de adaptações na implementação de um domínio-base de acordo com os domínios transversais aplicados no momento da instanciação. No exemplo anterior, as classes de um domínio-base poderiam declarar um conjunto de extensão para a inserção dos métodos `save()` e `find()` no locais necessários sempre que o domínio transversal de persistência for utilizado.

5.3 Processo de desenvolvimento de LPS para Domínios-Base e Domínios Transversais

Nesta seção são apresentados processos de engenharia de domínio e engenharia de aplicações adaptados para a modelagem de linhas de produtos para domínios-base e domínios transversais. Esses processos tem o foco voltado para a instanciação automatizada de produtos utilizando geradores de aplicação e especificações LMA. Os processos propostos nesta pesquisa são extensões dos processos de engenharia de domínio e de engenharia de aplicações propostos no trabalho de

Shimabukuro (2006), apresentados nas Seções 4.4 e 4.5, respectivamente. As novas atividades propostas neste trabalho foram modeladas a partir das necessidades identificadas durante a condução desta pesquisa, porém, tais atividades não sofreram nenhum tipo de validação formal, logo, não é possível afirmar que elas são aplicáveis para todos os casos.

5.3.1 Engenharia de Domínio

A realização da etapa de engenharia de domínio tem o objetivo de criar todos os artefatos necessários da linha de produtos. Conforme apresentado em Shimabukuro (2006), nessa etapa o engenheiro de domínio deve realizar as seguintes atividades: analisar o domínio, definir a LMA do domínio, projetar e implementar os artefatos reutilizáveis, criar os gabaritos e mapear os gabaritos que serão processados para cada elemento da LMA. Na Figura 5.2 é mostrado o diagrama de atividades dessa abordagem.

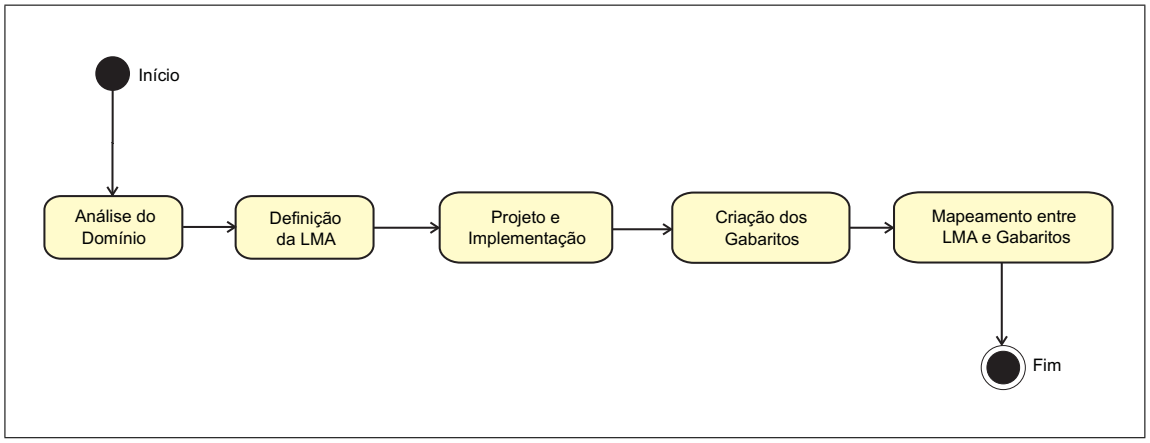

Figura 5.2: Diagrama de atividades da engenharia de domínio

Contudo, esse processo foi modificado com o objetivo de atender os novos conceitos propostos nesta pesquisa, ou seja, foram adicionadas novas atividades para apoiar a modelagem de domínios-base e domínios transversais. A seguir são mostradas em detalhes as atividades específicas para cada tipo de domínio.

Domínio-Base

Para um domínio-base, o processo é bastante parecido com o apresentado na Figura 5.2, com a diferença que após a atividade de mapeamento entre LMA e gabaritos, o engenheiro deve definir dentre os domínios transversais existentes, quais são compatíveis com o domínio-base em questão.

Entende-se como compatibilidade entre domínios a possibilidade de união e/ou complementação de features desses domínios sem a ocorrência de conflitos, resultando em novas features funcionais. Um domínio implementado com a linguagem Java, por exemplo, não pode ser combinado com outro implementado em Smalltalk. A compatibilidade entre domínios vai além de questões

de infra-estrutura, uma vez que a integração de determinadas entidades pode introduzir erros semânticos não previstos. Antes da escolha dos domínios compatíveis, o engenheiro deve realizar uma análise detalhada do comportamento das features de ambos os domínios quando utilizadas em conjunto.

Ao término dessa análise, o engenheiro estará apto para informar os domínios transversais compatíveis com o novo domínio-base, e caso seja necessário, adicionar conjuntos de extensão (CE) e pontos de junção pré-definidos (PJP) na implementação do domínio-base. É possível alterar o domínio-base para permitir e/ou facilitar a combinação com um determinado domínio transversal utilizando o conceito de conjuntos de extensão apresentado na Seção 5.2.5. Nesse caso, deve-se alterar os gabaritos do domínio-base inserindo os CE's necessários.

Finalmente, deve-se realizar a definição dos PJP's do domínio-base. Essa é uma tarefa opcional e só deve ser executada caso algum PJA do domínio transversal possua um valor padrão ou que seja invariável para todos os produtos do domínio-base que está sendo implementado. Na Figura 5.3 são mostradas as atividades realizadas durante a engenharia de domínios-base, com destaque para a nova atividade de definição de domínios compatíveis, em que é declarado um grupo de zero ou mais domínios transversais compatíveis com o novo domínio-base.

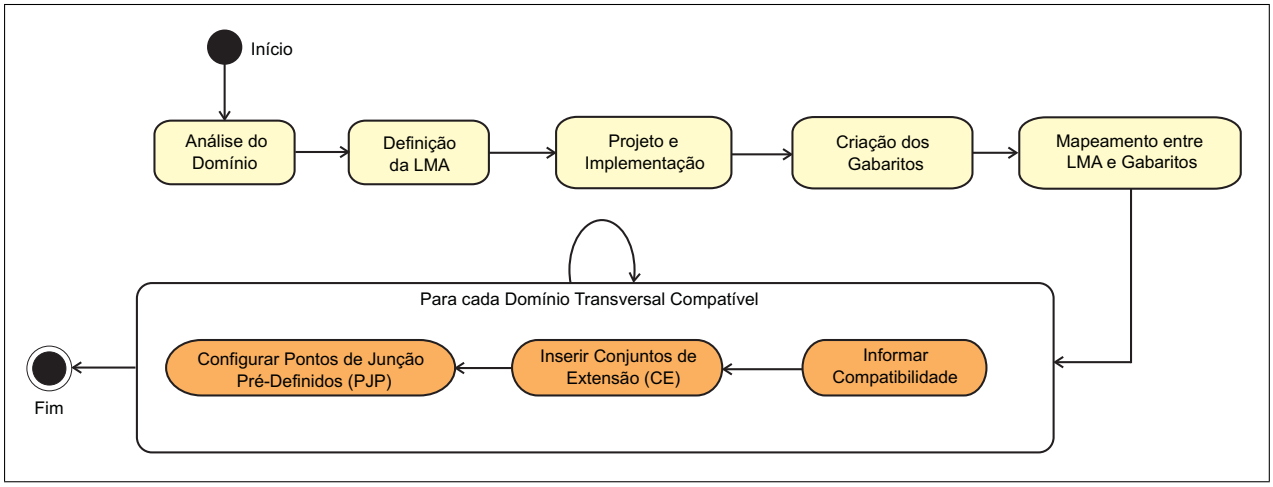

Figura 5.3: Diagrama de atividades da engenharia de domínios-base

Para auxiliar na compreensão do processo proposto, é apresentado um cenário hipotético na Figura 5.4. Esse cenário é formado por um conjunto de domínios (base e transversais) e seus relacionamentos de compatibilidade. O domínio-base A é compatível com os domínios transversais T1 e T2. Esse domínio também possui um conjunto de extensão (`CE-T1`) utilizado pelo domínio T1 e um ponto de junção pré-definido (`PJP-PJA1`) especificado para o ponto de junção abstrato `PJA1`. O domínio-base B é compatível com os domínios transversais T2 e T3 e possui um conjunto de extensão (`CE-T2`) utilizado na combinação com o domínio T2. O domínio B possui dois PJP's: `PJP-PJA4` e `PJP-PJA5`.

Na Figura 5.5 é ilustrada a inclusão de um novo domínio-base. Ao incluir um novo domínio base C, deve-se informar sua compatibilidade com os domínios transversais existentes (no caso do

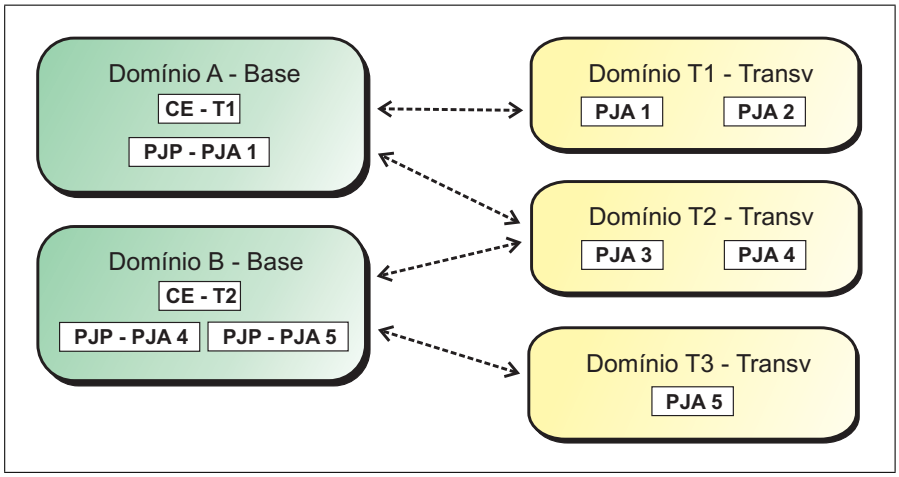

Figura 5.4: Exemplo de Compatibilidade entre domínios-base e domínios transversais

exemplo, C só é compatível com T2 e T3); se for o caso, inserir em C conjuntos de extensão em relação a cada um dos domínios transversais compatíveis (no exemplo foi inserido um CE para ser usado quando T2 for escolhido); e se for o caso, configurar no domínio-base os pontos de entrecorte pré-definidos em relação aos domínios transversais compatíveis (no exemplo foi inserido um PJP em relação ao ponto de junção abstrato `PJA5`).

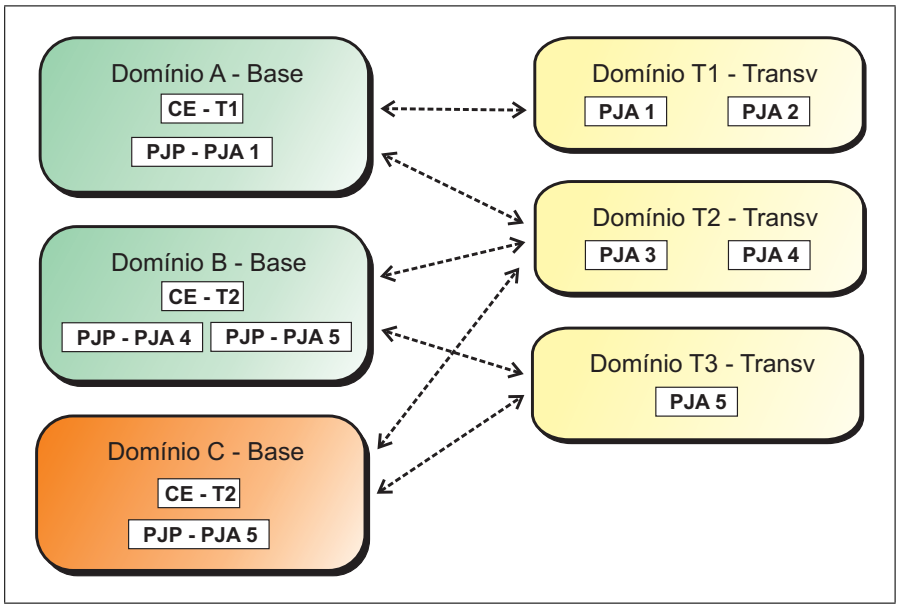

Figura 5.5: Inclusão do domínio-base C

Domínio Transversal

O processo de engenharia de domínios transversais possui passos em comum com o processo de engenharia de domínios-base e mais dois passos adicionais: definição das variabilidades de junção e escolha dos domínios-base compatíveis. A seguir são descritos em detalhes cada um desses novos passos.

Conforme visto na Seção 5.2.4, um domínio transversal pode possuir variabilidades funcionais e variabilidades de junção. O atividade de definição da LMA apresentada anteriormente já previa a modelagem das variabilidades funcionais de um domínio qualquer. Contudo, no caso de domínios transversais, também é necessário identificar as suas variabilidades de junção. Essa atividade se inicia com a modelagem dos PJA's do domínio.

Um domínio transversal pode possuir um ou mais PJA's para que a implementação das features transversais seja o mais genérica possível, permitindo o reúso dessas features em um número maior de domínios-base. Como o valor de um PJA deve variar de acordo com o produto instanciado, é necessário criar um gabarito para cada artefato que contenha um PJA. No gabarito é declarado o PJA propriamente dito e também um identificador para o PJA. Os identificadores de PJA's possibilitam criar um mapeamento entre a LMA do domínio transversal e os PJA's implementados. Existem diversas técnicas de implementação de PJA's no código-fonte, tal como *pointcuts* abstratos da linguagem AspectJ, porém, elas são diretamente ligadas à linguagem de programação adotada e estão fora do escopo desta pesquisa.

Após a implementação do domínio transversal, o engenheiro de domínio deve definir quais dos domínios-base existentes podem utilizar as features transversais recém criadas, isto é, quais domínios são compatíveis e podem ser combinados com o novo domínio transversal. Essa atividade é semelhante à realizada na engenharia de domínio-base mostrada no tópico anterior, com a diferença que agora a análise é feita a partir de um domínio transversal e que existe a possibilidade de incluir CE's e PJP's nos domínios-base compatíveis. Na Figura 5.6 são mostradas as atividades realizadas durante a engenharia de domínios, com destaque para as atividades realizadas especificamente para domínios transversais.

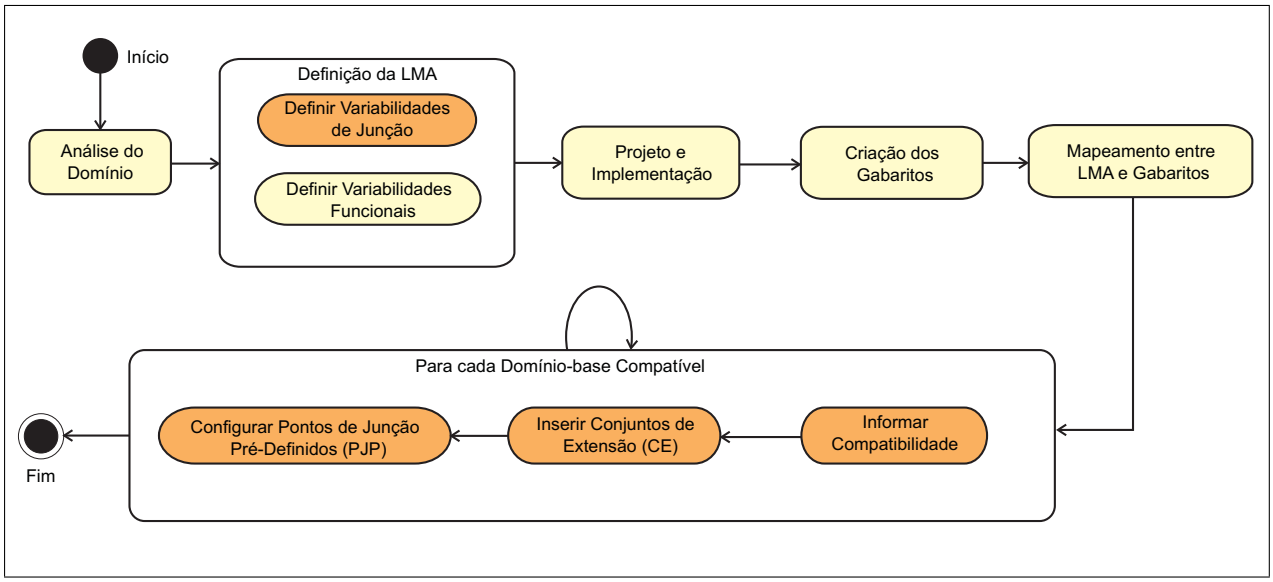

Figura 5.6: Diagrama de atividades da engenharia de domínios transversais

Na Figura 5.7 é ilustrada a inclusão do um novo domínio transversal utilizando como referência os domínios apresentados na Figura 5.4. Ao incluir um novo domínio transversal T4, deve-

se definir suas variabilidades de junção (no caso `PJA6`); informar sua compatibilidade com os domínios-base existentes (no caso do exemplo, T4 só é compatível com B); se for o caso, inserir em B conjuntos de extensão em relação ao novo domínio transversal compatível (no exemplo foi inserido `CE-T4`) e configurar nesse domínio-base os pontos de junção pré-definidos (no exemplo foi inserido um PJP em relação ao ponto de junção abstrato `PJA6`).

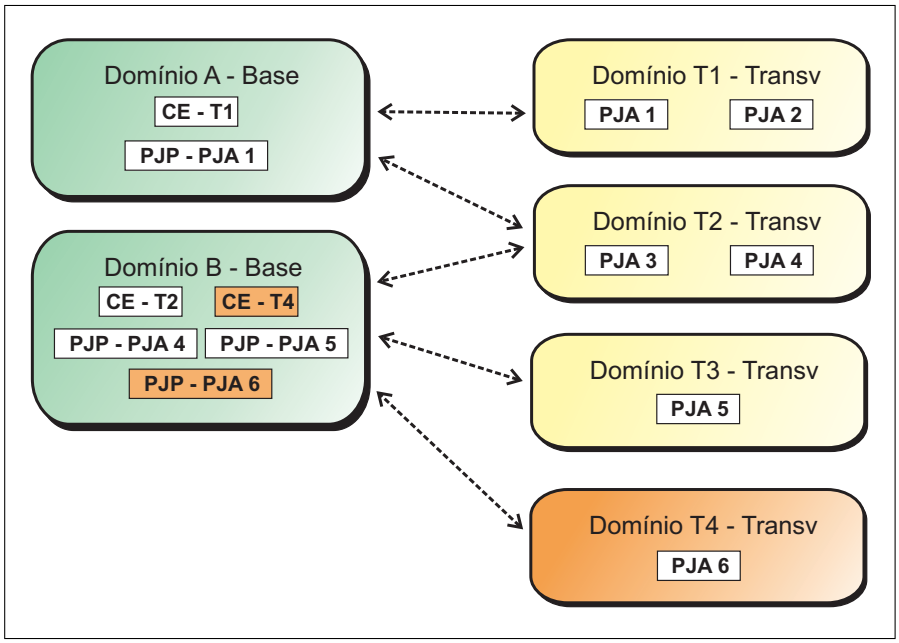

Figura 5.7: Inclusão do domínio transversal T4

5.3.2 Engenharia de Aplicações

Na fase de engenharia de aplicações deve-se criar uma aplicação com base nas necessidades dos clientes. O processo de engenharia descrito nesta Seção leva em conta a existência de uma linha de produtos implementada para o domínio desejado e a possibilidade de utilização de um gerador de aplicações para a geração do artefatos de software. Conforme apresentado em Shimabukuro (2006), nessa etapa o engenheiro da aplicação deve realizar as seguintes atividades: analisar os requisitos da aplicação, criar uma instância da LMA do domínio, gerar os artefatos, implementar manualmente funcionalidades não cobertas pela LPS e realizar testes e manutenção da aplicação gerada. Na Figura 5.8 é mostrado o diagrama de atividades dessa abordagem.

Entretanto, as atividades apresentadas na Figura 5.8 não suportam o conceito de combinação de domínios. Para tal, é necessário incluir nesse processo novas atividades que explorem a utilização de domínios-base e domínios transversais na geração de novos produtos. Na Figura 5.9 é apresentado o processo de engenharia de aplicações estendido com essas novas atividades.

Durante a análise de requisitos, o engenheiro de aplicações deve coletar as necessidades dos clientes e criar o documento de requisitos do sistema. Após a criação desse documento, o engenheiro deve avaliar a possibilidade de utilizar os domínios existentes e um gerador de aplicações. Em caso

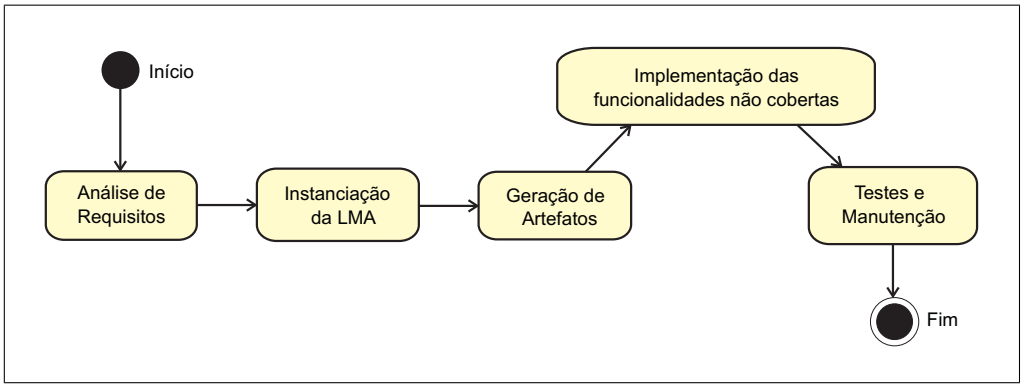

Figura 5.8: Diagrama de atividades da engenharia de aplicações

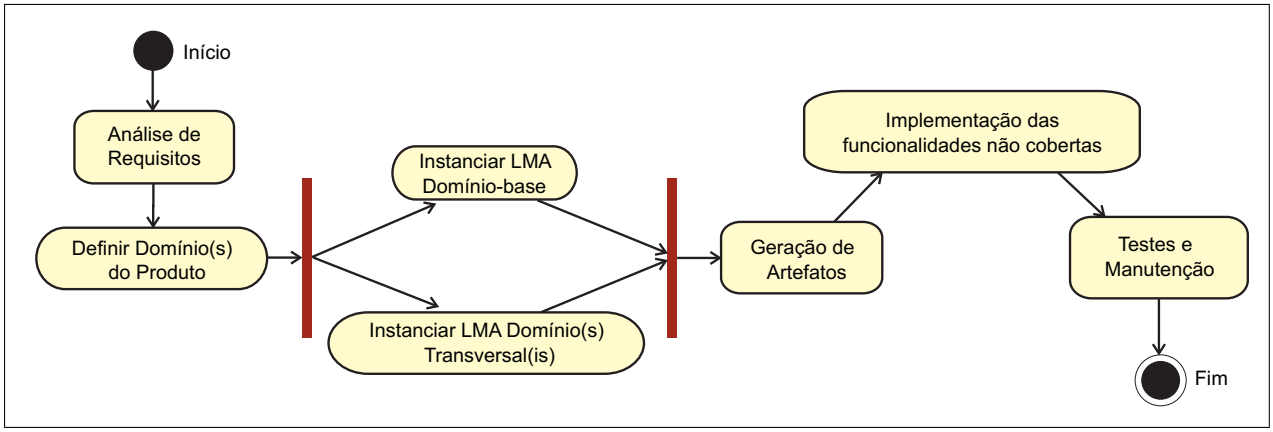

Figura 5.9: Diagrama de atividades da engenharia de aplicações com combinação de domínios

afirmativo, ele deve verificar qual domínio-base e/ou quais domínios transversais atendem aos requisitos do produto. A escolha dos domínios transversais é sempre feita utilizando como referência o domínio-base do produto, pois um domínio-base é "entrecortado" pelos domínios-transversais e não o contrário.

Após a definição dos domínios, o engenheiro de aplicações deve instanciar as LMA's de todos os domínios com base nos requisitos do produto, ou seja, o engenheiro irá preencher em paralelo as variabilidades de cada domínio. A instanciação das LMA's é feita ao mesmo tempo, e não em sequência, pois alguns elementos de uma LMA podem depender dos valores preenchidos em outras LMA's. Em seguida, é realizada a atividade de geração dos artefatos. Essa atividade é executada com a ajuda de um gerador de aplicações que deve validar cada uma das instâncias fornecidas e gerar os artefatos necessários. O processo de geração deve produzir diversos arquivos que representam a implementação das features selecionadas de todos os domínios envolvidos no processo.

Depois de gerar todos os artefatos necessários, o engenheiro de aplicações pode implementar manualmente modificações no sentido de incluir no produto final funcionalidades não cobertas. Em seguida, devem ser realizados testes e manutenção sobre o produto gerado. O processo de testes pode ser realizado somente sobre a integração entre os módulos de diferentes domínios do produto, visto que as funções individuais de cada domínio foram testadas isoladamente durante

a engenharia de domínio. Neste trabalho não é adotada nenhuma técnica ou método de testes específicos, deixando para o engenheiro de aplicações a liberdade de escolher o método mais apropriado. A tarefa de manutenção é feita após a entrega do produto, para a correção de erros, adição de novas funcionalidades, etc.

5.4 Considerações Finais

Neste Capítulo foi apresentado um processo de desenvolvimento de linhas de produtos baseado na combinação de domínios com a utilização de geradores de aplicação. Algumas pesquisas propõem processos de modularização de interesses transversais existentes em linhas de produtos (Anastasopoulos e Muthig, 2004; Lohmann et al., 2006; Lee et al., 2006; Pacios, 2006), porém o diferencial deste trabalho está no fato que a separação entre os conceitos de domínios-base e domínios transversais auxilia na generalização de interesses transversais ao ponto de possibilitar que esses interesses sejam reutilizados, não mais por uma família de produtos apenas, mas sim em várias linhas de produtos de domínios diferentes.

Foram adicionados novos passos nos processos de engenharia de domínio e de aplicações que ajudam na identificação e modelagem de características específicas de domínios-base e, principalmente, de domínios transversais, tais como variabilidades de junção e conjuntos de extensão.

Entretanto, não é possível combinar qualquer domínio transversal com qualquer qualquer domínio-base. Antes é necessário averiguar a compatibilidade entre os domínios que se deseja utilizar. O processo de verificação de compatibilidade entre um domínio-base e um domínio transversal não é trivial, isto é, deve-se realizar uma análise completa sobre os impactos das features transversais no código-base.

Além dos testes isolados de cada uma das linhas de produtos envolvidas, também é necessário testar o funcionamento conjunto de todas elas. O nível de complexidade desses testes é diretamente proporcional ao número de domínios transversais compatíveis aplicados, visto que a adição de um domínio compatível implica no re-teste do novo arranjo. Em contrapartida, após essa análise, será possível gerar artefatos e integrar automaticamente as features transversais reusadas pelo engenheiro de aplicações no produto final, reduzindo tempo e custos de desenvolvimento.

CAPÍTULO

6

Captor-AO

6.1 Considerações Iniciais

Neste capítulo é apresentada a ferramenta Captor-AO, que é uma extensão do gerador Captor capaz de realizar a combinação entre vários domínios distintos por meio da programação orientada a aspectos, tornando possível a geração de aplicações formadas por features de um domínio específico (domínio-base) e features de diferentes domínios transversais, desde que a união dessas features não produza conflitos no produto resultante.

O Capítulo está organizado da seguinte forma. Na Seção 6.2 são apresentados em detalhes o objetivo da ferramenta e a abordagem de geração proposta. A Seção 6.3 mostra quais são e como foram obtidos os requisitos do gerador Captor-AO, ou seja, é descrito como foi conduzido o processo de análise de requisitos da ferramenta. A Seção 6.4 trata sobre o projeto e a arquitetura do gerador Captor-AO. Na Seção 6.5 são apresentadas as principais soluções encontradas para os desafios dessa abordagem de geração e a forma como cada uma delas foi implementada. Por fim, na Seção 6.6 são apresentadas as considerações finais do capítulo.

6.2 Captor-AO: Extensão do gerador de aplicações Captor

Conforme visto no Capítulo 4, a ferramenta Captor é um gerador de aplicações baseado em composição, que pode ser configurado para diferentes domínios (Shimabukuro et al., 2006). Para

cada domínio, deve-se fornecer ao Captor a especificação de sua LMA, os artefatos a serem utilizados durante a geração (tanto artefatos fixos quanto os gabaritos com as partes variáveis) e o mapeamento entre a LMA e os gabaritos. Desde sua criação, o Captor já foi configurado para diversos domínios diferentes. No entanto, cada um dos domínios é utilizado isoladamente na criação de um produto, ou seja, ao iniciar um novo projeto o usuário do Captor deve escolher apenas um entre os domínios previamente configurados. Com o advento da separação avançada de interesses (Kiczales et al., 1997) surgiu a idéia de modularizar interesses transversais na forma de domínios transversais.

Assim, foi desenvolvida neste trabalho uma extensão do gerador Captor, denominada **Captor-AO**, que permite a integração entre features de domínios-base (domínios de aplicação) e domínios transversais, estando em conformidade com o processo proposto no Capítulo 5. O gerador Captor-AO torna possível a adição de features de um ou mais domínios transversais durante a instanciação de produtos. Dessa forma, o produto resultante pode possuir features provenientes de um domínio-base e de diversos domínios transversais. A Figura 6.1 ilustra o processo de utilização do gerador Captor-AO, em que a ferramenta é utilizada para instanciar um produto do domínio base `D1` estendido por features do domínio transversal `D2`.

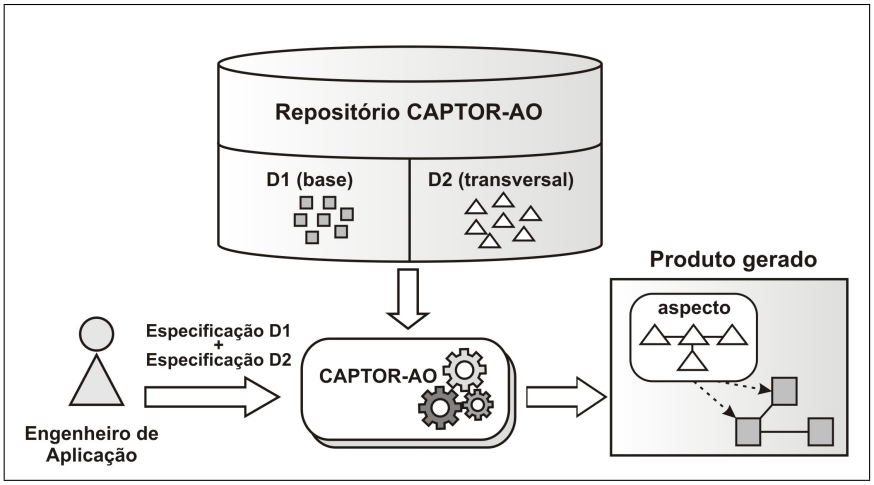

Figura 6.1: Processo de geração de aplicações com utilização de múltiplos domínios

O foco dessa extensão está na aplicação prática da combinação entre domínios-base e domínios transversais, com a finalidade de tirar proveito da facilidade de inclusão e evolução de features proporcionada pela programação orientada a aspectos. O Captor-AO possibilita a exploração das propriedades ortogonais de um domínio transversal e fornece formas de mapeamento dos PJA's de um domínio transversal. O gerador também verifica se o engenheiro de aplicações está respeitando a regras de combinação entre domínios para evitar que a composição entre features de diferentes domínios produza erros no código gerado.

O desenvolvimento do Captor-AO foi uma atividade que levou oito meses para ser completada e foi realizada de forma iterativa e incremental. A medida em que os requisitos eram coletados, o projeto e a implementação da ferramenta foram sendo atualizados. No sentido de tornar a apre-

sentação desse processo mais clara, os resultados foram agrupados nas seguintes fases do ciclo de desenvolvimento: engenharia reversa da aplicação Captor, análise de requisitos, projeto e implementação do sistema.

Pode-se citar a etapa de engenharia reversa como uma das mais difíceis de toda a pesquisa. Considerando a complexidade do gerador e o tamanho do software existente, foram necessários 3 meses de trabalho para analisar o sistema original que possuía em torno de 30.000 linhas de código segundo o sistema Metrics [1]. Também foram realizadas modificações em diversas partes da ferramenta de acordo com os pontos de melhoria identificados durante a engenharia reversa ou que foram sugeridos pelos usuários do gerador, principalmente colegas e professores do ICMC/USP. As alterações realizadas foram desde melhorias na interface e correções de defeitos pontuais (*bugfix*) até ajustes de portabilidade para outros sistemas operacionais.

6.3 Análise de Requisitos

Os requisitos necessários para o desenvolvimento do gerador Captor-AO foram obtidos de acordo com um levantamento das funcionalidades desejáveis para permitir a combinação de vários domínios. Esse levantamento foi realizado durante seminários com a orientadora deste trabalho e membros do laboratório de engenharia de software do ICMC, além de alunos que utilizaram o Captor durante exercícios de disciplinas de pós-graduação. O conjunto de funcionalidades foi aprimorado por meio de estudos de caso realizados durante a condução desta pesquisa de mestrado. A Tabela 6.1 apresenta um resumo dos requisitos levantados para a extensão do Captor em direção ao Captor-AO.

Na Figura 6.2 é mostrado o diagrama de casos de uso em que estão representados os casos de uso escritos no contexto desta pesquisa (os demais casos de uso implementados na primeira versão da ferramenta foram omitidos por questão de legibilidade).

Após esse levantamento inicial, foram realizados estudos de caso para averiguar se os casos de uso identificados seriam suficientes para proporcionar a geração de aplicações com base em vários domínios. O primeiro estudo conduzido foi baseado na linha de produtos GestorPsi (Ver Seção 4.6) e investigou a possibilidade de isolamento dos pontos de junção contidos nos aspectos que implementam as features opcionais dessa linha de produtos. Constatou-se que os pontos junção definidos dentro da LPS poderiam ser substituídos por pontos de junção abstratos, denominados PJA's de um domínio transversal (Ver Seção 5.2.1).

Em seguida, foi realizado um estudo para a geração de código para uma de linha de produtos de bóias náuticas FWS (do inglês *Floating Weather Station*) (Weiss e Lai, 1999), que já havia sido configurada pelo Captor, mas não possuía uma camada de persistência de dados. O domínio de bóias náuticas é apresentado em detalhes na Seção 7.3. Para a inclusão das funções de persistência foi escolhido o framework de persistência desenvolvido por Camargo (2006), que além de ser um

[1] Metrics Plugin: `http://metrics.sourceforge.net/`

Ponto de vista do Engenheiro de Domínio	
R1	O Captor-AO deve permitir ao engenheiro de domínio especificar um novo domínio-base e informar, dentre os domínios transversais existentes no repositório, quais são compatíveis com esse domínio-base.
R2	O Captor-AO deve permitir ao engenheiro de domínio especificar um novo domínio transversal e informar, dentre os domínios-base existentes no repositório, quais são compatíveis com esse novo domínio transversal, ou seja, que podem ser combinados com ele.
R3	O Captor-AO deve permitir ao engenheiro de domínio diferenciar as variabilidades de um domínio transversal que são inerentes aos pontos de junção (variabilidades de junção), em relação às variabilidades inerentes às regras de negócios do domínio (variabilidades funcionais).
R4	O Captor-AO deve permitir ao engenheiro de domínio especificar valores pré-definidos de pontos de junção em que um domínio transversal entrecorta um determinado domínio base, podendo usar esses valores como *default* no caso do engenheiro de aplicações não modificá-los.
R5	A especificação dos domínios-base e transversais dever ser feita da mesma forma que no Captor, ou seja, por meio de uma LMA baseada em formulários, gabaritos XSL e arquivos de mapeamento em XML.
R6	O Captor-AO deve permitir ao engenheiro de domínio incluir nos gabaritos dos domínios-base trechos que só farão parte do produto final caso certo domínio transversal faça parte da composição.
Ponto de vista do Engenheiro de Aplicações	
R7	O Captor-AO deve permitir ao engenheiro de aplicações criar projetos em que sejam combinados um domínio-base e zero ou mais domínios transversais.
R8	O Captor-AO deve permitir ao engenheiro de aplicações configurar separadamente os pontos em que os domínios transversais irão entrecortar a aplicação base.
R9	O Captor-AO deverá gerar os artefatos resultantes em um único local, de acordo com o nome do projeto fornecido pelo engenheiro de aplicações. Os artefatos ficarão disponíveis para pós-processamento pelo próprio Captor-AO ou para serem integrados manualmente pelo engenheiro de aplicações.

Tabela 6.1: Requisitos do gerador estendido Captor-AO

trabalho abrangente sobre esse tema, também foi um projeto do próprio grupo de pesquisas, logo, com facilidade de acesso ao código-fonte.

O processo de reúso do framework de persistência é baseado na instanciação de aspectos abstratos que realizam a tarefa de persistência em banco dos dados de uma ou mais classes classes base. No caso da LPS de bóias náuticas, o engenheiro de aplicações poderia instanciar manualmente os aspectos do framework, utilizando como base um sistema FWS específico gerado pelo Captor.

O objetivo do estudo era realizar a configuração do framework de persistência na forma de um domínio transversal e realizar a combinação com o domínio de bóias náuticas. Percebeu-se a possibilidade de geração automática das classes do sistema de bóias náuticas e do framework de persistência em iterações separadas, seguida da composição manual dos aspectos com o código-

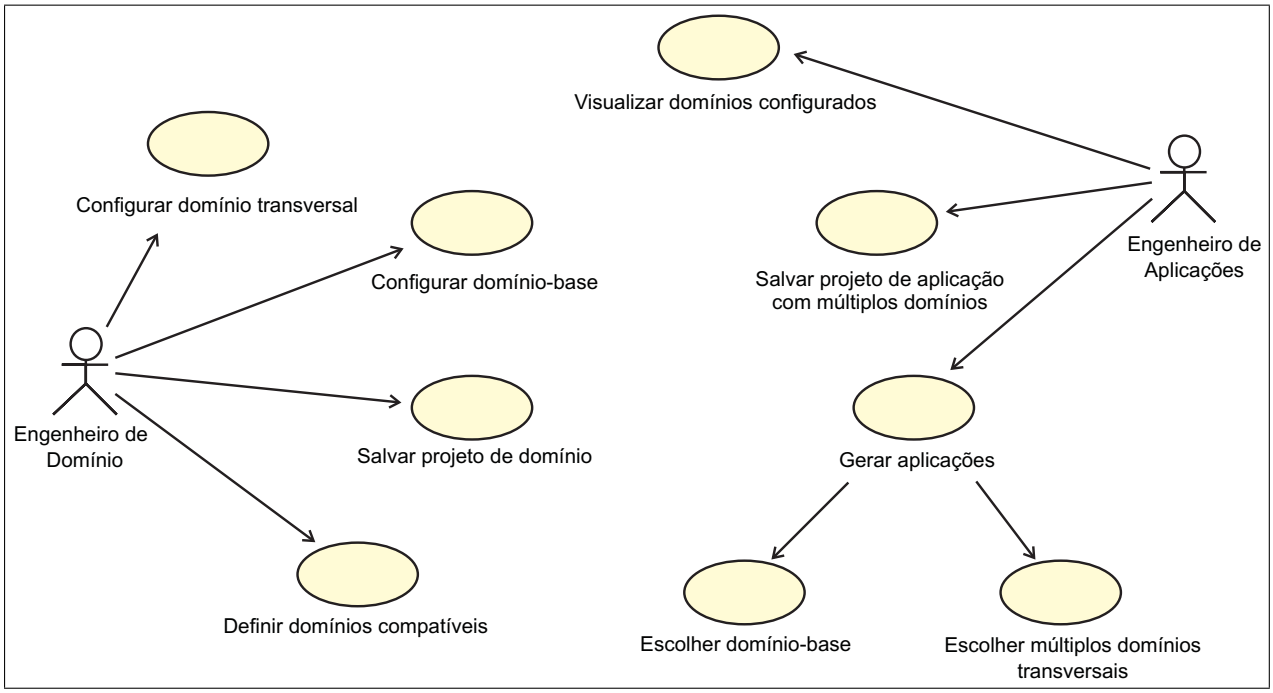

Figura 6.2: Diagrama de casos de uso da ferramenta Captor-AO

base, ou seja, usando a versão original do Captor é possível instanciar produtos do domínio bóias náuticas que possuam uma camada de persistência de dados, embora essa não seja a maneira otimizada. Nessa forma de instanciação não se separam as variabilidades de junção das variabilidades funcionais e também não ficam explícitos o domínio-base e o domínio transversal da composição. Esse processo de geração individual de cada produto e composição de features transversais em iterações separadas é exemplificado na Figura 6.3. No Capítulo 7 serão descritas em detalhes todas as atividades realizadas durante a condução desse estudo de caso, desde a configuração do domínio transversal de persistência, passando pela implementação dos gabaritos até a etapa de geração de produtos utilizando a ferramenta Captor-AO.

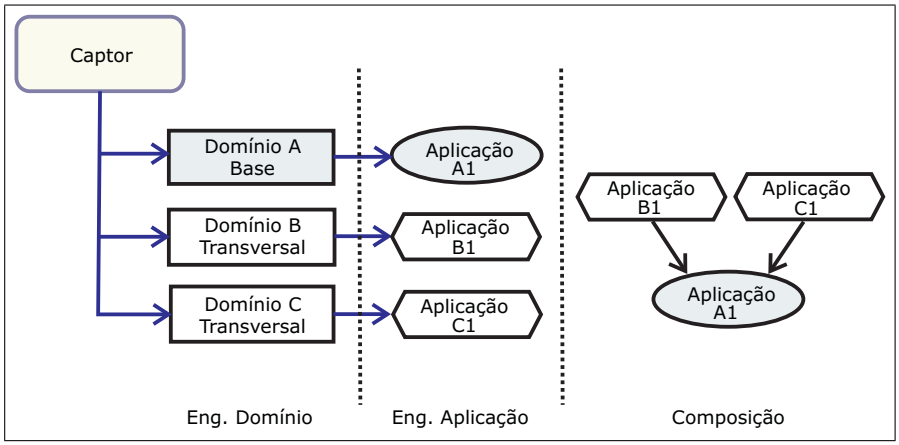

Figura 6.3: Geração de aplicações individuais e composição manual de pontos de junção (Captor versão original)

Após a condução desse estudo de caso, constatou-se a necessidade de evoluir o processo de engenharia de aplicações de forma a permitir a automatização da etapa de combinação dos pontos de junção, reduzindo assim o tempo necessário para instanciação de features transversais. Na Figura 6.4 é apresentado o novo processo de instanciação, onde a ferramenta estendida Captor-AO se encarrega de instanciar as variabilidades de junção dos domínios transversais `B` e `C` utilizando os pontos definidos no domínio-base `A`.

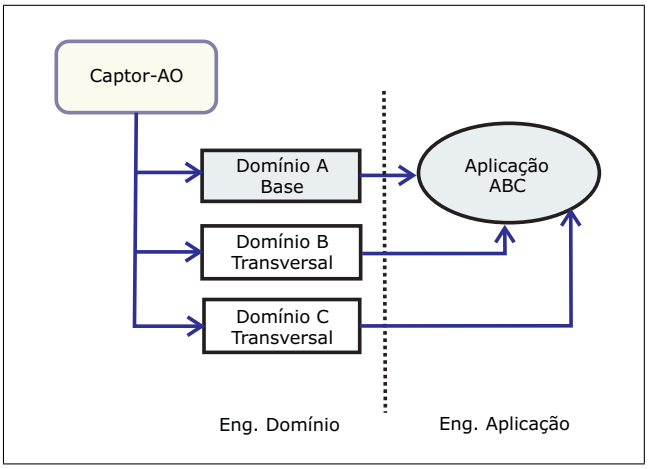

Figura 6.4: Instanciação de produtos com combinação automática de features base e transversais (Captor-AO)

O último estudo realizado envolveu o framework GREN (Braga e Masiero, 2002a). O GREN é um framework orientado a objetos que foi desenvolvido com base na linguagem de padrões GRN (Braga et al., 1999b), que permite a criação de aplicações no domínio de sistemas de gestão de recursos de negócios e foi implementado com a linguagem Smalltalk. Nesse estudo o domínio GREN, que já havia sido configurado no Captor durante a pesquisa de Shimabukuro (2006), foi adaptado como um domínio-base na ferramenta Captor-AO e posteriormente combinado com um domínio transversal de controle de acesso. Para tal, foi utilizada a implementação do sub-sistema orientado a aspectos para controle de acesso (Silva, 2004; Silva et al., 2006). Esse subsistema é uma evolução do framework GREN em que foi feita a implementação do interesse de segurança usando a abordagem de programação orientada a aspectos e a linguagem AspectS.

O estudo de caso foi utilizado como base de teste das funcionalidades até então desenvolvidas no Captor-AO. Na Figura 6.5 é apresentado o modelo conceitual do gerador estendido. As novas abstrações introduzidas são representadas por retângulos claros e as abstrações existentes na primeira versão do gerador, mas que sofreram alterações para apoiar os novos requisitos, são representadas por retângulos escuros.

6.4 Projeto

O projeto da ferramenta foi realizado seguindo o modelo original do Captor, utilizando a linguagem de programação Java e um conjunto de bibliotecas externas fornecidas pela Sun Microsys-

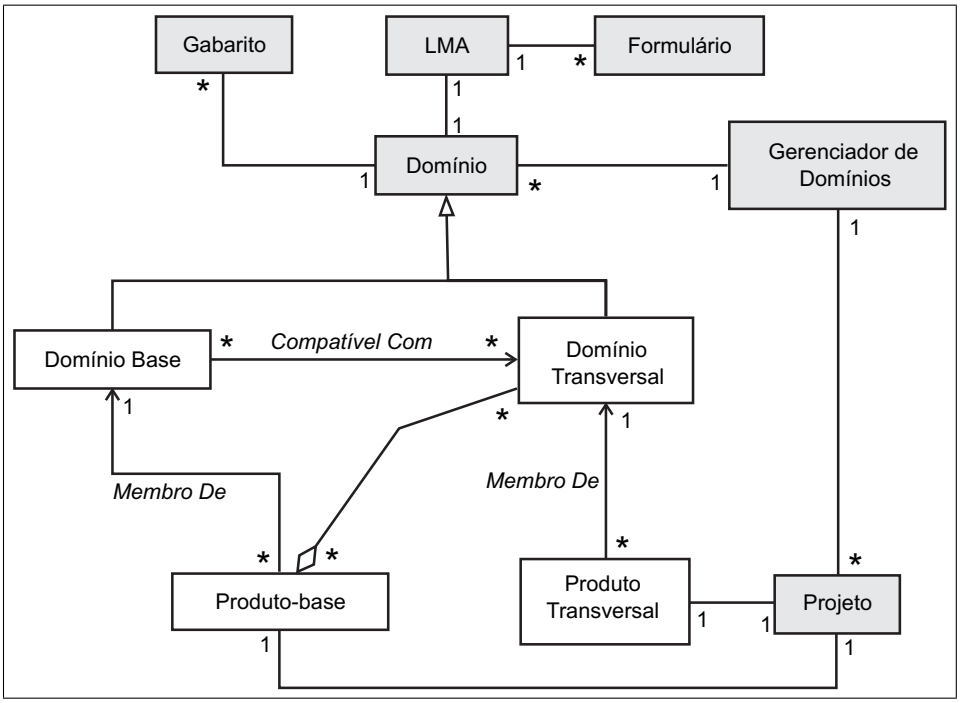

Figura 6.5: Modelo conceitual do sistema Captor-AO

tems[2]. As principais modificações em relação ao projeto inicial ocorreram em três módulos do sistema: Gerenciador de Domínios, Gerenciador de Aplicações e Compositor de Artefatos. Na Figura 6.6 é mostrada a arquitetura do Captor-AO já com as novas camadas desenvolvidas para apoiar as atividades de engenharia de domínios e de aplicações apresentadas na Seção 5.3.

O módulo Gerenciador de Domínios é responsável pela manutenção dos domínios-base e transversais configurados na ferramenta. Para a configuração de um novo domínio, o engenheiro de domínio deve implementar os artefatos fixos e os gabaritos dos elementos variáveis. A interface gráfica utilizada durante o processo de engenharia de domínio permite a criação da LMA do domínio e a coleta de algumas informações específicas de domínios-base e domínios transversais (Especificação Transversal na Figura 6.6). O módulo Gerenciador de Projetos é responsável pela manutenção dos dados dos projetos criados no Captor-AO, ou seja, criação, manutenção e remoção de projetos de novos domínios e também de novos produtos.

A interface gráfica utilizada durante a geração de novos produtos permite que o engenheiro de aplicações selecione o domínio base e os domínios transversais que serão utilizados. O módulo Gerenciador de Aplicações possibilita a instanciação simultânea de múltiplas LMA's, ou seja, em um mesmo passo, o engenheiro de aplicações pode especificar as variabilidades do domínio-base e dos domínios transversais escolhidos. Terminada a etapa de instanciação de LMA's, o engenheiro de aplicações pode salvar as informações do projeto ou iniciar o processo de geração dos artefatos. Para a geração do sistema, o módulo Compositor de Artefatos deve selecionar os gabaritos dos domínios envolvidos e processar os arquivos de saída de acordo com as variabilidades de cada domínio. O módulo Compositor de Artefatos também é responsável pela validação das instâncias

[2]Sun Microsystems: `http://www.sun.com`

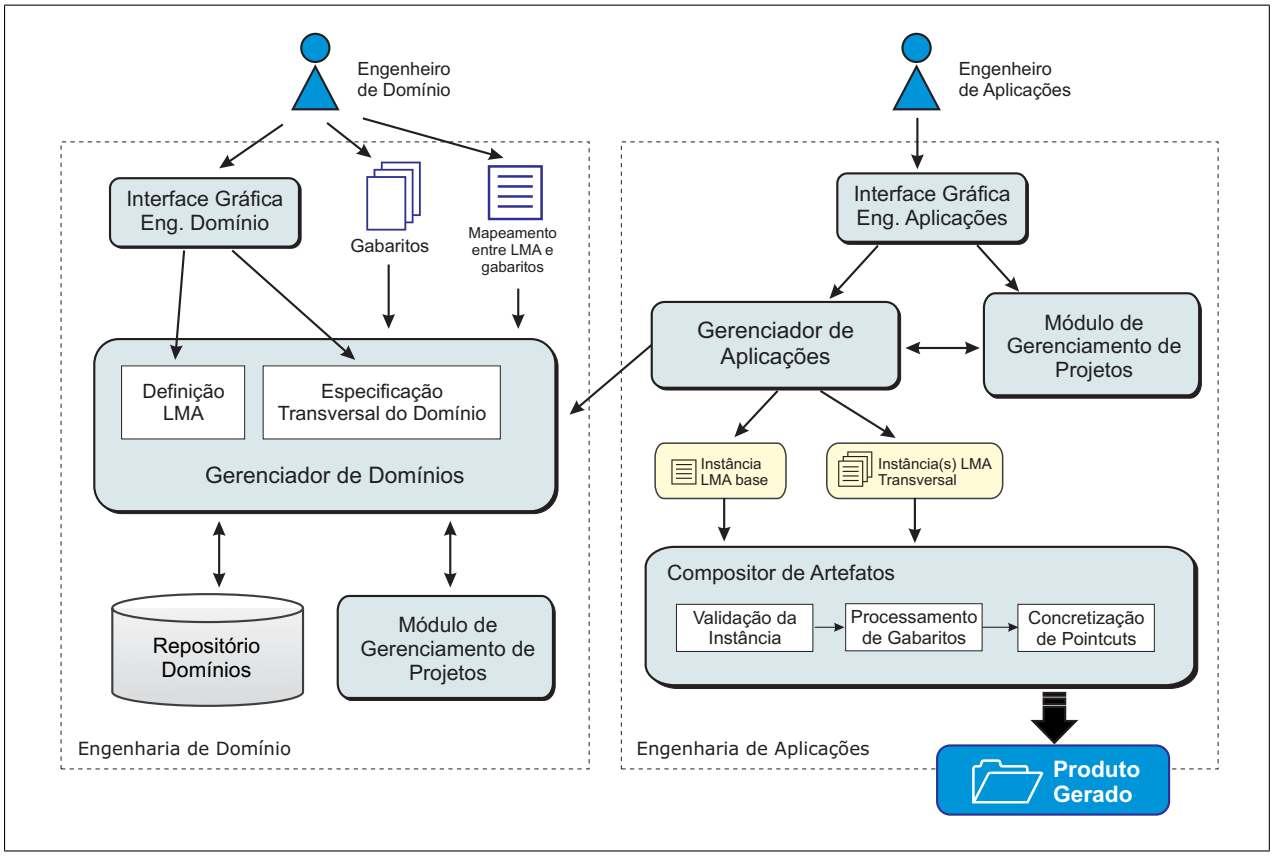

Figura 6.6: Arquitetura do Captor-AO

fornecidas e pela concretização dos pontos de entrecorte das features transversais no código-base do produto gerado.

6.5 Implementação

A etapa de implementação das novas funcionalidades foi realizada em paralelo com a condução dos estudos de caso e durou quatro meses. A versão atual do Captor-AO (versão 2.0.6), excluindo artefatos externos e casos de teste, possui 338 classes e um total de 38.033 linhas de código. Tanto o código do gerador Captor-AO quanto os binários executáveis podem ser obtidos na página *web* do projeto: `http://captor.googlecode.com`. Nessa página, além do software, também estão disponíveis artigos publicados e apresentações sobre o Captor-AO (Pereira e Braga, 2007; Pereira et al., 2008). O projeto da ferramenta é compatível com o gerenciador de projetos Maven[3], que facilita a automatização do processo de compilação e empacotamento do sistema. O diagrama de classes apresentado na Figura 6.7 mostra as principais classes de projeto implementadas no gerador Captor-AO.

[3]Apache Maven Project: `http://maven.apache.org/`

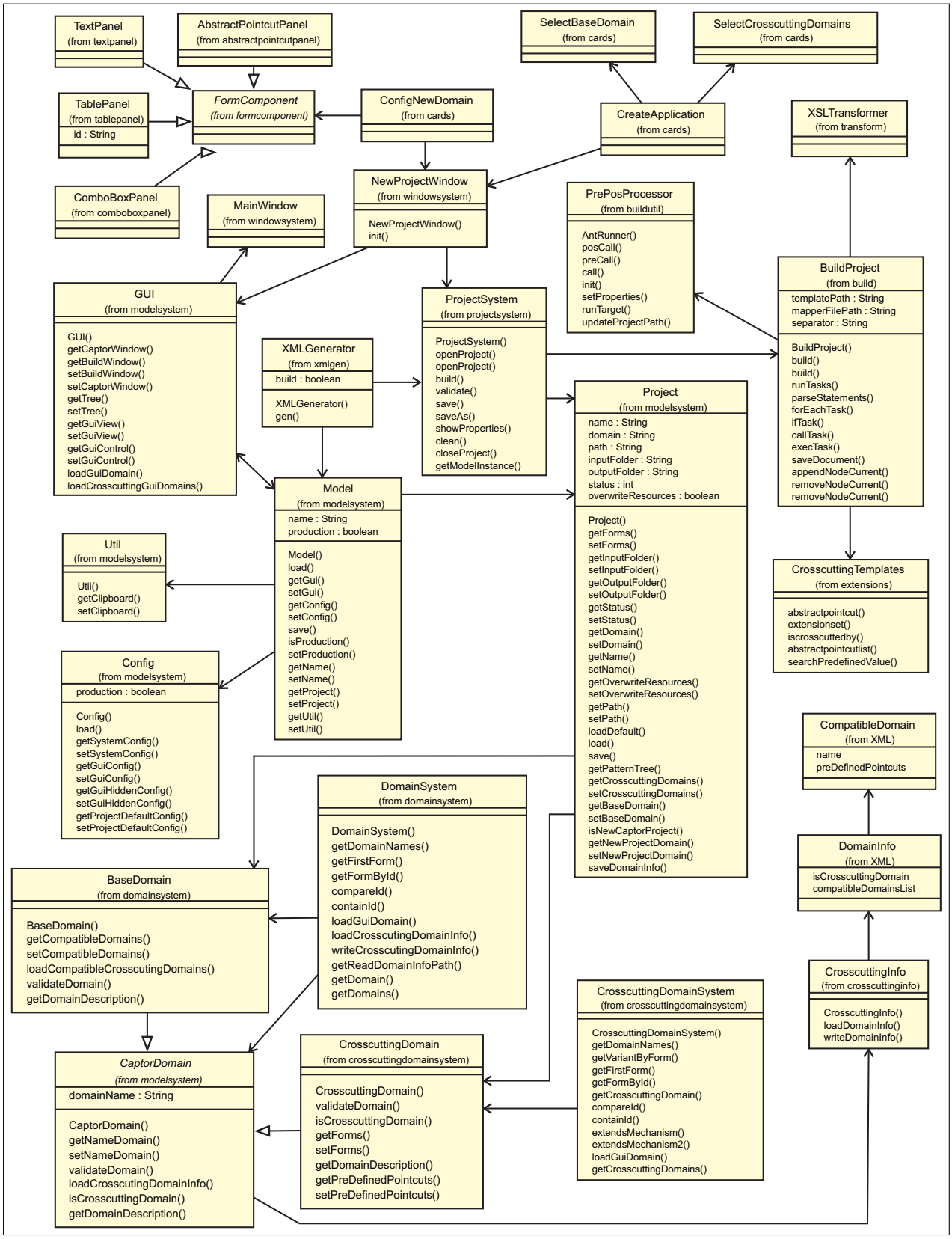

Figura 6.7: Diagrama de classes de projeto do gerador Captor-AO

Algumas das classes vistas na Figura 6.7 já existiam na versão inicial da ferramenta, como as classes `Project` e `DomainSystem`. Entretanto, tais classes sofreram diversas modifica-

ções para suportar os novos requisitos propostos. As classes de manipulação de domínios (classes `CaptorDomain`, `BaseDomain` e `CrosscuttingDomain`) foram concebidas a partir do projeto inicial do sistema e as demais entidades foram desenvolvidas a medida em que eram realizados os estudos de caso descritos na Seção 6.3. Nas próximas Seções são detalhados os mecanismos implementados no gerador Captor-AO para suportar as novas atividades de engenharia de domínios e de aplicações apresentadas nas Seções 5.3.1 e 5.3.2, respectivamente.

6.5.1 Gerenciamento de Domínios

A ferramenta Captor-AO tem a capacidade de armazenar diferentes domínios e utilizá-los isoladamente ou em conjunto com outros domínios durante o processo de geração de produtos. As classes `DomainSystem` e `CrosscuttingDomainSystem` tem a responsabilidade de acessar informações específicas dos domínios armazenados no repositório, em forma de arquivos XML. Essas classes também gerenciam instâncias dos objetos `BaseDomain` e `CrosscuttingDomain`.

As classes `BaseDomain` e `CrosscuttingDomain` são abstrações que representam domínios-base e transversais, respectivamente. Ambas as classes são especializações da classe `CaptorDomain`, que por sua vez, encapsula atributos comuns a todos os domínios da ferramenta. Na Figura 6.8 é mostrado o diagrama de classes das entidades que compõe o subsistema de gerenciamento de domínios. O relacionamento de compatibilidade entre domínios apresentado na Figura 6.8 é utilizado para indicar quais domínios transversais podem ser combinados com quais domínios-base, conforme discutido na Seção 5.3.1.

Na Figura 6.9 é mostrada a primeira interface utilizada no processo de engenharia de domínio. Pode-se observar na figura que o Captor-AO solicita ao engenheiro de domínio o tipo de domínio (base ou transversal). Caso seja escolhido o tipo base, o engenheiro tem a possibilidade de definir, dentre os domínios transversais registrados no repositório, quais domínios são compatíveis com o novo domínio-base.

6.5.2 Definição da LMA no Gerador Captor-AO

O processo de criação da LMA no gerador Captor-AO pode variar de acordo com tipo de domínio. Para a definição da LMA de um domínio-base é necessário modelar as variabilidades funcionais utilizando os elementos corretos do gerador Captor-AO de acordo com o tipo de informação que deverá ser guardada, por exemplo, caixas de texto, listas, botões, etc. Para um domínio transversal, além das variabilidades funcionais, também é necessário representar as suas variabilidades de junção (Ver Seção 5.2.4). A modelagem das variabilidades de junção pode ser feita com a ajuda de um elemento de formulário chamado `abstractpointcutpanel`. Esse elemento é utilizado para modelar e representar na interface gráfica da ferramenta os PJA's (Ver Seção 5.2.1) de um domínio transversal e é mostrado na Figura 6.10.

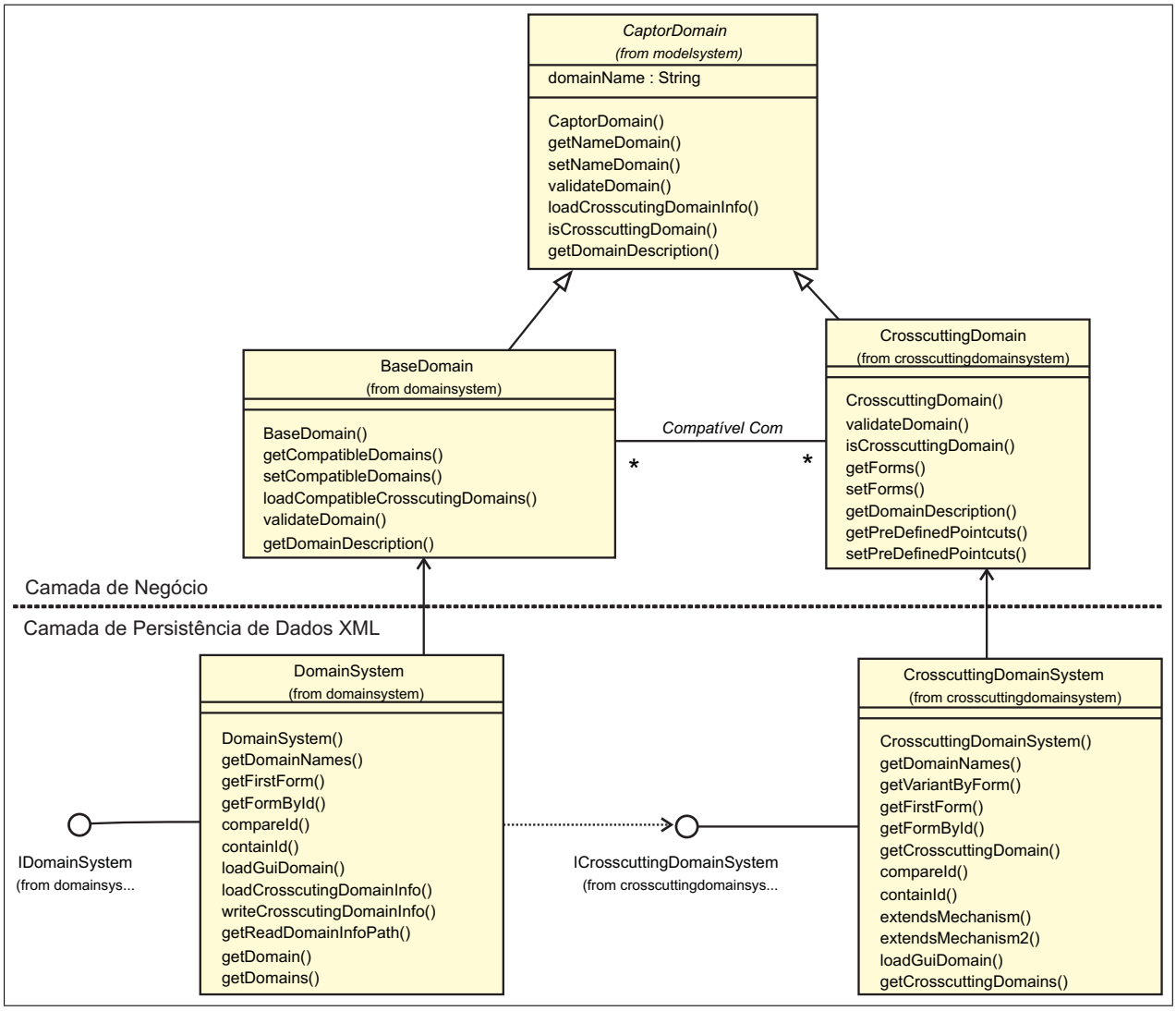

Figura 6.8: Classes de gerenciamento de domínios

O elemento `abstractpointcutpanel` é preenchido durante a engenharia de aplicações. O engenheiro deve fornecer um valor concreto para o PJA (caixa de texto **Concrete Value**) de acordo com o produto que está sendo gerado. O atributo **Refname** é um identificador que pode ser utilizado pelos gabaritos do domínio transversal para acessar o valor concreto do PJA. O atributo **Description** é empregado para fornecer informações de suporte sobre o preenchimento do campo. Se no momento da geração de um produto, um mesmo PJA tiver dois valores distintos, um PJP (Ver Seção 5.2.2) e outro fornecido pelo engenheiro de aplicações, apenas o segundo valor é utilizado.

Caso seja necessário associar não apenas um valor, mas um grupo de valores a um PJA, deve-se utilizar um outro componente, chamado `abstractpointcutlist`. A Figura 6.11 apresenta o elemento `abstractpointcutlist`, que é um elemento derivado de `abstractpointcutpanel` e possui um modo de utilização similar. O engenheiro de aplicações pode incluir quantos valores forem necessários na lista clicando no botão **Add** mostrado na Figura 6.11.

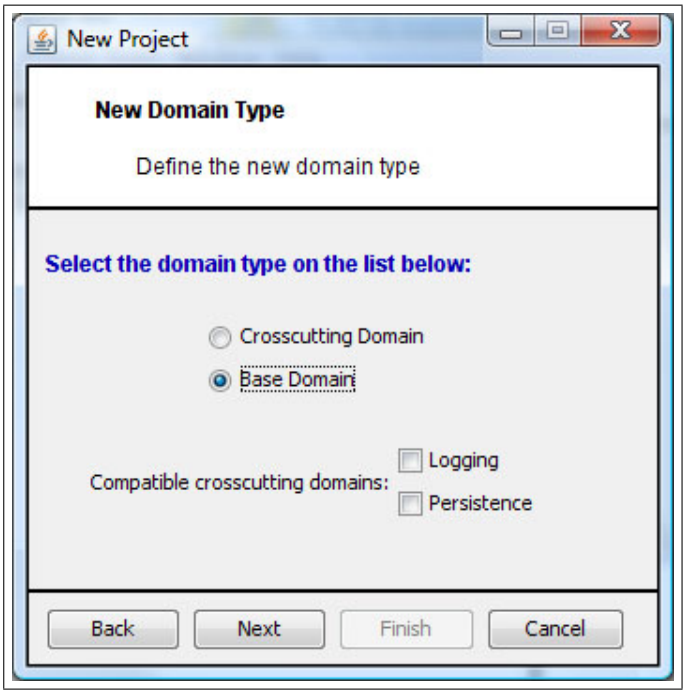

Figura 6.9: Interface do Captor-AO para configuração de um novo domínio

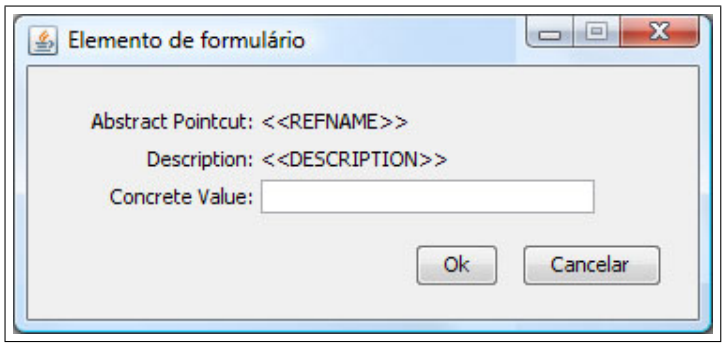

Figura 6.10: Modelagem de Pontos de Junção Abstratos

6.5.3 Extensão da Linguagem de Gabaritos

Foram implementadas novas extensões na linguagem de processamento de gabaritos utilizada pelo Captor-AO com o objetivo de facilitar a modelagem das variabilidades de junção (Ver Seção 5.2.4) e permitir a definição de conjuntos de extensão (Ver Seção 5.2.5) dentro dos gabaritos do Captor-AO.

Essas extensões foram implementadas na forma de componentes do processador de transformações Java-Xalan[4]. Essas extensões permitem que o engenheiro de domínio insira nos gabaritos comandos especiais que serão executados na fase de processamento de gabaritos. Tais comandos possibilitam a manipulação de dados dos PJA's e a criação de conjuntos de extensão. A seguir é detalhado o funcionamento de cada um desses comandos.

[4]Apache XML Project: `http://xml.apache.org/xalan-j/`

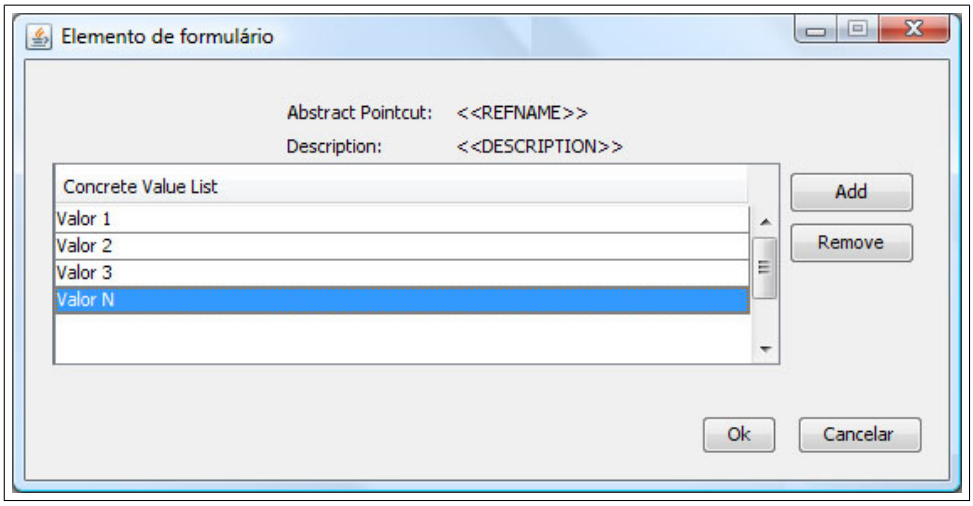

Figura 6.11: Ponto de Junção Abstrato com lista de valores

Como visto na Seção 6.5.2, o engenheiro de domínio pode representar graficamente na interface da ferramenta os PJA's de um domínio transversal. Visto que esses PJA's devem ser concretizados pelo engenheiro de aplicações durante a geração de um novo produto, nota-se que os gabaritos que utilizam tais PJA's devem acessar, ou o valor fornecido pelo engenheiro de aplicações, ou o valor pré-definido desses PJA's e realizar a substituição nos artefatos necessários. Para tal, o engenheiro de domínio pode utilizar dentro dos gabaritos o seguinte comando:

```
<captor:abstractpointcut refname="IDENTIFICADOR"/>
```

Esse comando pesquisa o valor concreto do PJA identificado pelo atributo **`refname`** e o insere no gabarito que está sendo processado. Na Figura 6.12 é apresentado um gabarito de exemplo, onde é utilizado o comando `abstractpointcut` (linhas 14 e 15).

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0"
3                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4                  xmlns:captor="captor.lib.xalan.extensions.CaptorExtension"
5                  extension-element-prefixes="captor">
6
7  <xsl:output method="text"/>
8  <xsl:template match="/">
9
10 package instantiation;
11 import persistence.connection.*;
12
13 public aspect MyConnectionComposition extends ConnectionComposition {
14         public pointcut openConnection() : <captor:abstractpointcut refname="openConnection"/>;
15         public pointcut closeConnection() : <captor:abstractpointcut refname="closeConnection"/>;
16     }
17 }
18     </xsl:template>
19 </xsl:stylesheet>
```

Figura 6.12: Exemplo de gabarito com utilização de PJA's

No exemplo é usado o aspecto `MyConnectionComposition` do framework transversal de persistência (Camargo, 2006). Esse aspecto define os pontos de entrecorte (*pointcuts*) da aplicação

base genérica onde devem ocorrer as operações de conexão (pointcut **openConnection** na linha 14 do gabarito) e desconexão (pointcut **closeConnection** na linha 15 do gabarito) com o banco de dados. Visto que os comandos implementados não fazem parte da especificação padrão da linguagem de processamento de gabaritos, é necessário incluir o pacote de extensão `CaptorExtension` manualmente para que o processador reconheça os novos elementos. A sintaxe de inclusão do pacote `CaptorExtension` está nas linhas 4 e 5 da Figura 6.12.

Também foi implementado um outro comando, denominado `extensionset`, que permite modificar o processamento de um **domínio-base** de acordo com os **domínios transversais** utilizados na geração do produto na forma de conjuntos de extensão. Como visto na Seção 5.2.5, conjuntos de extensão são adaptações nos gabaritos de um domínio-base de acordo com os domínios transversais aplicados no momento da geração. Para cada conjunto de extensão deve-se criar no gabarito um elemento do tipo `extensionset`, que é uma anotação especial em que um conjunto de comandos é processado somente se um determinado domínio transversal fizer parte da combinação. A declaração de conjuntos de extensão dentro dos gabaritos é feita utilizando o comando:

```
<captor:extensionset crosscuttingdomain="nomeDomínio">
  [BLOCO COMANDOS XSL]
</captor:extensionset>
```

Todos os comandos declarados dentro de BLOCO COMANDOS XSL, só serão executados se o domínio transversal identificado pelo atributo **`crosscuttingdomain`** estiver sendo combinado com o domínio-base do gabarito no momento da geração do produto. A Figura 6.13 apresenta um trecho de gabarito onde é declarado um conjunto de extensão (linhas 19 à 21).

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:captor="captor.lib.xalan.extensions.CaptorExtension"
        extension-element-prefixes="captor">

    <xsl:output method="text"/>
    <xsl:template match="/">

    package fwslib;

    public class DataBanker {
        private static Vector v = new Vector()
        ...

        public static synchronized void write(SensorReading r)  {
            v.removeElementAt(0);
            SensorReading sr = new SensorReading(r.getID(), r.res, r.value);
            <captor:extensionset crosscuttingdomain="Persistence">
                sr.save();
            </captor:extensionset>
            v.addElement(sr);
        }
        ...
    }
    </xsl:template>
</xsl:stylesheet>
```

Figura 6.13: Gabarito contendo a declaração de um conjunto de extensão

O trecho de gabarito mostrado na Figura 6.13 pertence ao domínio-base de bóias náuticas FWS (Weiss e Lai, 1999) e representa a classe `DataBanker`. Essa classe armazena em memória as leituras enviadas pelos sensores de uma bóia náutica. Como a linha de produtos não possui uma camada de persistência de dados, a medida que o vetor de armazenamento (atributo **`Vector v`**) fica cheio, as leituras antigas são descartadas e liberam espaço para as novas leituras.

Ao combinarmos esse domínio-base com o domínio transversal de persistência, podemos persistir em banco as leituras recebidas e assim evitar a perda de informações. Para tal, é necessário que a classe que contenha os dados que devem ser persistidos, nesse caso a classe `SensorReading`, execute o método `save()` declarado pelo domínio transversal. Visto que a chamada do método `save()` só é válida caso o sistema de bóias tenha a camada de persistência, é declarado um conjunto de extensão que insere o código desejado somente quando o domínio de persistência é utilizado na geração. Também é possível fazer asserções condicionais do tipo **if-else** utilizando uma outra função chamada `iscrosscuttedby`, como mostrado no código a seguir:

```
<xsl:choose>
  <xsl:when test="captor:iscrosscuttedby('nomeDomínio')">
    [BLOCO TESTE VERDADE]
  </xsl:when>
  <xsl:otherwise>
```

```
    [BLOCO TESTE FALSO]
  </xsl:otherwise>
</xsl:choose>
```

O teste lógico é realizado utilizando o elemento XSL `choose` e o nome do domínio transversal fornecido por meio do argumento **`nomeDominio`**. A forma de utilização do comando `iscrosscuttedby` é similar à do comando `extensionset`, entretanto, caso a condição seja satisfeita, isto é, se o domínio transversal consultado fizer parte da combinação, os comandos contidos em BLOCO TESTE VERDADE serão executados, caso contrário, serão executados os comandos de BLOCO TESTE FALSO.

6.5.4 Especificação Transversal

Cada domínio configurado na ferramenta, seja ele base ou transversal, possui um conjunto de propriedades que definem o comportamento do domínio em relação aos outros domínios existentes. Esse conjunto de propriedades, denominado **especificação transversal**, é armazenado dentro do diretório de cada domínio em um arquivo chamado `CrosscuttingInfo.xml`. Os dados contidos na especificação transversal permitem a realização de duas atividades apresentadas na Seção 5.3.1: Informar os domínios transversais compatíveis e declarar os pontos de junção pré-definidos. A Figura 6.14 mostra a estrutura básica dos elementos de uma especificação transversal.

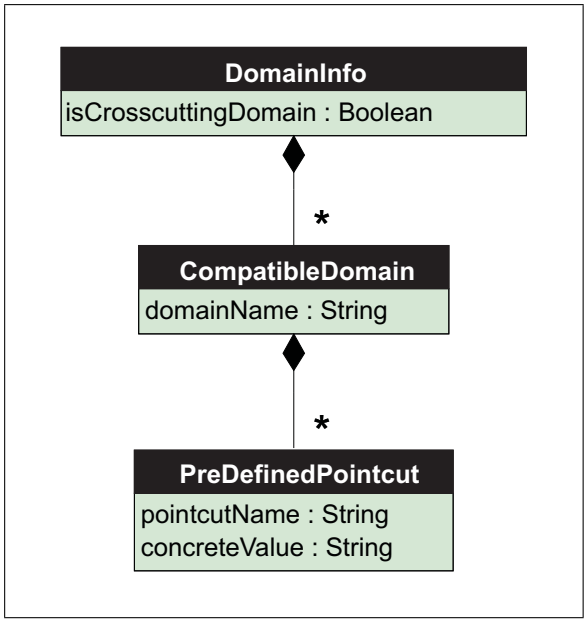

Figura 6.14: Hierarquia da especificação transversal

Caso o domínio da especificação seja um domínio-base, o engenheiro de domínio pode utilizar o elemento `CompatibleDomain` para definir quais dos domínios transversais já configurados podem ser combinados com esse domínio, ou seja, quais são seus domínios compatíveis. A versão

atual da especificação transversal permite a definição de um grupo de zero ou mais domínios transversais compatíveis.

A especificação transversal também pode ser utilizada para a definição dos pontos de junção pré-definidos (Ver Seção 5.2.2) de um domínio-base. Essa definição é feita por meio da declaração de um elemento do tipo `PreDefinedPointcut` para cada PJA contido no domínio transversal compatível que possuir um PJP no domínio-base. A Tabela 6.2 apresenta em detalhes o funcionamento dos componentes de uma especificação transversal.

Nome Elemento	**Descrição**
`DomainInfo`	Nó raiz da especificação transversal. Caso o atributo `isCrosscuttingDomain` esteja marcado como verdadeiro, o domínio da especificação é considerado um domínio transversal, caso contrário, o domínio é considerado um domínio-base. Se o domínio da especificação for um domínio-base ele pode ter um conjunto de elementos do tipo `CompatibleDomain`.
`CompatibleDomain`	Os elementos do tipo `CompatibleDomain` representam cada um dos domínios-transversais compatíveis com um determinado domínio-base. O atributo `domainName` é utilizado para identificar o nome do domínio transversal compatível. Um domínio compatível pode ter um grupo de elementos do tipo `PreDefinedPointcut`.
`PreDefinedPointcut`	Representa um ponto de junção pré-definido (PJP) pelo engenheiro de domínio. Uma vez que o engenheiro de domínio conhece os pontos do domínio-base afetados pelos aspectos de um domínio transversal, ele pode configurar na especificação transversal valores fixos para esses pontos de entrecorte. Esses valores devem ser aplicados quando o domínio-base da especificação e o domínio transversal compatível forem utilizados para a geração de um produto. O conceito de pontos de junção pré-definidos é descrito em detalhes na Seção 5.2.1.

Tabela 6.2: Elementos de uma especificação transversal

Na Figura 6.15 é apresentado um exemplo de especificação transversal. Nesse exemplo é mostrada a especificação transversal do domínio-base de bóias náuticas. Nas linhas 5 e 9 é realizada a declaração de dois domínios transversais compatíveis: *logging* e persistência. Nas linhas 13 e 17 são definidos PJP's para os PJA's `openConnection` e `closeConnection` do domínio de persistência.

6.5.5 Geração de Produtos

Seguindo o processo de engenharia de aplicações mostrado na Seção 5.3.2, o Captor-AO permite ao engenheiro selecionar qual domínio-base e, opcionalmente, quais domínios transversais serão utilizados para a geração do novo produto. O produto final dessa combinação de domínios

```xml
<?xml version="1.0"?>
<DomainInfo>
    <crosscuttingDomain>false</crosscuttingDomain>
    <CompatibleDomains>
        <Domain>
            <domainName>Logging</domainName>
            <PreDefinedPointcuts/>
        </Domain>
        <Domain>
            <domainName>Persistence</domainName>
            <PreDefinedPointcuts>
                <AbstractPointcut>
                    <refName>openConnection</refName>
                    <concreteValue>execution (void fws.FWS.windowOpened(..))</concreteValue>
                </AbstractPointcut>
                <AbstractPointcut>
                    <refName>closeConnection</refName>
                    <concreteValue>execution (void fws.FWS.windowClosing(..))</concreteValue>
                </AbstractPointcut>
            </PreDefinedPointcuts>
        </Domain>
    </CompatibleDomains>
</DomainInfo>
```

Figura 6.15: Exemplo de especificação transversal

terá funcionalidades de um domínio-base e de um ou mais domínios transversais. Dessa forma, cada novo projeto deve manter informações sobre o domínio-base e os domínios transversais combinados. A Figura 6.16 apresenta as entidades que implementam a combinação de domínios no Captor-AO.

A classe `Project` é uma abstração de um projeto do gerador Captor-AO e possui os atributos **`baseDomain`** e **`crosscutingDomains`** que representam o domínio-base do projeto e uma lista de domínios transversais, respectivamente. A escolha dos domínios transversais do projeto é feita a partir do conjunto de domínios compatíveis definidos na especificação transversal do domínio-base (Ver Seção 6.5.4). Esses dados são lidos e interpretados pela classe `CrosscuttingInfo`.

Após a seleção dos domínios transversais, o gerador irá disponibilizar os formulários das LMA's de todos os domínios escolhidos. O processo de instanciação de membros da linha de produtos é baseado no preenchimento de formulários referentes às LMA's de cada domínio envolvido na combinação utilizando a interface gráfica da engenharia de aplicações. Essa interface permite que o engenheiro preencha as variabilidades de todos os domínios em uma única tela. Na Figura 6.17 é mostrada a interface da ferramenta para instanciação de múltiplas LMA's.

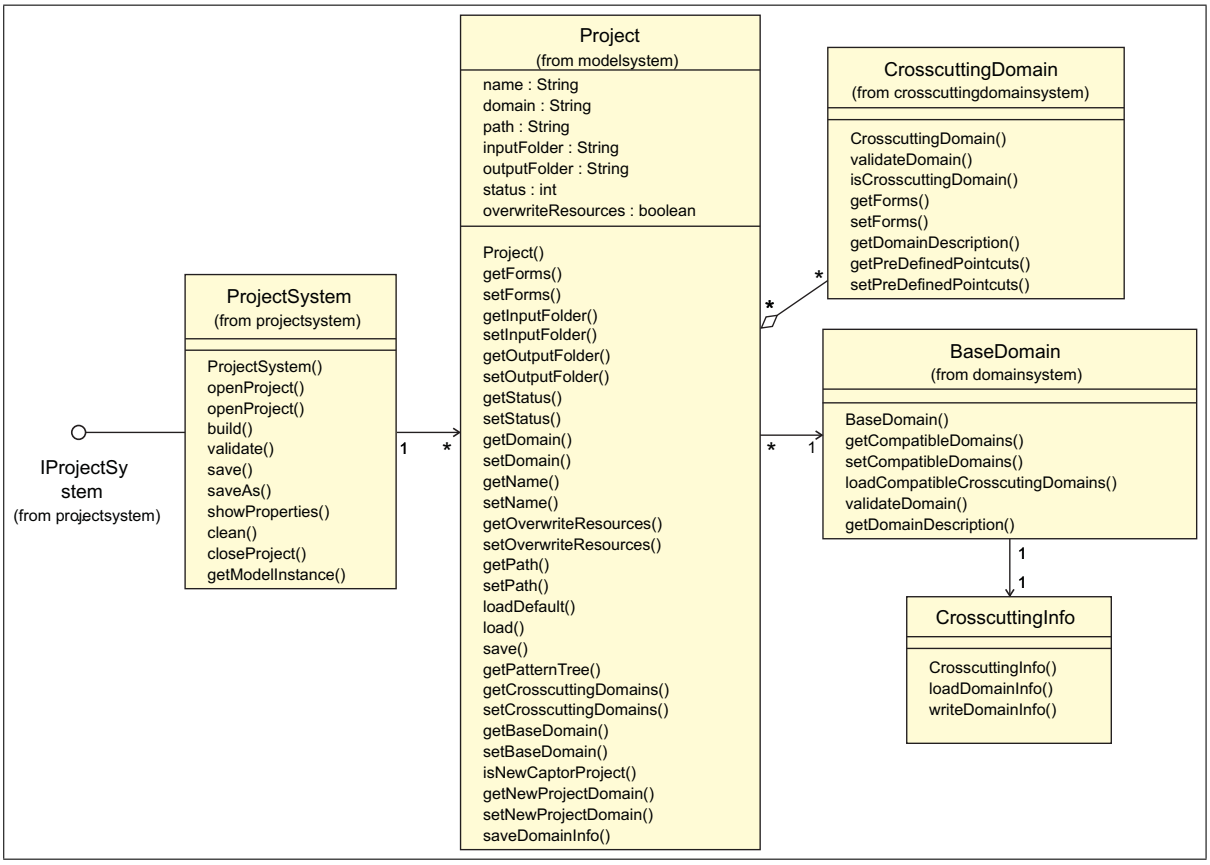

Figura 6.16: Classes utilizadas para combinação de domínios

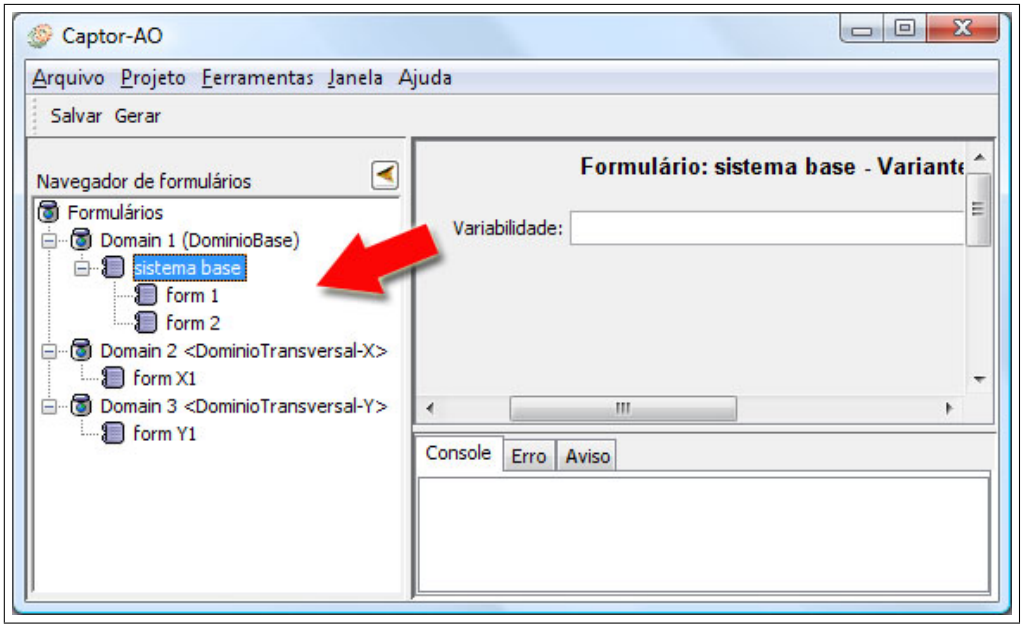

Figura 6.17: Interface utilizada para o preenchimento de variabilidades de diferentes domínios

No painel da esquerda são listados os domínios do projeto corrente. A ordem de visualização começa com o domínio-base do produto que está sendo gerado, seguido pelos domínios transversais escolhidos (identificados pelos caracteres “<” e “>”). A lista de domínios é organizada em

forma de árvore, onde os formulários da LMA são nós filhos do nó principal de cada domínio. O engenheiro de aplicações pode navegar entre os formulários clicando nos respectivos nós na lista de domínios.

Após o preenchimento dos formulários, o gerador deve validar a instância de cada domínio e gerar os artefatos correspondentes. A ferramenta Captor-AO utiliza a abordagem de geração por composição, isto é, a geração de produtos é feita a partir da composição de gabaritos previamente implementados. O processo de geração produz diversos artefatos que formam a implementação do sistema e que são armazenados em um diretório de saída, separados de acordo com o domínio de origem.

Na Figura 6.18 é mostrada a árvore de diretórios para um exemplo de projeto denominado "ProjetoTeste". Todos os projetos criados são armazenados no diretório **projects** que fica na pasta raiz da ferramenta. Dentro do diretório **ProjetoTeste** existem duas pastas, **input** e **output**. Na pasta **input** são guardadas as instâncias LMA fornecidas e na pasta **output** são armazenados os artefatos gerados. Os diretórios em destaque na Figura 6.18, **domain_0** e **domain_1**, possuem os códigos-fonte dos domínios FWS e Persistência, respectivamente.

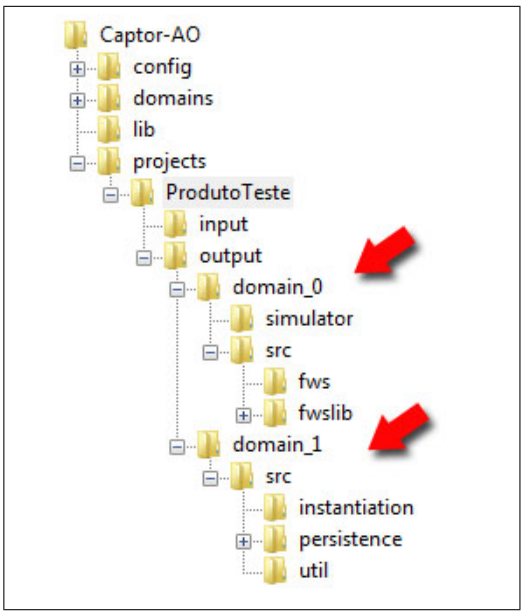

Figura 6.18: Árvore de diretórios para os arquivos gerados

O último passo do processo de instanciação de um novo produto é a compilação do código-fonte gerado. Nessa etapa, o produto resultante é transformado em uma aplicação executável. Para tal, é realizado o *weaving* dos aspectos dos domínios transversais utilizados, isto é, a junção entre os códigos-objeto das classes e dos aspectos empregados. O engenheiro de aplicações deve utilizar um compilador (ou combinador de aspectos) de sua preferência e fornecer como entrada o código-fonte gerado pelo Captor-AO.

6.6 Considerações Finais

Neste Capítulo foi apresentado o gerador de aplicações Captor-AO, uma extensão do gerador configurável Captor, que possui recursos para configuração de novos domínios e geração de produtos adaptados para fornecer suporte à utilização de features transversais em linhas de produtos. Outros trabalhos encontrados na literatura também apresentam ferramentas para a instanciação automática de produtos que oferecem suporte para o reúso de features transversais (Lesaint e Papamargaritis, 2004; Cirilo et al., 2007). A contribuição do gerador Captor-AO, em relação a essas pesquisas, é que ele é capaz de definir relacionamentos de compatibilidade entre os domínios configurados e gerar aplicações por meio do reúso de features provenientes de vários domínios distintos. Outra vantagem do Captor-AO é que ele permite diferenciar as variabilidades funcionais das variabilidades de junção para os domínios transversais, bem como a pré-configuração dos pontos de entrecorte abstratos, facilitando a instanciação pelo engenheiro de aplicações.

A implementação da ferramenta auxiliou na demonstração prática dos conceitos de domínios-base e domínios transversais apresentados nesta pesquisa e também ajudou na descoberta de algumas atividades dos processos de engenharia de domínios e de aplicações propostos. Pode-se utilizar essa ferramenta para realizar experimentos futuros com a finalidade de avaliar o processo de combinação de domínios utilizando linhas de produtos reais.

O gerador Captor-AO foi configurado e utilizado para realizar a geração de artefatos para dois estudos de caso: combinação do domínio-base de bóias náuticas (FWS) com o domínio transversal de persistência e combinação do domínio-base de gestão de recursos de negócios (GRN) com o domínio transversal de controle de acesso. Visto que o primeiro estudo, bóias náuticas e persistência, é o mais completo e exercita a maioria das funcionalidades desenvolvidas, são apresentadas no próximo Capítulo as atividades realizadas durante a condução desse estudo. Todos os domínios-base e transversais configurados no gerador Captor-AO estão disponíveis para *download* na página do projeto[5].

[5] `http://captor.googlecode.com`

CAPÍTULO

7

Estudo de Caso

7.1 Considerações Iniciais

Este capítulo descreve um estudo de caso sobre a configuração e geração de produtos utilizando um domínio transversal e um domínio-base. Esse estudo de caso é apresentado com o objetivo de auxiliar no entendimento dos conceitos propostos, além de servir como demonstração da utilização prática do gerador Captor-AO. Para esse estudo, foi realizada a configuração de um domínio transversal de persistência de dados (Camargo, 2006) e também a combinação desse domínio com o domínio-base de bóias náuticas (Weiss e Lai, 1999).

Este Capítulo está organizado da seguinte forma: na Seção 7.2 é apresentado em detalhes o domínio transversal de persistência de dados e o processo de configuração desse domínio no gerador Captor-AO; na Seção 7.3 é apresentado em detalhes o domínio-base de bóias náuticas e como esse domínio foi adaptado para permitir a combinação com o domínio de persistência; na Seção 7.4 são apresentadas as atividades realizadas durante a geração de um produto especificado a partir da combinação de ambos os domínios; finalmente, na Seção 7.5 são apresentadas as considerações finais deste Capítulo.

7.2 Domínio Transversal de Persistência

Nas próximas Seções é apresentado o processo de configuração de um domínio transversal de persistência de dados. A configuração do domínio consistiu na adaptação do framework de persistência de dados (Camargo, 2006) como um domínio do gerador Captor-AO, seguindo o processo

de criação de domínios transversais descrito na Seção 5.3.1. A seguir são mostrados em detalhes os passos realizados durante a configuração do domínio transversal no gerador Captor-AO.

7.2.1 Análise e Implementação do Domínio

O domínio transversal de persistência de dados visa a inclusão da camada de persistência de informações em um sistema base. Para a criação do domínio transversal foi utilizada a implementação do framework de persistência apresentado por Camargo (2006). Esse framework tem como objetivo facilitar o desenvolvimento de uma aplicação orientada a objetos que utiliza banco de dados relacional.

O framework consiste de duas partes que foram implementadas em módulos distintos. A primeira parte, chamada de "operações persistentes", é um conjunto de operações de persistência que devem ser herdadas por classes de aplicação persistentes e que podem ser utilizadas para armazenar, remover e atualizar informações, e realizar consultas no banco de dados. Uma classe de aplicação persistente é uma classe cujos objetos devem ser persistidos em alguma tabela do banco de dados. Cada classe de aplicação persistente possui uma tabela correspondente no banco de dados com o mesmo nome da classe.

A estratégia de implementação adotada no projeto do framework foi introduzir todas as operações de persistência em uma interface e fazer com que as classes da aplicação que devem ser persistidas implementem essa interface. Algumas das operações de persistência disponibilizada na interface do framework podem ser vistas na Tabela 7.1.

Operação	**Descrição**
`save()`	Utilizada para armazenar o objeto alvo da chamada em sua respectiva tabela do banco de dados. Retorna "verdadeiro" se a operação foi realizada com sucesso.
`delete()`	Utilizada para remover o objeto alvo da chamada de sua respectiva tabela do banco de dados. Retorna "verdadeiro" se a operação foi realizada com sucesso.
`find()`	Utilizada para localizar e recuperar o objeto alvo no banco de dados. Se o objeto foi encontrado, retorna os dados desse objeto, caso contrário retorna vazio.
`findAll()`	Utilizada para recuperar todos os registros da tabela representada pela classe do objeto alvo. Retorna um conjunto com todos os registros.

Tabela 7.1: Operações do framework de persistência

A segunda parte, chamada de "conexão", é referente ao interesse de conexão com o banco de dados e é responsável pela identificação dos locais do código-base em que a conexão deve ser aberta e fechada. A primeira parte depende da segunda para conseguir executar as operações de persistência.

Na Figura 7.1 (Camargo, 2006) é mostrado um diagrama das classes e aspectos que compõem os dois módulos do framework: Operações Persistentes e Conexão. Nessa figura, os aspectos são representados por retângulos destacados em cinza, e os relacionamentos de associação que partem dos aspectos para alguma entidade representam que o aspecto afeta a entidade entrecortando a execução/chamada de seus métodos ou introduzindo operações, por meio de declarações inter-tipo. Os demais relacionamentos possuem semântica convencional da UML. Todos os métodos com o estereótipo «`hook`» são abstratos e devem ser sobrepostos pelo engenheiro de aplicações em classes concretas.

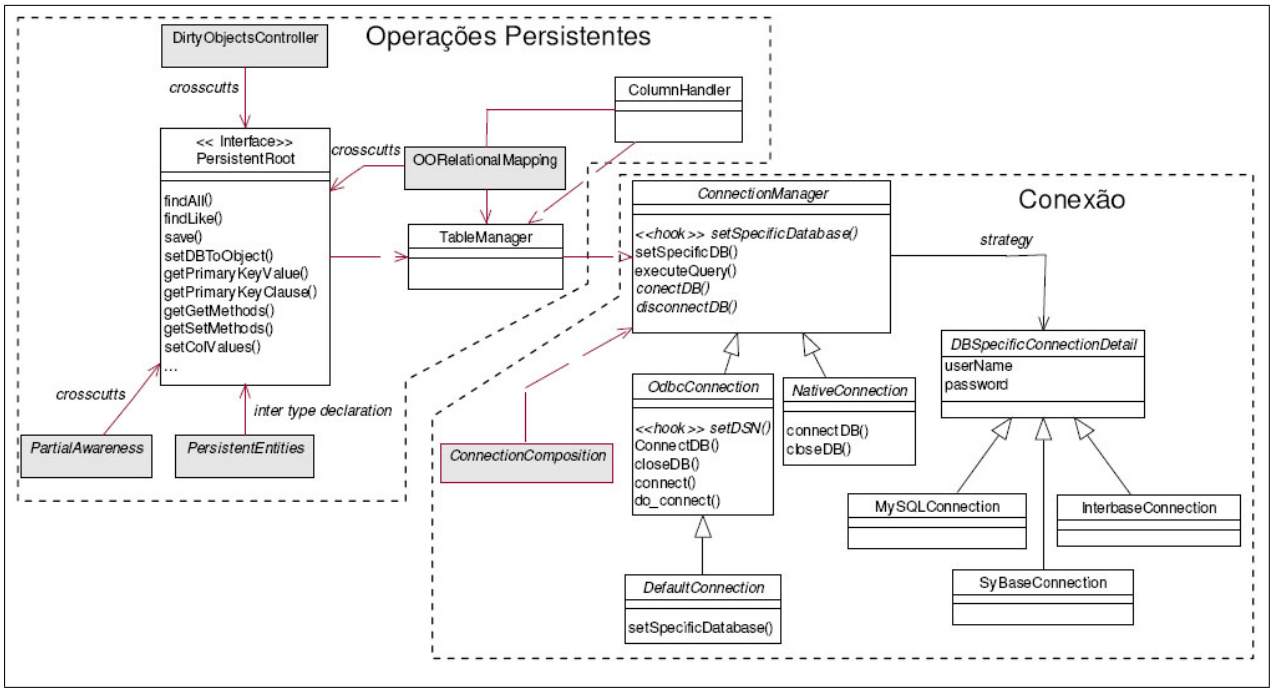

Figura 7.1: Estrutura do framework de persistência (Camargo, 2006)

A classe `TableManager` é responsável por montar dinamicamente as declarações SQL e executar as operações de persistência. O aspecto `DirtyObjectsController` é responsável por "marcar" os objetos que foram alterados em memória para que essas alterações sejam refletidas no disco em momentos apropriados.

Ao invés de realizar modificações invasivas em todas as classes de aplicação durante o processo de reúso do framework, deve-se criar um aspecto concreto que deve estender o aspecto abstrato `PersistentEntities`. Esse aspecto concreto deverá introduzir a declaração `implements PersistentRoot` nas classes utilizando declarações inter-tipo da linguagem AspectJ. A interface `PersistentRoot` declara um conjunto de atributos e operações de persistência (Ver Tabela 7.1) que serão herdados pelos sub-tipos dessa interface. As classes de aplicação cujos objetos devem ser persistidos em alguma tabela do banco de dados devem implementar a interface `PersistentRoot` e possuir correspondência com o banco de dados. Por questão de simplicidade, tais classes serão referenciadas deste ponto em diante como **classes persistentes** apenas.

Na Figura 7.2 pode ser observado um trecho de código utilizado para a declaração das classes persistentes fictícias `Cliente`, `Fornecedor` e `Produto`.

```
1  package instantiation;
2
3  import persistence.*;
4  import util.*;
5  import java.sql.*;
6  import java.util.*;
7
8  public aspect MyPersistentEntities extends PersistentEntities {
9
10     declare parents: Cliente implements PersistentRoot;
11     declare parents: Fornecedor implements PersistentRoot;
12     declare parents: Produto implements PersistentRoot;
13
14 }
```

Figura 7.2: Exemplo de instanciação do aspecto `PersistentEntities`

O mapeamento dos atributos das classes persistentes com o banco de dados é feito internamente pelo framework. O aspecto `OORelationalMapping` introduz em cada classe persistente meta-informações sobre sua correspondência com o banco de dados, como por exemplo: o nome da tabela correspondente, o número de colunas da tabela e o nome dessas colunas.

O aspecto `ConnectionComposition`, é responsável por entrecortar pontos de junção do código-base, e utilizar os métodos `connectDB()` e `disconnectDB()` da classe `ConnectionManager` para abrir e fechar conexões com o banco de dados. Os pontos de junção devem ser fornecidos pelo engenheiro de aplicações por meio da criação de um aspecto concreto que estende o aspecto `ConnectionComposition` em tempo de composição.

A escolha das variabilidades de conexão é feita estendendo-se uma das classes filhas de `ConnectionManager`. Nota-se na Figura 7.1 a existência de uma classe abstrata chamada `DefaultConnection`. Essa classe é responsável por definir as variabilidades padrão para a conexão, que consiste em uma conexão feita via ODBC do Windows com banco de dados MySQL. Caso a conexão padrão seja escolhida, o responsável pelo desenvolvimento deve estender diretamente essa classe. Se alguma outra variabilidade for desejada, basta estender qualquer uma das outras classes (`InterbaseConnection` ou `SyBaseConnection`).

O framework de persistência foi implementado com duas variabilidades mutuamente exclusivas chamadas **Consciência Total** e **Consciência Parcial**. No caso da persistência, durante o desenvolvimento da aplicação é necessário ter "consciência" de algumas operações. As operações que tratam de armazenamento e atualização podem ser completamente tratadas pelo framework e assim, podem ser negligenciadas durante o desenvolvimento de um sistema. Por exemplo, pode-se implementar um aspecto que armazena um objeto no banco de dados sempre que o construtor desse objeto for executado. Também pode ser implementado um aspecto de atualização, que entrecorta as chamadas aos métodos que modificam os valores dos atributos (*setters*) e realizam as atualizações no banco de dados. Nota-se que, nesses dois casos, há pontos de junção adequados que podem ser entrecortados, independentemente da linguagem de programação utilizada.

O mesmo não ocorre com as operações de remoção e busca, pois elas devem ser explicitamente chamadas em momentos definidos pelo engenheiro da aplicação. Quando a linguagem Java é utilizada, não há pontos de junção adequados no código-base que podem ser entrecortados para a remoção de um objeto. Um possível ponto de junção para a remoção de um objeto seria sua destruição, isto é, quando o seu destrutor é executado. Porém, no caso da linguagem Java isso é feito pela JVM (Java Virtual Machine) em momentos não-determinísticos.

Dessa forma, a variabilidade Consciência Total delega para o engenheiro de software a responsabilidade de invocar todos os métodos de persistência disponibilizados pelo framework (Ver Tabela 7.1), enquanto que a outra variabilidade, Consciência Parcial, habilita a execução automática de algumas operações de persistência, evitando assim que o engenheiro de aplicações tenha que se preocupar com a maioria das operações realizadas pelo framework, com exceção das operações de remoção (`delete()`) e busca (`find()`).

A variabilidade Consciência Total foi implementada como variabilidade padrão no framework e não requer nenhuma codificação adicional. Para escolher a variabilidade Consciência Parcial basta que o aspecto `PartialAwareness`, mostrado na Figura 7.1, seja adicionado ao projeto. Esse aspecto tem conhecimento dos pontos que devem ser entrecortados, não havendo necessidade de concretização por parte do engenheiro de software.

O framework de persistência ainda possui várias outras features (*Caching* e *Pooling*, por exemplo) que não foram citadas para evitar o aumento desnecessário de complexidade do estudo de caso, visto que essas features não exercitam nenhum dos novos processos propostos. Maiores detalhes sobre o framework de persistência podem ser encontrados em Camargo et al. (2003) e Camargo (2006). Nas próximas Seções são abordadas as atividades realizadas para a adaptação do framework de persistência como um domínio transversal do gerador Captor-AO.

7.2.2 Criação do projeto do domínio transversal

A primeira tarefa para a configuração de um novo domínio é a criação de um projeto na ferramenta Captor-AO. O projeto manterá informações como o nome do domínio e os diretório onde serão armazenados os artefatos do domínio. Na Figura 7.3 são mostradas as interfaces utilizadas para a criação do novo domínio no gerador Captor-AO.

Como mostrado na Figura 7.3, o engenheiro de domínio deve escolher a opção “New Captor Project”, clicar no botão “Próximo” e em seguida preencher o nome do domínio e o diretório de armazenamento do projeto. Na interface seguinte, o engenheiro do domínio deve informar o tipo do domínio, isto é, se ele é um domínio-base ou um domínio transversal, conforme mostrado na Figura 7.4. Nesse estudo de caso, foi escolhida a opção “Domínio Transversal” para a configuração do domínio de persistência.

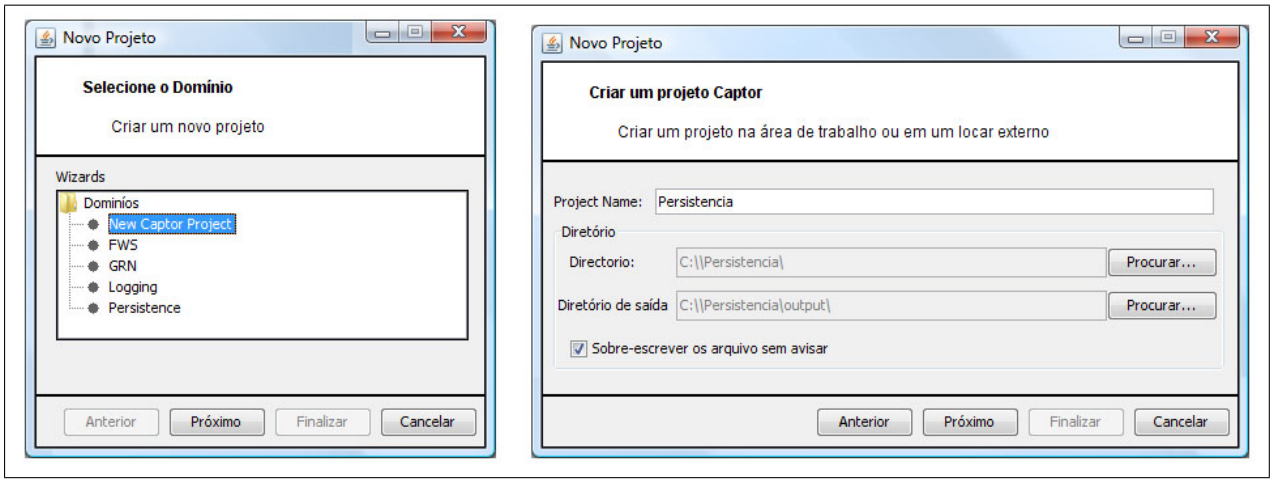

Figura 7.3: Criação do projeto para o domínio de persistência de dados

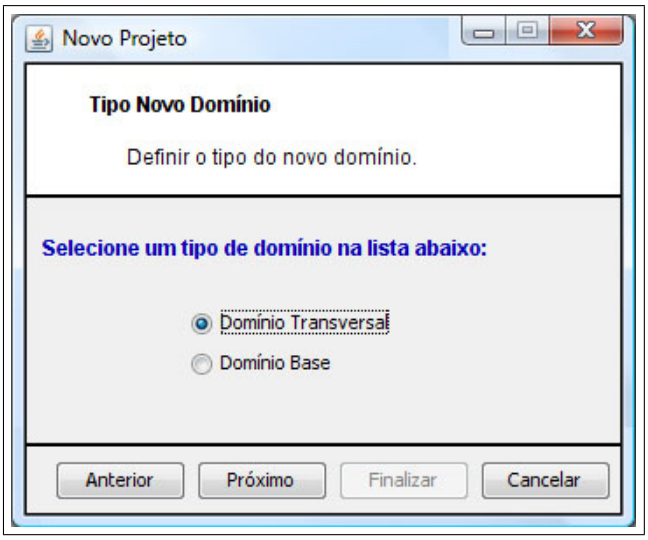

Figura 7.4: Escolhendo o tipo do domínio

7.2.3 Identificação dos Pontos de Junção Abstratos

Um ponto de junção abstrato (PJA) proporciona uma interface de entrecorte genérica, tornando a implementação das features transversais que atuam sobre aquele ponto de junção menos dependente do código-base (Ver Seção 5.2.1). Foram identificados três PJA's na implementação do domínio de persistência. Os dois primeiros são referentes ao interesse de conexão e são representados pelos *pointcuts* abstratos `openConnection` e `closeConnection` declarados dentro do aspecto `ConnectionComposition`. Esses PJA's podem ser concretizados em tempo de reúso pelo engenheiro de aplicações com as assinaturas dos métodos da aplicação base em que serão executadas as operações de conexão e desconexão com o banco de dados. Também é possível que o domínio-base compatível defina PJP's para esses PJA's do domínio transversal, evitando assim que o engenheiro de aplicações precise preencher esses dados.

O último PJA está relacionado com o interesse de operações persistentes. Esse PJA é declarado dentro do aspecto `PersistentEntities` que é responsável pela injeção da interface

`PersistentRoot` em cada classe persistente da aplicação. Para que o framework possa diferenciar as classes persistentes das demais classes, esse PJA deve ser concretizado com uma lista contendo os nomes de todas as classes que devem ser persistidas pelo framework. A Tabela 7.2 apresenta um resumo dos PJA's identificados no domínio de persistência analisado.

PJA	**Finalidade**
AbrirConexão	Representa os métodos da aplicação base que irão disparar a conexão com o banco de dados.
FecharConexão	Representa os métodos da aplicação base que irão disparar a desconexão com o banco de dados.
ClassesPersistentes	Lista de todas as classes da aplicação base que devem ser persistidas pelo domínio transversal.

Tabela 7.2: Pontos de Junção Abstratos do domínio de persistência

7.2.4 Criação da linguagem de modelagem de aplicações

Conforme visto na Seção 2.5.3, a Linguagem de Modelagem de Aplicações (LMA) é uma linguagem de alto nível de abstração utilizada para representar aplicações. O gerador Captor-AO permite a criação de uma LMA capaz de representar os produtos de um domínio. Uma vez que as features obrigatórias e os artefatos fixos do domínio estão presentes em todos os produtos, a LMA do domínio pode diferenciar todos os possíveis produtos em termos das variabilidades utilizadas em cada aplicação. Nas Tabelas 7.3 e 7.4 são listadas as variabilidades funcionais e as variabilidades de junção (Ver Seção 5.2.3) do domínio de persistência consideradas para a criação da LMA.

Variabilidade	**Descrição**
Banco de Dados	Nome do banco de dados relacional utilizado. A implementação utilizada nesse estudo de caso fornece três opções de banco de dados: MySQL, Interbase e Sybase.
Login	Nome de usuário utilizado para acessar o banco de dados.
Senha	Senha de segurança do banco de dados.
Nome DSN	Nome do *driver* ODBC de conexão utilizado pelo sistema operacional.
Tipo Consciência	Escolha entre as estratégias de consciência total ou parcial.

Tabela 7.3: Variabilidades funcionais do domínio de persistência

A definição da LMA do domínio de persistência é feita no gerador Captor-AO por meio da declaração de formulários que serão utilizados durante a engenharia de aplicações para especificar as variabilidades de cada produto. Nesse estudo de caso, a LMA do domínio de persistência foi modelada de acordo com a estrutura apresentada na Figura 7.5.

Na Figura 7.5 estão representados os três formulários principais da LMA do domínio de persistência. No formulário "Dados Conexão" são preenchidas as variabilidades referentes ao interesse

Variabilidade	Descrição
Ponto(s) de Conexão	Assinatura(s) do(s) método(s) em que ocorre a conexão com o banco de dados. A conexão é aberta ANTES que os métodos listados sejam executados.
Ponto(s) de Desconexão	Assinatura(s) do(s) método(s) em que a conexão é encerrada. A conexão é fechada APÓS a execução dos métodos listados.
Classes Persistentes	Nomes qualificados das classes persistentes da aplicação base.

Tabela 7.4: Variabilidades de junção do domínio de persistência

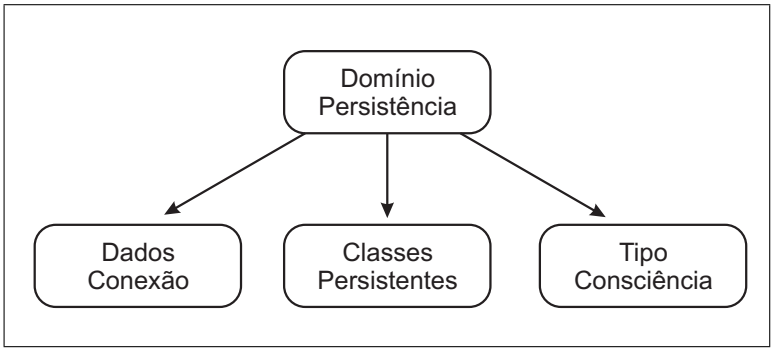

Figura 7.5: Árvore de formulários do domínio de persistência

de conexão com o banco de dados. O formulário "Classes Persistentes" é utilizado para definir as classes persistentes do produto e no formulário "Tipo Consciência" é escolhida a estratégia de consciência (Total ou Parcial) adotada pelo domínio. A Figura 7.6 apresenta a interface utilizada para criação do formulário "Dados Conexão".

Para a criação de um novo formulário da LMA é necessário que o engenheiro de domínio informe o nome e os campos desse formulário. Cada formulário pode conter um ou mais campos gráficos (caixas de texto, tabelas, caixas de seleção, etc) que serão preenchidos pelo engenheiro de aplicações. Pode-se observar na Figura 7.7 a interface utilizada para a inserção do elemento de formulário responsável pela variabilidade "Banco de Dados". Esse elemento é modelado como uma caixa de seleção que exibirá três opções de banco de dados: MySQL, Interbase e Sybase.

O gerador Captor-AO disponibiliza um campo especialmente desenvolvido para a modelagem das variabilidades de junção de um domínio transversal chamado `abstractpointcutpanel` (Ver Seção 6.5.2). Cada `abstractpointcutpanel` possui um identificador que pode ser utilizado para acessar tanto o valor concreto do PJA, fornecido pelo engenheiro de aplicações, quanto o valor pré-definido (Ver Seção 5.2.1) armazenado na especificação transversal. Na Figura 7.8 é mostrada a configuração do PJA "AbrirConexão" utilizando um elemento de formulário do tipo `abstractpointcutpanel`. Nota-se na Figura 7.8 que esse elemento pode ser utilizado para: (i) definir o identificador do PJA, (ii) fornecer uma breve descrição sobre o comportamento do PJA e (iii) para o preenchimento dos pontos de junção concretos.

O próximo formulário a ser criado é o formulário "Classes Persistentes". Esse formulário é utilizado para representar as classes de um domínio-base que serão persistidas em banco.

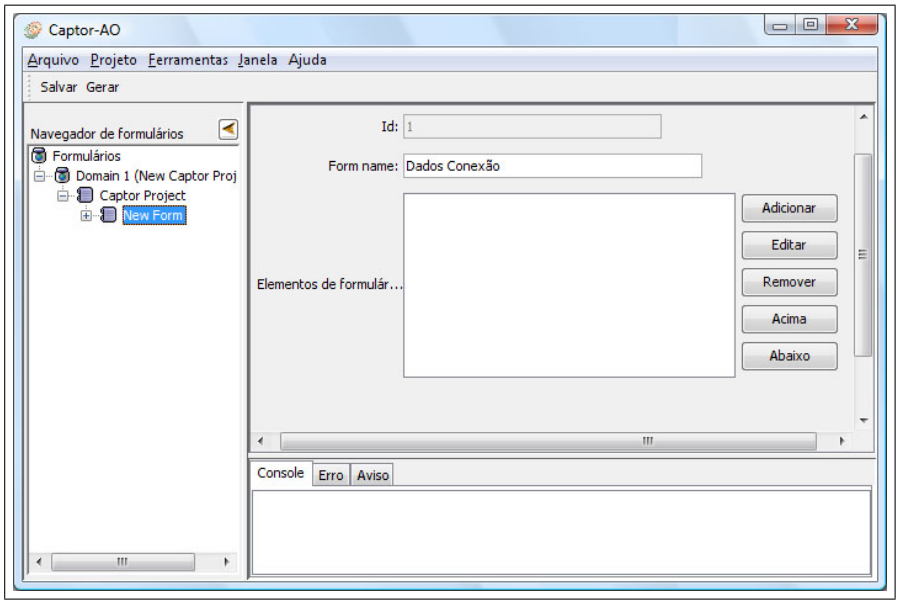

Figura 7.6: Interface utilizada para criação da LMA do domínio de persistência

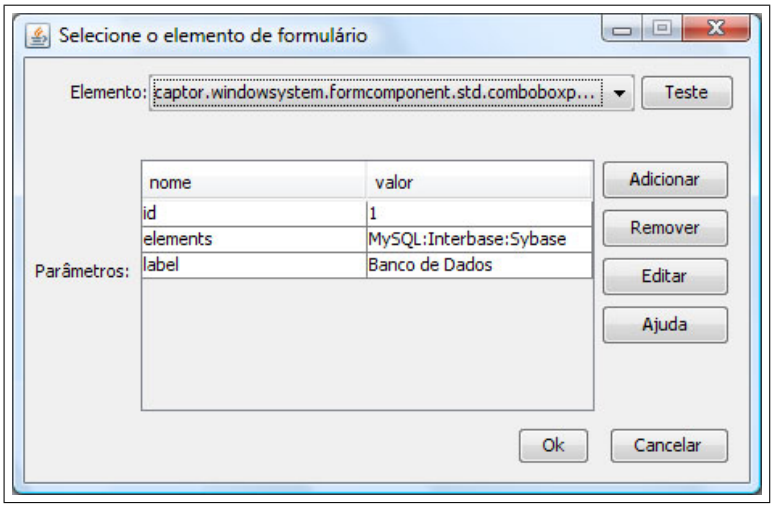

Figura 7.7: Elemento de formulário da variabilidade Banco de Dados

Tais classes, são determinadas pelo PJA "ClassesPersistentes", entretanto, graças a cardinalidade desse PJA, seriam necessários diversos elementos do tipo `abstractpointcutpanel` para determinar todas as classes persistentes. Entretanto, pode-se utilizar um outro elemento, chamado `abstractpointcutlist`, que é capaz de representar um PJA multi-valorado. Na Figura 7.9 é mostrada a configuração do PJA "ClassesPersistentes" utilizando o elemento `abstractpointcutlist`. O engenheiro de aplicações deve preencher a lista mostrada na Figura 7.9 com os nomes qualificados das classes persistentes do produto.

O último formulário da LMA do domínio transversal de persistência é o formulário "Tipo Consciência", que é o formulário responsável pela escolha entre a estratégia de consciência total ou consciência parcial (Ver Seção 7.2.1). Esse formulário é formado apenas por uma caixa de seleção de exibirá ambas as opções para o engenheiro de aplicações. Na Figura 7.10 é mostrada a caixa de seleção criada para o formulário "Tipo Consciência".

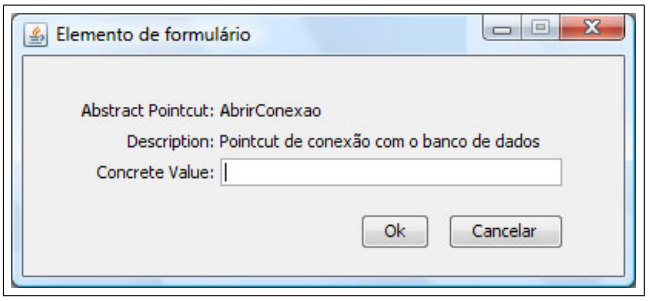

Figura 7.8: Elemento de formulário do PJA: "AbrirConexão"

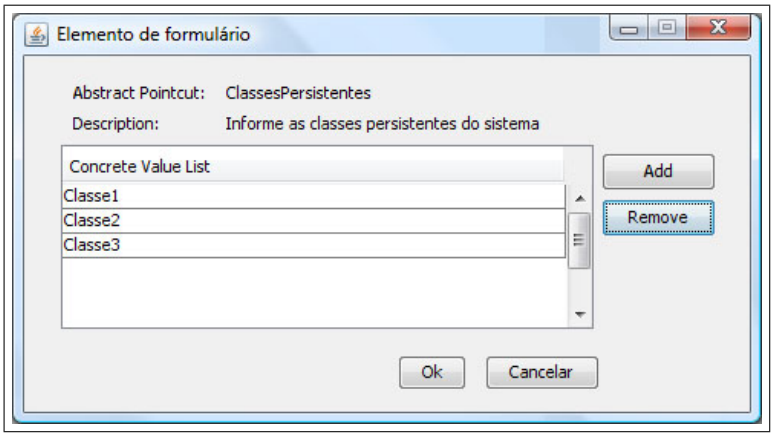

Figura 7.9: Elemento de formulário do PJA: "ClassesPersistentes"

7.2.5 Implementação dos Gabaritos

A próxima atividade na configuração do domínio de persistência é a implementação dos gabaritos. Os gabaritos são utilizados para produzir artefatos concretos, tais como: código-fonte, casos de testes e documentos, de acordo com a instância LMA fornecida. São implementados gabaritos apenas para os artefatos que podem variar de acordo com o produto gerado, visto que o gerador pode copiar os artefatos fixos diretamente para o diretório de saída do produto.

Para o domínio de persistência, deve-se criar gabaritos para as classes e aspectos contidos em um pacote do framework chamado `Instantiation`. Esse pacote foi originalmente projetado para abrigar as classes utilizadas para a instanciação caixa branca do framework. No sentido de possibilitar a geração automática de produtos, os gabaritos desse domínio devem "simular" a instanciação dessas classes, tal como se ela estivesse sendo feita manualmente, ou seja, o código

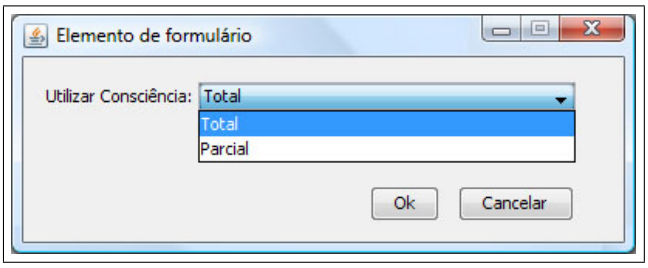

Figura 7.10: Caixa de seleção utilizada para escolha do tipo de consciência

gerado deve realizar a herança das classes genéricas e concretizar os PJA's desse domínio. Os gabaritos utilizados pelo Captor-AO são implementados utilizando a linguagem de gabaritos XSL e utilizam como dados de entrada uma instância LMA armazenada na forma um arquivo XML.

Na Figura 7.11 é visto um trecho do gabarito responsável pela especificação do banco de dados e do *driver* DSN do produto. O processamento desse gabarito resulta em uma classe chamada `MyODBCConnection`. Essa classe é responsável pela herança do comportamento específico do banco escolhido. O gabarito declara duas variáveis (linhas 5 a 10) chamadas `BancoDados` e `DSN`, que realizam a leitura dos valores fornecidos pelo engenheiro de aplicações para as variabilidades "Banco de Dados" e "Nome DSN". Os valores dessas variáveis serão utilizados para substituir os marcadores (*placeholders*) existentes nas linhas 15 e 18 durante o processamento do gabarito.

```
1  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
2      <xsl:output method="text"/>
3      <xsl:template match="/">
4
5          <xsl:variable name="BancoDados">
6              <xsl:value-of select="/forms/form[@id='1.1']/data/combo[@name='nomeBanco']"/>
7          </xsl:variable>
8          <xsl:variable name="DSN">
9              <xsl:value-of select="/forms/form[@id='1.1']/data/textatt[@name='nomeDSN']"/>
10         </xsl:variable>
11
12         package instantiation;
13         import persistence.connection.pre_weaved.*;
14
15         public class MyODBCConnection extends <xsl:value-of select="$BancoDados"/> {
16
17             public String setDSN(){
18                 return "<xsl:value-of select="$DSN"/>";
19             }
20         }
21
22     </xsl:template>
23 </xsl:stylesheet>
```

Figura 7.11: Gabarito da classe `MyODBCConnection`

A linguagem de gabaritos do gerador Captor-AO possui uma extensão própria para a manipulação de PJA's. Essa extensão inclui novos comandos que realizam a leitura do valor concreto de um PJA a partir do identificador definido na LMA do domínio transversal. A Figura 7.12 apresenta o gabarito da classe `MyConnectionComposition`, que foi escrito utilizando essas novas funções de manipulação de PJA's. Nesse gabarito são declarados dois elementos do tipo `abstractpointcut` (linhas 17 e 18), que têm a função de realizar a leitura dos valores concretos dos PJA's "AbrirConexao" e "FecharConexao" diretamente da instância LMA fornecida.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:captor="captor.lib.xalan.extensions.CaptorExtension"
                extension-element-prefixes="captor">

    <xsl:output method="text"/>
    <xsl:template match="/">

package instantiation;

import persistence.connection.*;


public aspect MyConnectionComposition extends ConnectionComposition {

    public pointcut openConnection(): <captor:abstractpointcut refname="AbrirConexao"/>;
    public pointcut closeConnection() : <captor:abstractpointcut refname="FecharConexao"/>;

    public String getNameOfConnectionVariabilitiesClass(){
        return "instantiation.MyODBCConnection";

    }
}

    </xsl:template>
</xsl:stylesheet>
```

Figura 7.12: Gabarito da classe `MyODBCConnection`

Uma outra abordagem para esse gabarito seria realizar a leitura dos valores de cada PJA utilizando variáveis parecidas com as mostradas na Figura 7.11. Contudo, a utilização desses novos comandos possibilita um tratamento diferenciado de determinadas características próprias dos PJA's, como a definição de pontos de junção pré-definidos apresentada na Seção 5.2.1.

7.2.6 Mapeamento entre LMA e Gabaritos

O engenheiro de domínio deve criar um arquivo de mapeamento para guiar a transformação dos gabaritos durante a geração dos produtos. Esse arquivo é um *script* escrito na linguagem MTL (Shimabukuro, 2006) (Ver Seção 4.4) que permite definir quais gabaritos devem ser processados de acordo com a especificação LMA fornecida. O arquivo contendo o mapeamento deve ser armazenado no diretório raiz do domínio com o nome `rules.xml`. Na Figura 7.13 é apresentado um trecho do mapeamento MTL criado para o domínio de persistência.

O trecho de código apresentado na Figura 7.13 é dividido em dois módulos: `main` (linha 2) e `tasks` (linha 16). No módulo `main` são declarados os elementos responsáveis pela geração dos artefatos, denominados `callTask`. Cada elemento `callTask` dispara o processamento de um gabarito e produz um determinado artefato como saída. O gerador Captor-AO realiza o processamento dos gabaritos seguindo a ordem sequencial de declaração dos elementos `callTask` dentro do módulo `main`, isto é, o primeiro elemento declarado é processado, em seguida o segundo e assim por diante.

```
1  <?xml version="1.0"?>
2      <main>
3
4          <callTask id="MyODBC_instantiation"/>
5          <callTask id="MyConnectionComposition_instantiation"/>
6          <callTask id="MyPersistentEntities_instantiation"/>
7
8          <if test="equal(/form[@id='tipoConsciencia']/data,'Consciência Parcial')">
9              <callTask id="PartialAwareness"/>
10         </if>
11
12     </main>
13
14     <!--_____________________________________________________________-->
15
16     <tasks>
17         <task id="MyODBC_instantiation">
18             <compose>
19                 <template>instantiation\MyODBCConnection.java</template>
20                 <newFilename>src\instantiation\MyODBCConnection.java</newFilename>
21             </compose>
22         </task>
23 ...
```

Figura 7.13: Mapeamento MTL para os gabaritos do domínio de persistência

O mapeamento entre um gabarito e seu respectivo artefato de saída é feito no segundo módulo, `tasks`. Nesse módulo são declarados o gabarito e o artefato resultante de cada um dos elementos `callTask` definidos no primeiro módulo. Esse mapeamento é realizado utilizando um identificador que associa cada elemento `callTask` a um elemento chamado `task`. O elemento `task`, por sua vez, possui dois atributos: **`template`** e **`newFileName`**. Esses atributos representam o gabarito a ser processado e o caminho do arquivo de saída, respectivamente.

Nota-se na linha 8 da Figura 7.13 que é executado um teste condicional para verificar se, na instância da LMA, foi escolhida a variabilidade Consciência Parcial. Em caso afirmativo, é realizada a geração do aspecto responsável por essa estratégia. Não é necessário efetuar o teste lógico para a variabilidade Consciência Total porque ela já é utilizada como *default* na implementação do domínio de persistência.

Após a conclusão dessa atividade, o domínio de persistência está quase pronto para ser utilizado. A última atividade a realizar é a instalação dos artefatos criados no repositório de domínios do gerador Captor-AO. Essa instalação é feita automaticamente pelo gerador clicando no botão "Gerar" (Ver Figura 7.14) disponibilizado na interface de engenharia de domínio.

7.3 Domínio-Base de Bóias Náuticas

O próximo passo no estudo de caso realizado envolveu a definição de um domínio base compatível com o domínio de persistência. Para tal, foi escolhido o domínio de bóias náuticas FWS proposto por Weiss e Lai (1999). Esse domínio foi escolhido por se tratar de um domínio que

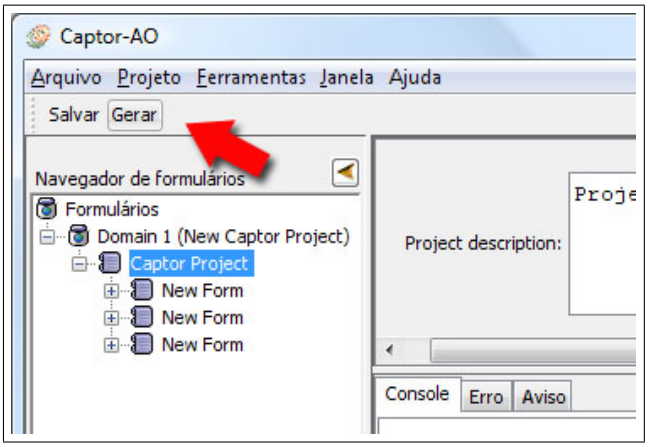

Figura 7.14: Instalação do domínio de persistência no gerador Captor-AO

não implementa a camada de persistência de dados e que suporta a combinação com os aspectos do domínio transversal de persistência, além de já ter sido utilizado no trabalho de Shimabukuro (2006) e portanto, já ter sido implementado anteriormente.

O domínio FWS representa uma família de produtos de sistemas embarcados utilizados em bóias náuticas. As bóias náuticas estão localizadas em alto mar e realizam a medição da velocidade do vento. Cada bóia é equipada com um ou mais sensores de vento, um transmissor de rádio e um computador de bordo. Vários sensores de diferentes resoluções são espalhados pela superfície da bóia para captar a velocidade do vento proveniente de várias direções. Cada bóia possui um computador de bordo que mantém o histórico das leituras da velocidade do vento captada por cada sensor. Em intervalos regulares, o computador de bordo calcula a média ponderada das medições realizadas pelos sensores e emite essa informação por meio do transmissor de rádio para uma torre de controle que monitora as velocidades captadas por diversas bóias espalhadas pelo mar.

O domínio FWS já havia sido configurado no gerador Captor durante a pesquisa de Shimabukuro (2006), entretanto, foram realizadas modificações na sua implementação visando a combinação com o domínio transversal de persistência. Para tal, foram executadas as três atividades de definição de domínios compatíveis apresentadas na Seção 5.3.1: Informar o domínio transversal compatível, incluir os CE's e definir PJP's.

7.3.1 Informação do Domínio Compatível e Definição de PJP's

As atividades de informação de compatibilidade e definição de PJP's foram realizadas por meio da especificação transversal do domínio FWS, conforme descrito na Seção 6.5.4. A Figura 7.15 apresenta o trecho de código da especificação transversal onde é definida a compatibilidade com o domínio de persistência (linha 6).

Para a instanciação dos PJA's "AbrirConexão" e "FecharConexão" (Ver Seção 7.2.3) é necessário que o engenheiro de aplicações possua conhecimentos técnicos sobre as classes e os métodos

```
1  <?xml version="1.0"?>
2  <DomainInfo>
3      <crosscuttingDomain>false</crosscuttingDomain>
4      <CompatibleDomains>
5          <Domain>
6              <domainName>Persistencia</domainName>
7              <PreDefinedPointcuts>
8                  <AbstractPointcut>
9                      <refName>AbrirConexao</refName>
10                     <concreteValue>execution (public static void main(..))</concreteValue>
11                 </AbstractPointcut>
12                 <AbstractPointcut>
13                     <refName>FecharConexao</refName>
14                     <concreteValue>execution (public static void main(..))</concreteValue>
15                 </AbstractPointcut>
16             </PreDefinedPointcuts>
17         </Domain>
18     </CompatibleDomains>
19 </DomainInfo>
```

Figura 7.15: Especificação transversal do domínio FWS

utilizados na implementação do domínio FWS, uma vez que esses PJA's são concretizados com as assinaturas dos métodos em que serão realizadas a conexão e a desconexão com banco de dados.

Assim, optou-se pela utilização de PJP's para essas variabilidades de junção no sentido de facilitar a etapa de geração de produtos. Pode-se observar nas linhas 10 e 14 da Figura 7.15 a definição de dois PJP's relacionados com os PJA's "AbrirConexão" e "FecharConexão". Os PJP's foram configurados para permitir que a conexão com o banco seja realizada antes da execução do método principal do produto (`public static void main(..)`) e encerrada após o retorno desse mesmo método.

7.3.2 Inclusão dos Conjuntos de Extensão

Na Seção 6.5.3 foi mostrado um exemplo de gabarito do domínio FWS com um conjunto de extensão para o domínio de persistência. A utilização de CE's para a combinação de ambos os domínios se faz necessária, pois o código-fonte das classes do domínio FWS não realiza as chamadas apropriadas para os métodos de persistência de dados declarados na interface `PersistentRoot` e que são injetados nas classes persistentes por meio do aspecto `PersistentEntities`.

Alguns métodos de persistência, como o método `save()` ou `update()`, não precisam ser executados caso a variabilidade de consciência parcial tenha sido escolhida, visto que os dados são persistidos no momento em que o construtor ou os *setters* da classe persistente são acionados. Contudo, os métodos `delete()` e `find()` devem ser explicitamente invocados para realizar as operações de remoção e busca, respectivamente.

Assim, foram declarados CE's que injetam nos locais apropriados das classes do domínio FWS, as funções de manipulação de dados disponibilizadas pelo domínio de persistência. Vale lembrar que a lógica de inclusão de um CE deve levar em conta a estratégia de consciência escolhida (Parcial ou Total) e também se a classe afetada pelo CE foi apontada ou não como uma classe

persistente durante a etapa de engenharia de aplicações. A Figura 7.16 apresenta a criação de um CE dentro do gabarito da classe `DataBanker` do domínio FWS (linhas 16 à 20).

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0"
3                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4                  xmlns:captor="captor.lib.xalan.extensions.CaptorExtension"
5                  extension-element-prefixes="captor">
6
7      <xsl:output method="text"/>
8      <xsl:template match="/">
9
10     package fwslib;
11
12     public class DataBanker {
13         public static synchronized void write(SensorReading r)  {
14         ...
15
16         <captor:extensionset crosscuttingdomain="Persistence">
17           <xsl:if test="isPersistent('SensorReading') and $parcialObliv=false">
18             sr.save();
19           </xsl:if>
20         </captor:extensionset>
21             v.addElement(sr);
22         }
23         ...
24     }
25     </xsl:template>
26
27 </xsl:stylesheet>
```

Figura 7.16: Conjunto de extensão do domínio FWS

Ao término das atividades de configuração mostradas, o domínio FWS estará apto para ser combinado com o domínio de persistência, o que leva à etapa final desse estudo de caso, a geração de aplicações. Na próxima Seção são abordados os passos realizados para a geração de novos produtos formados a partir da combinação do domínio-base FWS com o domínio transversal de persistência.

7.4 Geração de Produtos

A etapa de geração de produtos é baseada no processo de engenharia de aplicações apresentado na Seção 5.3.2. Entretanto, nesse estudo de caso é dada maior ênfase para a utilização prática da ferramenta Captor-AO durante esse processo. Primeiramente, deve-se criar um novo projeto e escolher os domínios do produto: FWS (domínio-base) e persistência (domínio transversal). Na Figura 7.17 é apresentada a interface da ferramenta Captor-AO utilizada para a definição dos domínios do produto.

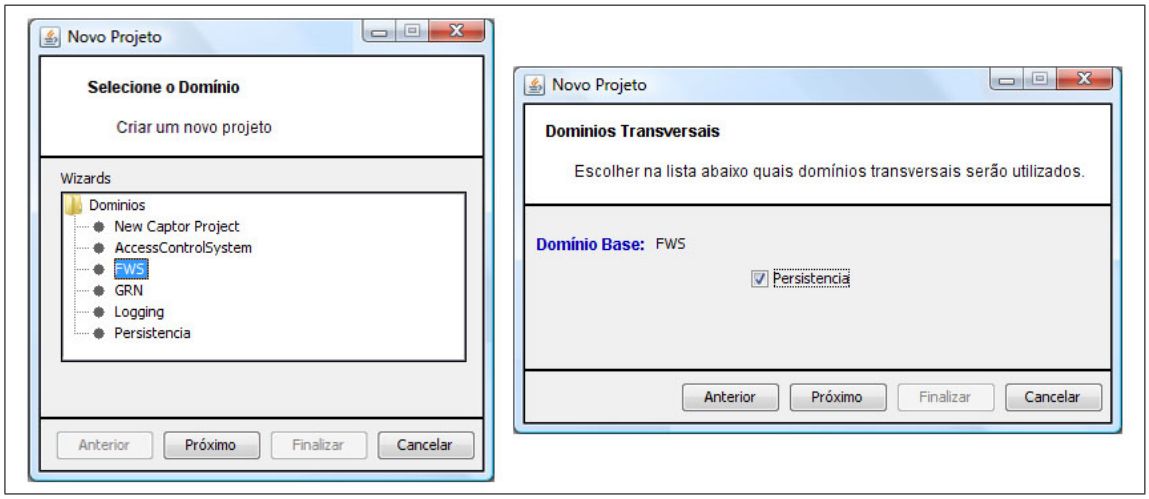

Figura 7.17: Definição dos domínios do novo produto

Também são definidos o nome do produto e o diretório de armazenamento dos artefatos gerados. Na sequência, a ferramenta é utilizada para instanciar as LMA's de ambos os domínios. Como exemplo, ilustra-se na Figura 7.18 o gerador sendo utilizado para especificar as variabilidades do novo produto.

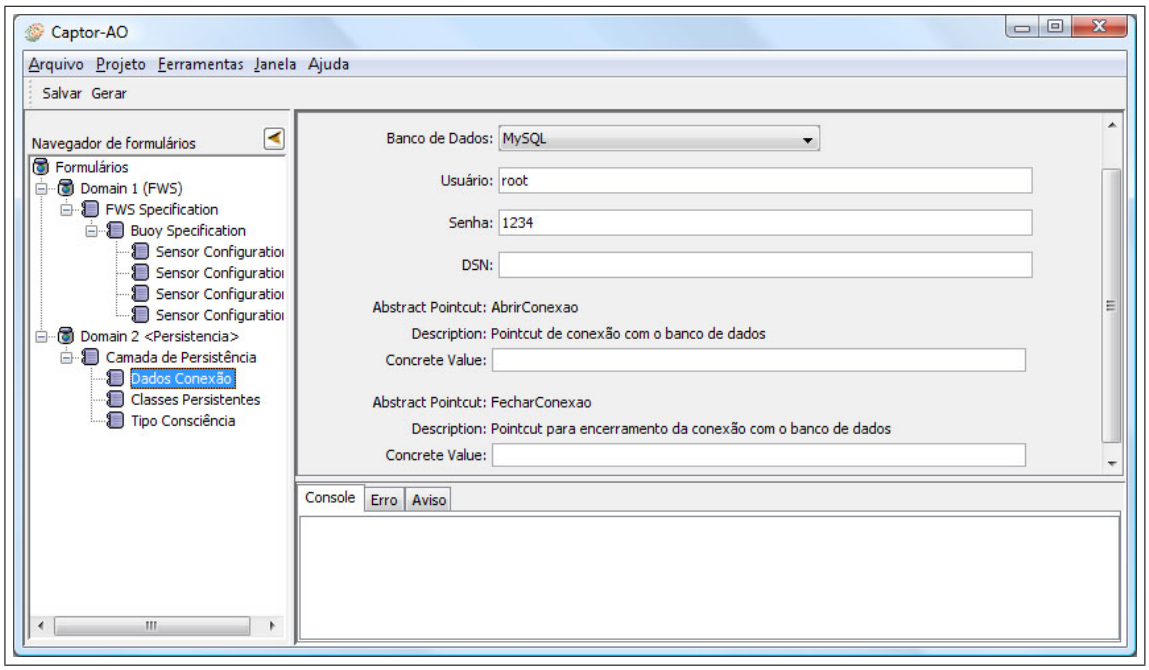

Figura 7.18: Interface utilizada durante a engenharia de aplicações do estudo de caso

Após a definição das variabilidades, o engenheiro de aplicações pode solicitar a geração do novo produto. Para disparar o processo de geração, deve-se utilizar o botão "Gerar" situado na barra de ferramentas da interface mostrada na Figura 7.18. É possível acompanhar o andamento do processo de geração no *console* mostrado na Figura 7.19.

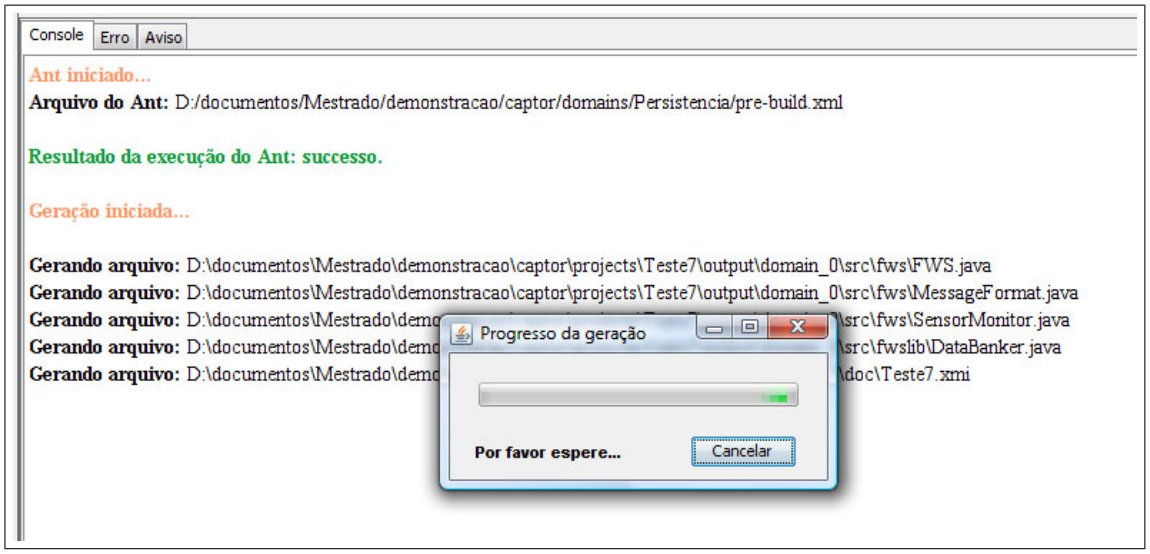

Figura 7.19: Progresso da geração

Terminado o processo de geração, todos os artefatos produzidos são copiados para o diretório de saída do projeto e armazenados em pacotes separados de acordo com o domínio original de cada artefato. Por fim, é realizada a compilação do código-fonte gerado. Para tal, foi utilizado nesse estudo de caso o ambiente de desenvolvimento Eclipse[1] para a compilação das classes Java e também para combinação dos aspectos utilizando o *plugin* AJDT[2], conforme mostrado na Figura 7.20.

[1]Eclipse Project: `http://www.eclipse.org/`.
[2]AspectJ Development Tools: `www.eclipse.org/ajdt/`.

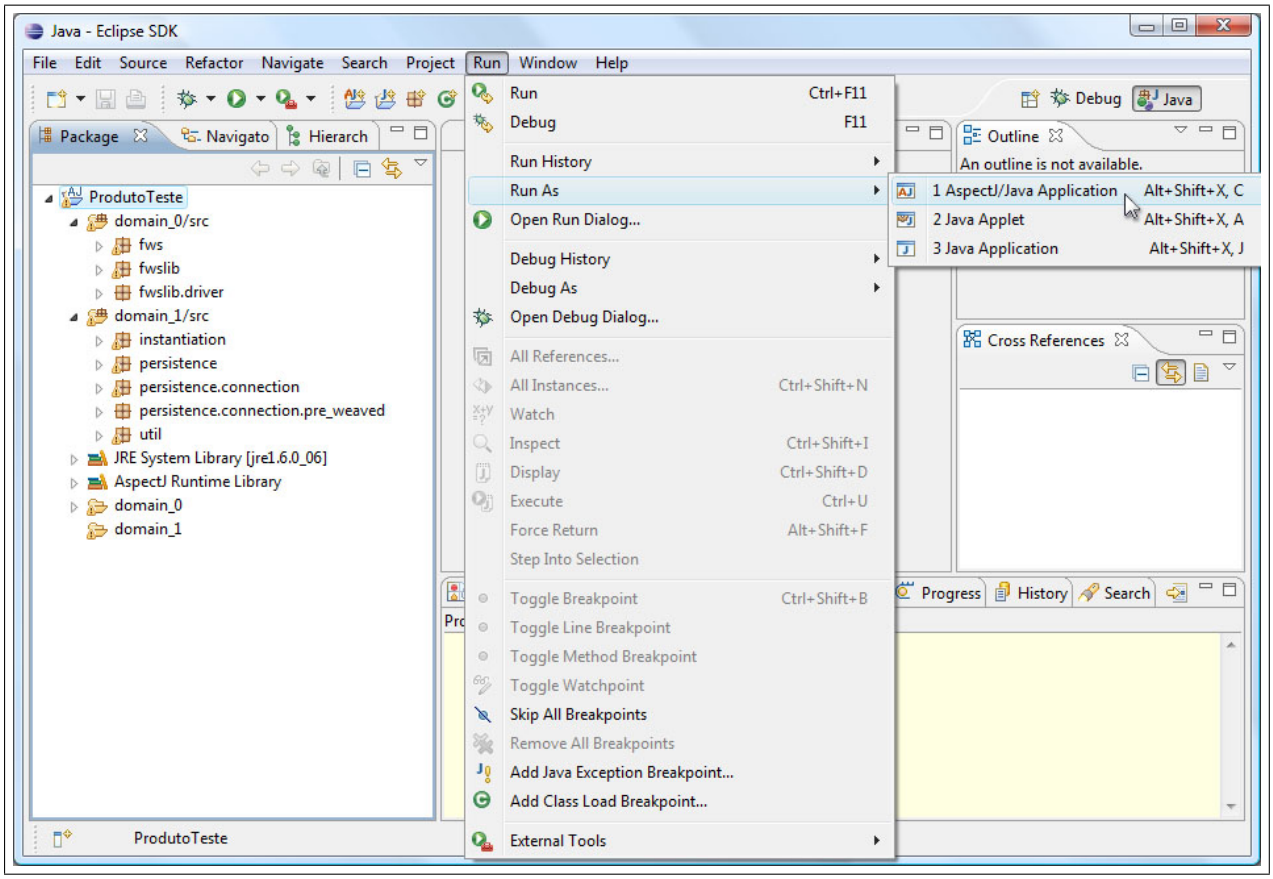

Figura 7.20: Compilação do produto utilizando o ambiente Eclipse

7.5 Considerações Finais

Neste Capítulo foi apresentado um estudo de caso realizado com o gerador Captor-AO. Esse estudo envolveu a configuração de um novo domínio transversal (persistência de dados) na ferramenta e a configuração de um domínio-base compatível (bóias náuticas). Também foi realizada com sucesso a geração de produtos a partir da combinação desses domínios. Os produtos gerados conseguem utilizar as funções dos dois domínios sem a ocorrência de conflitos. A realização desse estudo de caso permitiu uma análise prática dos processos de engenharia de domínios e engenharia de aplicações propostos nesta pesquisa. Além disso, esse estudo auxiliou nos testes das novas funcionalidades de configuração de domínios-base e domínios transversais desenvolvidas.

CAPÍTULO

8

Conclusão

8.1 Considerações Iniciais

Nesta dissertação foi apresentado um processo para o desenvolvimento de LPS e geração de produtos baseado na POA. Para tal, foram inicialmente mostrados os principais conceitos da engenharia de linha de produtos e da programação orientada a aspectos. Além desses conceitos, foram apresentadas pesquisas sobre o desenvolvimento de linhas de produtos com a utilização de aspectos. Observou-se a possibilidade de combinação entre as features transversais de determinados tipos de domínios durante a instanciação de produtos. Essa combinação de features é facilitada pela POA, que torna possível a inclusão de novas funcionalidades com pouca ou nenhuma modificação no código afetado, permitindo que tais features possam ser reusadas em um número maior de linhas de produtos.

O processo proposto permite a reutilização e combinação de domínios de aplicação de acordo com os interesses de cada domínio. Foram definidas novas atividades nas duas principais etapas da engenharia de linhas de produtos: engenharia de domínio e engenharia de aplicações. A nova abordagem de engenharia de domínio prevê uma distinção entre domínios formados por features específicas de uma área de negócio (domínios-base), dos domínios que encapsulam comportamentos genéricos referentes a um interesse transversal (domínios transversais). Também é prevista nesse método a definição de um ou mais domínios transversais compatíveis com um determinado domínio-base. Essa restrição de compatibilidade tem o objetivo de indicar quais domínios podem ser combinados entre si sem a ocorrência de erros.

O processo de engenharia de aplicações proposto possibilita a utilização de um domínio-base e vários domínios transversais para a criação de um novo produto. A escolha dos domínios trans-

versais é feita a partir do conjunto de domínios compatíveis definidos durante a engenharia de domínio. A instanciação dos novos produtos é feita por meio da especificação das variabilidades funcionais e de junção existentes nos domínios da combinação. No sentido de facilitar a etapa de engenharia de aplicações e tornar o processo menos suscetível a erros, foi desenvolvida uma extensão do gerador Captor, denominada Captor-AO. Essa extensão implementa todos os requisitos necessários para a configuração de domínios-base e domínios transversais, além de possibilitar a geração de produtos formados a partir da combinação de features provenientes de diferentes domínios.

Também foi mostrado neste trabalho um estudo de caso envolvendo a configuração de um domínio transversal e a definição de um domínio-base compatível utilizando o gerador Captor-AO. Para esse estudo de caso foram escolhidos o domínio transversal de persistência e o domínio-base de bóias náuticas. Após a configuração dos domínios na ferramenta, foi possível realizar a geração de produtos utilizando a combinação de features de ambos os domínios. Os produtos resultantes dessa combinação possuem as funcionalidades comuns do domínio de bóias náuticas juntamente com uma camada de persistência de dados.

8.2 Contribuições

A principal contribuição deste trabalho é a proposta de um processo de desenvolvimento de linhas de produtos apoiadas por geradores e aspectos que inclui: a separação entre variabilidades de junção e variabilidades funcionais, classificação de domínios de aplicação em domínios-base ou domínios transversais, definição de compatibilidade entre domínios e instanciação de produtos com reúso de features transversais de diferentes domínios.

Adicionalmente, foi implementada uma extensão do gerador Captor para apoiar a abordagem proposta. Essa extensão, chamada Captor-AO, permite a criação de linhas de produtos para domínios-base e domínios transversais, utilizando os novos conceitos propostos. A geração de produtos é feita a partir da composição entre features comuns de um domínio-base e features transversais de um ou mais domínios transversais compatíveis, previamente configurados na ferramenta. O gerador Captor-AO realiza a composição automática das features transversais no código-base, evitando que o engenheiro de aplicações tenha que realizar a instanciação manual dos pontos de junção nos aspectos utilizados, reduzindo tempo e custos da instanciação de produtos.

O gerador Captor-AO foi configurado para os domínios-base de clínicas de psicologia, Gestão de Recursos de Negócio (GRN) e bóias náuticas. O gerador também foi configurado para os domínios transversais de controle de acesso (compatível com o domínio GRN) e persistência de dados (compatível com o domínio de bóias náuticas). Uma vantagem da ferramenta Captor-AO é a possibilidade de geração de um número maior de produtos, graças à grande variedade de possíveis combinações entre domínios compatíveis. Também foram implementadas no gerador Captor-AO

diversas melhorias em relação à primeira versão da ferramenta, como a correção de defeitos (*bugs*), melhorias na interface e na usabilidade, e documentação.

8.3 Trabalhos Futuros

Como trabalhos futuros pretende-se realizar estudos de caso utilizando linhas de produtos reais com a finalidade de avaliar a eficiência da abordagem proposta em sistemas mais complexos. Vale ressaltar que os três estudos de caso realizados não são suficientes para garantir a corretude do processo em qualquer situação. Ainda é necessário realizar a validação formal, tanto do processo, quanto da ferramenta Captor-AO.

Um possível trabalho futuro seria permitir a utilização de dois ou mais domínios-base compatíveis no processo de combinação de domínios. Na versão atual do processo e da ferramenta, o engenheiro de aplicações deve escolher um domínio-base e, opcionalmente, um ou mais domínios transversais. Dessa forma, além do reúso dos domínios transversais, seria também maximizado o reúso de domínios-base. Por exemplo, é intuitivo enxergar a combinação de um domínio-base de contabilidade com domínios-base de vendas, hospitais, seguradoras, etc.

Uma outra possível extensão deste trabalho é criar mecanismos de validação dos pontos de entrecorte em que os domínios transversais afetam os domínios-base, analogamente ao que é proposto pelas XPI (Griswold et al., 2006) (Ver Seção 5.2.5). Na versão atual, o engenheiro de domínio especifica os PJA's e, se for o caso, alguns PJP's. Pode ser feito um estudo em que seriam definidas interfaces XPI para os domínios-base e os PJA's/PJP's estariam em conformidade com essas XPI's, garantindo assim, que só seriam interceptados pontos bem definidos nos domínios-base.

Atualmente, o processo de análise de compatibilidade entre domínios é feito de forma manual e baseado no conhecimento do engenheiro de domínio. Visto que essa é uma atividade difícil e extensa, pode-se investigar novos processos e/ou ferramentas que auxiliem nos testes de compatilidade e na identificação de possíveis conflitos.

Outro possível trabalho seria a criação de um *plugin* do Captor-AO para o ambiente de desenvolvimento Eclipse. Esse *plugin* teria a função de importar automaticamente o código-fonte gerado na forma de um novo projeto do ambiente Eclipse, facilitando a manutenção do código e aproveitando as funções existentes no ambiente para a compilação das classes e *weaving* dos aspectos.

Referências Bibliográficas

ANASTASOPOULOS, M.; GACEK, C. Implementing product line variabilities. *ACM Sigsoft Software Engineering Notes*, v. 23, n. 3, p. 109–117, 2001.

ANASTASOPOULOS, M.; MUTHIG, D. An evaluation of aspect-oriented programming as a product line implementation technology. In: *ICSR '04: Proceedings of the 8th International Conference Software Reuse*, Springer, 2004, p. 141–156.

APEL, S.; BATORY, D. When to use features and aspects?: a case study. In: *GPCE '06: Proceedings of the 5th international conference on Generative programming and component engineering*, ACM Press, 2006, p. 59–68.

APEL, S.; LEICH, T.; ROSENMULLER, M.; SAAKE, G. Combining feature-oriented and aspect-oriented programming to support software evolution. In: *RAM-SE '05: Proceedings of 2nd ECOOP Workshop on Reflection, AOP and Meta-Data for Software Evolution*, 2005a, p. 3–16.

APEL, S.; LEICH, T.; ROSENMULLER, M.; SAAKE, G. FeatureC++: On the symbiosis of Feature-Oriented and Aspect-Oriented programming. In: *GPCE '05: Proceedings of the 4th International Conference on Generative Programming and Component Engineering*, Springer, 2005b, p. 125–140.

APEL, S.; LEICH, T.; SAAKE, G. Aspect refinement and bounding quantification in incremental designs. In: *APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)*, IEEE Computer Society, 2005c, p. 796–804.

BACHMANN, F.; GOEDICKE, M.; LEITE, J.; NORD, R.; POHL, K.; RAMESH, B.; VILBIG, A. A meta-model for representing variability in product family development. In: *PFE '03: Proceedings of the 5th International Workshop on Software Product-Family Engineering*, Springer, 2003, p. 66–80.

BATORY, D.; CARDONE, R.; SMARAGDAKIS, Y. Object-oriented framework and product lines. In: *Proceedings of the First Software Product Line Conference*, Kluwer Academic Publishers, 2000, p. 227–247.

BATORY, D.; SARVELA, J. N.; RAUSCHMAYER, A. Scaling step-wise refinement. In: *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, IEEE Computer Society, 2003, p. 187–197.

BECKER, M.; KAISERSLAUTERN, G. Towards a general model of variability in product families. In: *Proceedings of the 1st Workshop of Software Variability Management at ICSE 2003*, 2003, p. 9.

BEUCHE, D. *Composition and construction of embedded software families*. Tese de doutorado, Ottovon-Guericke-Universität, Magdeburg, 2003.

BIGGERSTAFF, T. J.; PERLIS, A. J. *Software reusability*. ACM Press, 1989.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The unified modeling language user guide*. Addison-Wesley, 2000.

BOSCH, J.; FLORIJN, G.; GREEFHORST, D.; KUUSELA, J.; OBBINK, J.; POHL, K. Variability issues in software product lines. 2002.

BOSCH, J.; MOLIN, P.; MATTSSON, M.; BENGTSSON, P.; FAYAD, M. *Framework problems and experiences*, v. 1 de *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. New York, NY, USA: John Willey and Sons, 55–82 p., 1999.

BRACHA, G.; COOK, W. Mixin-based inheritance. In: *International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) and European Conference on Object-Oriented Programming (ECOOP)*, 1990, p. 303–311.

BRAGA, R. T. V. *Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico*. Tese de doutorado, ICMC/USP, São Carlos, SP, 2003.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. A pattern language for business resource management. In: *In Proceedings of the 6th Annual Conference on Pattern Languages of Programs*, 1999a, p. 1–33.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. A pattern language for business resource management. In: *Proceedings of the 6th Pattern Languages of Programs Conference (PLoP'99)*, 1999b, p. 1–34.

BRAGA, R. T. V.; MASIERO, P. C. Gren-wizard: a tool to instantiate the gren framework. In: *Caderno de Ferramentas do XVI Simpósio Brasileiro de Engenharia de Software (SBES'2002)*, 2002a, p. 408–413.

BRAGA, R. T. V.; MASIERO, P. C. A process for framework construction based on a pattern language. In: *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC), IEEE Computer Society*, 2002b, p. 615–620.

BRUGALI, D.; MENGA, G.; AARSTEN, A. *A case study for flexible manufacuring systems*. Domain-Specific Application Frameworks: Frameworks Experience by Industry. John Willey and Sons, 85–99 p., 2000.

BUSCHMANN, F. *Pattern-oriented software architecture - a system of patterns*. Wiley, 1996.

CAMARGO, V. V. *Frameworks transversais: definições, classificações, arquitetura e utilização em um processo de desenvolvimento de software*. Tese de doutorado, ICMC-USP, São Carlos, SP, 2006.

CAMARGO, V. V.; RAMOS, R. A.; PENTEADO, R. A. D.; MASIERO, P. C. Projeto orientado a aspectos do padrão camada de persistência. In: *Anais do XVII Simpósio Brasileiro de Engenharia de Software (SBES'2003)*, 2003, p. 114–129.

CHAVEZ, C. *A model-driven approach to aspect-oriented design*. Tese de doutorado, PUC-Rio, Rio de Janeiro, Brasil, 2004.

CIRILO, E.; KULESZA, U.; LUCENA, C. Genarch: a model-based product derivation tool. In: *Proceedings of Simpósio Brasileiro de Componentes, Arquitetura e Reutilização de Software (SBCARS'2007)*, 2007, p. 31–46.

CLEAVELAND, J. C. Building application generators. *IEEE Software*, v. 9, n. 4, p. 25–33, 1988.

CLEAVELAND, J. C. *Program generators with xml and java*. Prentice Hall, 2001.

CLEMENTS, P.; NORTHROP, L. *Software product lines: Practices and patterns*. Addison-Wesley, 576 p., 2001.

CLIFTON, C.; LEAVENS, G. T. *Obliviousness, modular reasoning, and the behavioral subtyping analogy*. Relatório Técnico TR03-01a, Iowa State University, 2003.

COPLIEN, J. O. *Software design patterns: Common questions and answers*. The Patterns Handbook: Techniques, Strategies, and Applications. Cambridge University Press, 311–320 p., 1998.

CZARNECKI, K.; EISENERCKER, U. W. *Generative programming*. Addison-Wesley, 2002.

DIAZ, R. P. Domain analysis: An introduction. *ACM SIGSOFT Software Engineering Notes*, v. 15, p. 47–54, 1990.

DIJKSTRA, E. *A discipline of programming*. Prentice-Hall, 1976.

ELER, M. M. *Um método para o desenvolvimento de software baseado em componentes e aspectos*. Dissertação de Mestrado, ICMC/USP, São Carlos, SP, 2006.

ELRAD, T.; FILMAN, R. E.; BADER, A. Aspect oriented programming. *Communications of the ACM*, v. 44, n. 10, p. 29–32, 2001a.

ELRAD, T.; KICZALES, G.; AKSIT, M.; LIEBERHER, K.; OSSHER, H. Discussing aspects of aop. *Communications of the ACM*, v. 44, n. 10, p. 33–38, 2001b.

FAYAD, M. E.; JOHNSON, R.; SCHMIDT, D. C. *Building application frameworks: Object oriented foundations of framework design*. John Wiley and Sons, 1999.

FILMAN, R. E.; FRIEDMAN, D. P. Aspect-oriented programming is quantification and obliviousness. In: *OOPSLA '00: Position paper for the Advanced Separation of Concerns Workshop at the Conference on Object-Oriented Programming Systems, Languages, and Applications*, ACM, 2000.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, 1995.

GOMAA, H. *Designing software product lines with uml: From use cases to pattern-based software architectures*. Addison-Wesley, 2004.

GREENFIELD, J.; SHORT, K. *Software factories - assembling applications with patterns, models, frameworks, and tools*. Wiley Publishing, Inc, 2004.

GRISS, M. L. Implementing product-line features with component reuse. In: *6th International Conference on Software Reuse (ICSR)*, 2000, p. 137–152.

GRISS, M. L. *Product-line architectures*, cap. 22. Component-Based Software Engineering Addison-Wesley, p. 405–420, 2001.

GRISWOLD, W. G.; SULLIVAN, K.; SONG, Y.; SHONLE, M.; TEWARI, N.; CAI, Y.; RAJAN, H. Modular software design with crosscutting interfaces. *IEEE Software*, v. 23, n. 1, p. 51–60, 2006.

GROHER, I.; BAUMGARTH, T. Aspect-orientation from design to code. In: *Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, 2004, p. 62–68.

HARSU, M. *Fast product-line architecture process*. Ed. Tampere University of Technology, 2002.

HEO, S.; CHOI, E. M. Representation of variability in software product line using aspect-oriented programming. In: *SERA '06: Proceedings of the 4th International Conference on Software Engineering Research, Management and Applications*, IEEE Computer Society, 2006, p. 66–73.

JACOBSON, I.; GRISS, M.; JONSSON, P. *Software reuse: Architecture, process and organization for business success*. Addison-Wesley, 1997.

JAKARTA Apache software foundation. Disponível em: `http://jakarta.apache.org`. Acessado em 05/08/2008, 2008.

JAVA Java 2 standard edition. Disponível em: `http://java.sun.com`. Acessado em 24/09/2008, 2008.

JOHNSON, R. E. Frameworks = (components + patterns). *Communications of the ACM*, v. 40, n. 10, p. 39–42, 1997.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. *Journal of Object Oriented Programming*, v. 1, n. 2, p. 22–35, 1988.

KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. *Feature-oriented domain analysis (foda) feasibility study*. Relatório Técnico CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

KANG, K. C.; KIM, S.; LEE, J.; KIM, K.; SHIN, E.; HUH, M. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, v. 5, p. 143–168, 1998.

KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J.; GRISWOLD, W. G. An overview of aspectj. *Lecture Notes in Computer Science*, v. 2072, p. 327–355, 2001.

KICZALES, G.; LAMPING, J.; MENHDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J. M.; IRWIN, J. Aspect-oriented programming. In: *ECOOP '97: Proceedings of European Conference on Object-Oriented Programming*, Springer-Verlag, 1997, p. 220–242.

KRUEGER, C. Software reuse. *ACM Computing Surveys*, v. 24, n. 2, p. 131–183, 1992.

KULESZA, U.; ALVES, V.; GARCIA, A.; LUCENA, C.; BORBA, P. Improving extensibility of object-oriented frameworks with aspect-oriented programming. In: *ICSR '06: Proceedings of the 9th International Conference on Software Reuse*, Springer Verlag, 2006a, p. 231–245.

KULESZA, U.; ALVES, V.; GARCIA, A.; NETO, A.; CIRILLO, E.; LUCENA, C.; BORBA, P. Mapping features to aspects: A model-based generative approach. In: *AOSD '07: Proceedings of 10th International Workshop on Early Aspects*, 2007.

KULESZA, U.; COELHO, R.; ALVES, V.; GARCIA, A.; STAA, A.; LUCENA, C.; BORBA, P. Implementing framework crosscutting extensions with ejps and aspectj. In: *Proceedings of ACM SIGSoft on XX Brazilian Symposium on Software Engineering*, 2006b.

KULESZA, U.; GARCIA, A.; BLEASBY, F.; LUCENA, C. Instantiating and customizing product line architectures using aspects and crosscutting feature models. In: *OOPSLA '05: Proceedings of the Workshop on EarlyAspects*, 2005.

LAJOIE, R.; KELLER, R. Design and reuse in object-oriented frameworks: Patterns, contracts, and motifs in concert. In: *62nd Congress of the Association Canadienne Française pour lavancement des Sciences (ACFAS)*, 1994.

LEE, K.; KANG, K. C.; KIM, M.; PARK, S. Combining feature-oriented analysis and aspect-oriented programming for product line asset development. In: *SPLC '06: Proceedings of the 10th International on Software Product Line Conference*, IEEE Computer Society, 2006, p. 103–112.

LESAINT, D.; PAPAMARGARITIS, G. Aspects and constraints for implementing configurable product-line architectures. In: *WICSA '04: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture*, IEEE Computer Society, 2004, p. 135–144.

LOHMANN, D.; SCHRÖDER-PREIKSCHAT, W.; SPINCZYK, O. *The design of application-tailorable operating system product lines*. Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. Springer, 99–117 p., 2006.

MASIERO, P. C.; MEIRA, C. A. A. Development and instantiation of a generic application generator. *Journal of Systems and Software*, v. 23, n. 1, p. 27–37, 1993.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM Computing Surveys*, v. 37, n. 4, p. 316–344, 2005.

MEZINI, M.; OSTERMANN, K. Conquering aspects with caesar. In: *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, ACM Press, 2003, p. 90–99.

MEZINI, M.; OSTERMANN, K. Variability management with feature-oriented programming and aspects. *SIGSOFT Softw. Eng. Notes*, v. 29, n. 6, p. 127–136, 2004.

NEIGHBORS, J. M. The draco approach to constructing software from reusable components. *IEEE Transactions on Software Engineering*, v. SE-10, n. 5, p. 564–574, 1984.

PACIOS, S. F. *Uma abordagem orientada a aspectos para desenvolvimento de linhas de produtos de software*. Dissertação de Mestrado, ICMC/USP, São Carlos, SP, 2006.

PARNAS, D. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, v. 5, n. 2, p. 128–138, 1979.

PEREIRA, C. A. F.; BRAGA, R. T. V. Composição e geração de aplicações usando aspectos. In: *Anais do XII Workshop de Teses e Dissertações em Engenharia de Software (WTES'2007)*, 2007, p. 33–38.

PEREIRA, C. A. F.; BRAGA, R. T. V.; MASIEIRO, P. C. Captor-ao: Gerador de aplicações apoiado pela programação orientada a aspectos. In: *Anais da Seção de Ferramentas do II Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS'2008)*, 2008, p. 1–8.

PRESSMAN, R. S. *Engenharia de software*. 5nd. ed. McGraw-Hill, 2002.

SEI A framework for software product line practice. Versão 4.2. Software Engineering Institute, Carnegie Mellon University. Disponível em: `http://www.sei.cmu.edu/productlines/index.html`. Acessado em 27/09/2008, 2008.

SHIMABUKURO, E. K. *Um gerador de aplicações configurável*. Dissertação de Mestrado, ICMC/USP, São Carlos, SP, 2006.

SHIMABUKURO, E. K.; MASIERO, P. C.; BRAGA, R. T. V. Captor: Um gerador de aplicações configurável. Sessão de ferramentas do XX Simpósio Brasileiro de Engenharia de Software (SBES'2006), 2006.

SILVA, M. T. F. *Uso de aspectos para apoiar a evolução não funcional de frameworks: Aplicação ao framework GREN*. Dissertação de Mestrado, ICMC/USP, São Carlos - SP, 2004.

SILVA, M. T. F.; BRAGA, R. T. V.; MASIERO, P. C. Evolução orientada a aspectos de um framework oo. Versão estendida do artigo publicado no WMSWM 2004, Revista Tecnologia da Informação - RT-Info, Edição Temática em: Novas Abordagens para Manutenção de Software, to appear., 2006.

SMARAGDAKIS, Y. *Implementing large-scale object-oriented components*. Tese de doutorado, Department of Computer Sciences, University of Texas, 1999.

SMARAGDAKIS, Y.; BATORY, D. Implementing layered designs with mixin layers. In: *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Springer-Verlag LNCS 1445, 1998, p. 550–570.

SMARAGDAKIS, Y.; BATORY, D. *Application generators. encyclopedia of electrical and electronics engineering*. John Wiley and Sons, 2000.

SMARAGDAKIS, Y.; BATORY, D. Mixin layers: An object-oriented implementation technique for refinements and collaboration designs. *ACM TSEM*, v. 11, n. 2, p. 215–255, 2002.

TALIGENT, I. Leveraging object-oriented frameworks. White-paper, Disponível em `http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/leveragingoo.pdf`. Acessado em 27/09/2008, 2008.

WEISS, D. M.; LAI, C. T. R. *Software product-line engineering: a family-based software development process*. Addison-Wesley, 1999.

XSL The extensible stylesheet language family (xsl). Disponível em: `http://www.w3.org/Style/XSL/`. Acessado em 05/08/2008, 2008.

YASSIN, A.; FAYAD, M. E. *Application frameworks: A survey*, cap. 29. Domain-Specific Application Frameworks: Frameworks Experience by Industry John Willey and Sons, p. 615–632, 2000.

APÊNDICE

A

Manual de Instalação do Gerador Captor-AO

A.1 Executando o Arquivo Binário

A ferramenta Captor-AO está hospedada no seguinte *website*: `http://captor.googlecode.com`. Ao entrar na url indicada, basta clicar na aba *downloads*, onde podem ser encontradas as diferentes versões do gerador.

Atualmente, a ferramenta se encontra na versão 2.0.6 e está disponível tanto para sistema operacional Windows quanto Linux. Neste manual, será adotado como referência o sistema operacional Windows.

Primeiramente, deve-se efetuar o *download*, salvando o arquivo que contém a ferramenta em uma pasta conhecida. Em seguida, é necessário descompactar o arquivo salvo. Por fim, basta acessar o diretório descompactado chamado "captor" e executar o aplicativo `captor-ao_pt.exe` Neste momento, a ferramenta já está pronta para ser utilizada, como mostra a Figura A.1.

A.2 Obtendo o Código-Fonte

O código da ferramenta é aberto e está disponível para ser utilizado por outros pesquisadores ou mesmo para customização pelos próprios usuários.

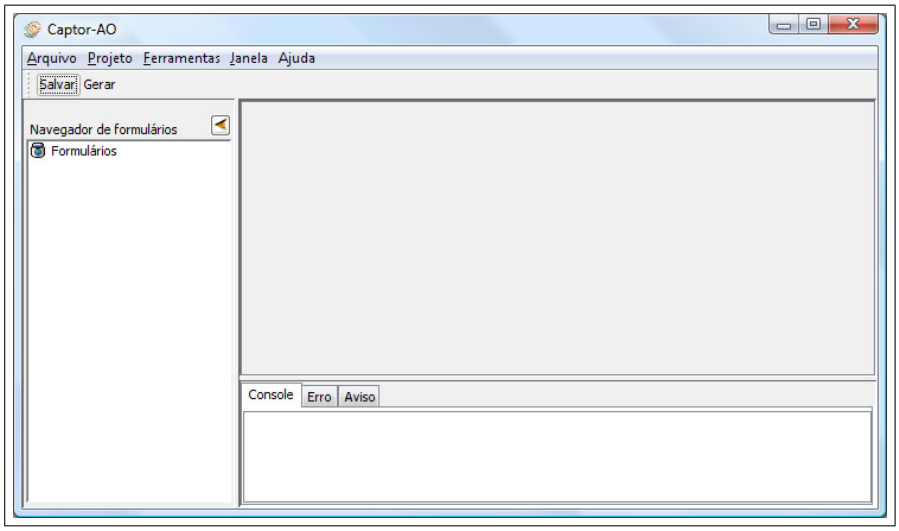

Figura A.1: Interface do gerador Captor-AO

Neste trabalho, foi utilizada a ferramenta para gerenciamento e automação de projetos Maven. O Maven contém tarefas pré-definidas que realizam funções bem conhecidas como compilação e empacotamento de código.

Primeiramente, deve-se instalar e configurar o Maven, que está disponível em `http://maven.apache.org`. Em seguida, é necessário efetuar o *checkout* do código do gerador Captor-AO armazenado em `http://code.google.com/p/captor/source/checkout`, utlizando um sistema de controle de versão.

Finalmente, é utilizado o comando "`mvn install`" para compilar e instalar as bibliotecas do gerador Captor-AO. A ferramenta Maven irá efetuar o cópia das dependências necessárias de forma automática.

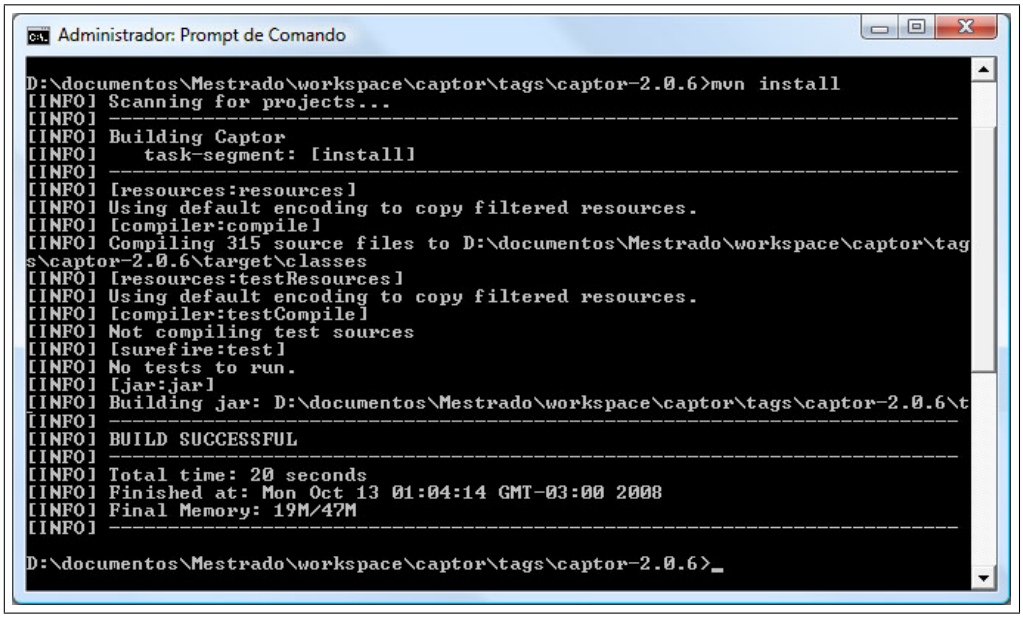

Figura A.2: Compilando o gerador Captor-AO

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo