Um Gerador de aplicações configurável

Edison Kicho Shimabukuro Junior

Um Gerador de aplicações configurável

Edison Kicho Shimabukuro Junior

Orientador: ***Prof. Dr. Paulo Cesar Masiero***

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
Abril/2006

Agradecimentos

Ao longo deste período de estudos e pesquisas tenho muito a agradecer às pessoas que me ajudaram diretamente ou indiretamente na realização deste trabalho. Das pessoas que me ajudaram diretamente agradeço:

Ao meu pai e a minha mãe, por me incentivarem a entrar no mestrado e me apoiarem nos momentos em que foi preciso.

Ao meu orientador, Prof. Dr. Paulo Cesar Masiero, pela valiosa orientação, pelo direcionamento do trabalho, contribuições, acompanhamento e validação do trabalho efetuado.

A Profa. Dra Rosana T. V. Braga, por ter realizado a pesquisa que originou este trabalho, sem a qual ele nunca existiria. Pelo acompanhamento do projeto de pesquisa, pelas reuniões, correções, sugestões, dicas e atenção.

Ao Kiko, pela contribuição realizada após a qualificação.

A Carina, pela ajuda nas correções finais desta dissertação.

Aos professores e funcionários do ICMC, pela disposição e atenção.

À FAPESP pelo apoio financeiro.

Das pessoas que me ajudaram indiretamente agradeço:

A Carina, pelo companheirismo, amizade e amor.

Aos meus pais, pela formação, educação e amor.

Ao Carlos, pela convivência, longas conversas, amizade e viagens realizadas.

Ao pessoal da lista do futebol do PGCOMPUSP-2004, pelas inúmeras partidas de futebol, bem estar e divertimento proporcionado.

Aos amigos, Clóvis, Giordano e Hecht pela boa compahia, pela amizade e pelos bons momentos.

A todos os colegas do LabES, em especial, ao Fabiano, Stanley, MarEler, Antonielly, Paula, André, Érika e Sandro.

Ao pessoal do PGCOMPUSP-2004 como um todo, pela animação e pelas boas festas.

Obrigado a todos!

Resumo

Os geradores de aplicação são ferramentas que recebem uma especificação de software, validam essa especificação e geram artefatos automaticamente. Os geradores de aplicação podem trazer benefícios em termos de produtividade por gerarem automaticamente artefatos de baixo nível com base em especificações de nível mais alto. Um dos problemas dos geradores de aplicação é o seu alto custo de desenvolvimento. Os geradores de aplicação configuráveis são adaptados para fornecer apoio em domínios específicos, ou seja, são considerados meta-geradores utilizados para obter geradores de aplicação específicos. Este trabalho delineia um processo de desenvolvimento com geradores configuráveis, define a arquitetura e as características de um gerador configurável e apresenta a ferramenta Captor, que é um gerador de aplicação configurável desenvolvido para facilitar a construção de geradores específicos. Três estudos de caso nos quais a Captor é configurada para domínios de aplicação específicos são apresentados: persistência de dados, gestão de recursos de negócios e bóias náuticas.

Abstract

Application generators are tools that receive as input a software specification, validate it and automatically generate artifacts based on it. Application generators can bring several benefits in terms of productivity, as they automatically generate low-level artifacts based on higher abstraction level specifications. A major concern of application generators is their high development cost. Configurable application generators are those generators that can be adapted to give support in specific domains, i.e., they are considered as meta-generators through which it is possible to obtain specific application generators. This work presents an approach for software development supported by configurable application generators. It defines the architecture and main features of a configurable application generator and presents Captor, which is a configurable application generator developed to ease the creation of specific generators. Three case studies were conducted to show the configuration of the Captor tool to different application domains: objects persistence, business resource management and floating weather stations.

Sumário

1 Introdução **1**
- 1.1 Contextualização . . . 1
- 1.2 Motivação . . . 2
- 1.3 Objetivos . . . 2
- 1.4 Organização . . . 3

2 Reúso de software **5**
- 2.1 Considerações iniciais . . . 5
- 2.2 Padrões de Software . . . 6
  - 2.2.1 Padrões de projeto . . . 7
  - 2.2.2 Padrões arquiteturais . . . 7
  - 2.2.3 Linguagem de padrões . . . 9
    - 2.2.3.1 Linguagem de padrões de Gestão de Recusos de Negócio (GRN) . . . 9
- 2.3 Frameworks de Software Orientados a Objetos . . . 13
  - 2.3.1 Exemplos de frameworks relevantes . . . 15
- 2.4 Geração de código . . . 18
  - 2.4.1 Geradores de aplicação . . . 19
  - 2.4.2 Alteração do software gerado . . . 21
  - 2.4.3 Exemplos de geradores . . . 24
- 2.5 Engenharia de software de linhas de produtos . . . 25
  - 2.5.1 Processo de engenharia de software de linhas de produtos . . . 26
    - 2.5.1.1 Processo FAST . . . 28
- 2.6 Considerações finais . . . 30

3 Geradores de aplicação configuráveis **33**
- 3.1 Considerações iniciais . . . 33
- 3.2 Visão geral dos geradores de aplicação configuráveis . . . 34
- 3.3 Visão geral da etapa de desenvolvimento . . . 37
- 3.4 Engenharia de domínio . . . 39
  - 3.4.1 Análise de domínio . . . 39
    - 3.4.1.1 Descrição do domínio . . . 40
    - 3.4.1.2 Definição da terminologia comum . . . 40

3.4.1.3 Análise de aspectos similares e variáveis 41
3.4.1.4 Criação da linguagem de modelagem de aplicações 41
3.4.2 Projeto de domínio 43
3.4.3 Implementação de domínio 44
3.4.4 Configuração da ferramenta 45
3.4.4.1 Desenvolvimento de gabaritos 46
3.4.4.2 Mapeamento da LMA nos gabaritos 47
3.4.5 Documentação do domínio 50
3.5 Engenharia da aplicação 50
3.5.1 Engenharia de requisitos 50
3.5.2 Criação do modelo da aplicação 50
3.5.3 Geração de artefatos 51
3.5.4 Implementação das funcionalidades não cobertas pelo gerador 54
3.5.5 Testes e Manutenção 54
3.5.5.1 Manutenção no modelo da aplicação 54
3.5.5.2 Manutenção nos artefatos implementados manualmente . . . 55
3.5.5.3 Manutenção manual nos artefatos gerados 55
3.5.5.4 Manutenção na configuração de domínio 55
3.6 Considerações finais 55

4 O Gerador de aplicação configurável Captor **57**
4.1 Considerações iniciais 57
4.2 Etapa de desenvolvimento da Captor 58
4.2.1 Análise 58
4.2.1.1 Visão geral da Captor 58
4.2.2 Projeto 60
4.2.2.1 Gerenciamento de projeto 60
4.2.2.2 Gerenciamento de domínio 61
4.2.2.3 Gerenciamento de interface GUI 62
4.2.2.4 Transformação de gabaritos 62
4.2.3 Implementação e Testes 63
4.3 Engenharia de domínio 63
4.3.1 Análise de domínio 64
4.3.1.1 Descrição do domínio 64
4.3.1.2 Definição da terminologia comum 64
4.3.1.3 Análise dos aspectos similares e variáveis 64
4.3.1.4 Criação da linguagem de modelagem de aplicações 64
4.3.2 Projeto de domínio 69
4.3.3 Implementação de domínio 69
4.3.4 Configuração da ferramenta 69
4.3.4.1 Configuração da interface e das regras de validação da especificação 72
4.3.4.2 Criação dos gabaritos 80
4.3.4.3 Criação do arquivo de mapeamento de transformação de gabaritos 83
4.3.4.4 Criação dos arquivos de pré e pós-processamento 87

4.3.5 Documentação de domínio . . . 88
4.4 Engenharia de aplicação . . . 88
4.5 Discussão . . . 88
4.6 Considerações finais . . . 89

5 Estudos de caso **91**
5.1 Considerações iniciais . . . 91
5.2 Linha de produtos de bóias náuticas - FWS . . . 92
5.2.1 Engenharia de domínio . . . 92
5.2.1.1 Análise de domínio . . . 92
5.2.1.2 Projeto de domínio . . . 97
5.2.1.3 Implementação do domínio FWS . . . 99
5.2.1.4 Configuração da Captor . . . 99
5.2.1.5 Documentação de domínio . . . 104
5.2.2 Engenharia de aplicação . . . 104
5.2.2.1 Engenharia de requisitos . . . 104
5.2.2.2 Criação do modelo da aplicação . . . 104
5.2.2.3 Geração de artefatos . . . 106
5.2.2.4 Implementação de funcionalidades não cobertas pelo gerador 107
5.2.2.5 Testes e manutenção . . . 107
5.3 Gestão de recursos de negócios . . . 108
5.3.1 Engenharia de domínio . . . 108
5.3.1.1 Análise de domínio . . . 108
5.3.1.2 Projeto de domínio . . . 111
5.3.1.3 Implementação de domínio . . . 112
5.3.1.4 Configuração da ferramenta . . . 112
5.3.1.5 Documentação de domínio . . . 113
5.3.2 Engenharia de aplicação . . . 114
5.3.2.1 Engenharia de requisitos . . . 114
5.3.2.2 Criação do modelo da aplicação . . . 114
5.3.2.3 Geração de artefatos . . . 114
5.3.2.4 Implementação das funcionalidades não cobertas pelo gerador 118
5.3.2.5 Testes e manutenção . . . 118
5.4 Comparação de resultados . . . 118
5.5 Considerações finais . . . 121

6 Conclusões **123**
6.1 Considerações finais . . . 123
6.2 Contribuições . . . 123
6.3 Limitações do trabalho efetuado . . . 124
6.4 Sugestões para trabalhos futuros . . . 124

Referências **132**

A Manual do simulador das bóias FWS **133**

Lista de Figuras

2.1 Padrão de projeto método-gabarito (Gamma et al., 1995) 8
2.2 Relacionamento entre os padrões da linguagem de padrões GRN (Braga, 2003) 11
2.3 Diagrama de classes para o padrão IDENTIFICAR O RECURSO 12
2.4 Diagrama de classes de um exemplo para o padrão IDENTIFICAR O RECURSO 13
2.5 Arquitetura do GREN 16
2.6 Exemplo de algumas classes do GREN 18
2.7 Transformações vertical, horizontal e oblíqua (Czarnecki e Eisenercker, 2002) . 21
2.8 Manutenção nos artefatos gerados 21
2.9 Métodos delegate 22
2.10 Geração de artefatos com zonas de segurança 23
2.11 Gerador baseado em XML e gabaritos 24
2.12 Processo de desenvolvimento de linhas de produtos (Weiss e Lai, 1999) 27

3.1 Exemplo do ciclo de vida de um gerador configurável 35
3.2 Avaliação do gerador configurável 36
3.3 Representação da LMA por meio de formulários 42
3.4 LMA no formato de um diagrama UML 42
3.5 LMA no formato do XML 43
3.6 Artefatos do domínio de persistência 44
3.7 Arquitetura do domínio de persistência de dados 45
3.8 Exemplo simplificado de transformação de gabaritos 46
3.9 Exemplo de gabarito inserido no código do gerador de aplicação 47
3.10 Mapeamento de variabilidades 48

4.1 Arquitetura da Captor 60
4.2 Módulo de gerenciamento de projetos 61
4.3 Classes que implementam o módulo de gerenciamento de projetos 61
4.4 Módulo de gerenciamento de domínio 62
4.5 Módulo de transformação de gabaritos 63
4.6 Representação gráfica de um formulário exemplo 65
4.7 Árvore com três formulários 65
4.8 Árvore com três formulários e duas variantes para o formulário filho 1 65
4.9 Exemplo de árvore com três formulários 66

4.10 Formulários do domínio de persistência 67
4.11 Representação gráfica de um formulário com quatro elementos 68
4.12 Primeira etapa do fluxo de execução da Captor 70
4.13 Segunda etapa do fluxo de execução da Captor 70
4.14 Seleção de gabaritos durante a geração de artefatos 70
4.15 Implantação da configuração no gerador configurável 71
4.16 Criação de um novo projeto 73
4.17 Edição da especificação 74
4.18 Inserção de novos formulários 74
4.19 Novos formulários 75
4.20 Edição de um novo formulário 75
4.21 Edição dos formulário filhos 76
4.22 Edição de variantes 77
4.23 Adição dos elementos de formulário nas variantes 77
4.24 Geração do arquivo de configuração 78
4.25 Detecção de erros na especificação 79
4.26 Formulários do domínio de persistência 83

5.1 Árvore de formulários do domínio FWS 95
5.2 Ambiente de simulação das bóias FWS 97
5.3 Diagrama de classes do domínio FWS 98
5.4 Estrutura de diretório dos arquivos de uma aplicação FWS 103
5.5 Captor sendo utilizada para armazenar o modelo de uma aplicação FWS 105
5.6 Formulário de especificação de bóias 105
5.7 Formulário de especificação de sensores 105
5.8 Árvore de formulários da LMA da GRN 109
5.9 Captor sendo utilizada para armazenar o modelo de uma aplicação no domínio GRN 115
5.10 Formulário do padrão: Identificar o recurso 115
5.11 Formulário do padrão: Aluguel do recurso 115
5.12 Aplicação orientada a objetos 120
5.13 Instanciação de frameworks caixa-branca 120

A.1 Execução da aplicação FWS e dos simuladores. 134

Lista de Tabelas

1 Lista de Siglas xiii

3.1 Terminologia comum do domínio de persistência 40
3.2 Aspectos similares 41
3.3 Variabilidades identificadas 41
3.4 Representação tabular do mapeamento das variabilidades 48

4.1 Representação tabular do formulário da Figura 4.11 68

5.1 Terminologia comum do domínio FWS 92
5.2 Aspectos similares 93
5.3 Variabilidades identificadas 93
5.4 Representação tabular da linguagem de modelagem de aplicações para o domínio FWS 94
5.5 Elementos do formulário de especificação FWS 95
5.6 Elementos do formulário de especificação de bóia 96
5.7 Elementos do formulário de especificação de sensor 96
5.8 Terminologia comum do domínio de persistência 109
5.9 Representação tabular do formulário do padrão 1 variante 1 110
5.10 Representação tabular do formulário do padrão 2 variante 2 110
5.11 Representação tabular do formulário do padrão 4 variante 1 111

Siglas

Tabela 1: Lista de Siglas

Sigla	**Descrição**
Captor	Gerador de aplicação configurável desenvolvido neste trabalho (do inglês *Configurable Application Generator*).
FWS	Bóias náuticas (do inglês *Floating Weather Station*).
GAC	Gerador de aplicação configurável.
GRN	Gestão de Recursos de Negócios.
GUI	Interface gráfica com o usuário (do inglês *Graphical User Interface*).
LMA	Linguagem de modelagem de aplicação.
MDA	Arquitetura dirigida a modelos (do inglês, *Model-Driven Architecture*).
MMV	Ferramenta de validação do meta-modelo da configuração da interface da Captor (do inglês, *Meta-Model Validator*).
MTL	Linguagem de mapeamento de transformação de gabaritos (do inglês *Mapping Transformation Language*).
SQL	Linguagem de acesso a bases de dados relacionais (do inglês *Structured Query Language*).
TCP/IP	Protocolo de comunicação de computadores utilizado na internet e em redes privadas.
UML	Linguagem de modelagem de software (do inglês *Unified Modeling Language*).
URL	Endereço de um recurso ou ficheiro disponível na Internet (do inglês *Universal Resource Locator*).
XML	Linguagem de marcação extensível e personalisável (do inglês *Extensible Markup Language*).
XSL	Linguagem de transformação de gabaritos (do inglês *Extendible Stylesheet Language*).

CAPÍTULO

1

Introdução

1.1 Contextualização

A indústria de software tem como uma de suas principais metas a construção de sistemas de alta qualidade, baixo custo e em curto espaço de tempo. Nesse ambiente de alta competitividade, diversos sistemas são desenvolvidos para suprir as necessidades do mercado. A construção desses sistemas possui normalmente um ciclo de vida em que determinadas tarefas são executadas em uma ou mais etapas e, durante a realização dessas tarefas, diversas técnicas podem ser aplicadas para garantir que o projeto atinja os resultados esperados com relação à qualidade, tempo e custo de desenvolvimento.

Uma das técnicas que podem ser aplicadas para facilitar o trabalho que deve ser realizado é o reúso de software, que pode ser definido como o uso de conhecimento ou artefatos de sistemas existentes na construção de sistemas novos (Frakes e Terry, 1995). No processo de desenvolvimento, o reúso pode ser alcançado de diversas formas, tais como: componentes de software, frameworks orientados a objetos, padrões e linguagens de padrões e geradores de aplicação. Nesse contexto, muitos autores afirmam que se o reúso for aplicado de forma sistemática, ele pode trazer benefícios de qualidade e produtividade para as etapas do processo de desenvolvimento (Frakes e Terry, 1995; Frakes e Isoda, 1994; J. E. Gaffney e Cruickshank, 1992; Lim, 1992).

1.2 Motivação

Braga et al. (1999b) criaram a linguagem de padrões GRN e baseado nessa linguagem foi desenvolvido o framework orientado a objetos GREN (Braga, 2003). A linguagem de padrões e o framework facilitam o reúso no desenvolvimento de aplicações que gerenciam recursos de negócios, tais como identificação, quantificação, armazenagem, locação, venda e manutenção de recursos de negócios (Braga, 2003).

Para implementar aplicações específicas com a GRN e o GREN, o desenvolvedor deve selecionar os padrões da linguagem de padrões que devem ser aplicados para representar as necessidades dos clientes e instanciar as classes correspondentes aos padrões escolhidos no framework GREN. Como o GREN é um framework caixa-branca, o processo de instanciação das classes consiste na criação de classes que estendem as classes abstratas do framework, no preenchimento dos métodos-gancho dessas classes, e caso necessário, na criação de novos métodos.

Braga (2003) mostra que embora o reúso seja alcançado com o uso de frameworks, a sua instanciação pode ser uma tarefa complexa e que exige uma curva de aprendizado consideravelmente alta por parte do desenvolvedor. Para ajudar a solucionar esse problema, Braga construiu o GREN-Wizard (Braga e Masiero, 2003), uma ferramenta que aceita uma especificação da linguagem GRN e gera automaticamente o código necessário para instanciar o framework GREN. Com o uso dessa ferramenta, o framework GREN pode ser instanciado rapidamente sem a necessidade de codificação manual, exceto para os requisitos da aplicação não previstos pelo framework.

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). A motivação inicial deste trabalho está nas investigações não concluídas de Braga e Masiero sobre a construção de uma ferramenta genérica que pudesse ser utilizada para instanciar diversos frameworks baseados em linguagens de padrões. Essa ferramenta deveria ser configurada para fornecer apoio no desenvolvimento de aplicações em domínios específicos e gerar o código necessário para instanciar frameworks orientados a objetos.

1.3 Objetivos

Como foi apresentado na Seção 1.2, um dos principais problemas encontrados no uso de ferramentas de geração de artefatos é o seu custo de desenvolvimento, utilização e manutenção. Como essas ferramentas apresentam um custo elevado, existe um risco muito alto no seu desenvolvimento e utilização, pois se o investimento necessário para a sua construção não for

recompensado durante o seu uso posterior, utilizar essas ferramentas pode aumentar o esforço e o custo necessários para construir o sistema de software.

Este trabalho tem o objetivo de contribuir para diminuir esse risco, de forma que seja economicamente viável utilizar geradores de aplicações na construção de diversos sistemas de software, aumentar a qualidade do produto final e aumentar a produtividade em diversas fases do processo de desenvolvimento.

Para isso, este trabalho propõe a criação de um gerador de aplicação configurável, ou seja, um gerador de aplicação que pode ser configurado para realizar a geração de artefatos em domínios diferentes e delineia um processo de desenvolvimento apoiado pelo gerador configurável. Com essa abordagem, para obter 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.

1.4 Organização

Este trabalho está organizado da seguinte forma: no Capítulo 2 são apresentadas algumas formas de reúso de software, tais como: padrões e linguagens de padrões, frameworks orientados a objetos, processo de engenharia de software de linha de produtos e geradores de aplicação. Os padrões, frameworks e linhas de produtos são apresentados para introduzir alguns dos conceitos utilizados nos capítulos posteriores e os geradores de aplicação são apresentados para introduzir o principal objeto de estudo deste trabalho.

No Capítulo 3 são apresentados de forma genérica os geradores de aplicação configuráveis, a sua arquitetura e a proposta de um processo de desenvolvimento de software com uso de geradores de aplicação configuráveis. Um exemplo simples de persistência de dados é utilizado para ilustrar esses conceitos.

No Capítulo 4 é apresentado o gerador de aplicação configurável Captor. Esse gerador foi construído com base nos conceitos apresentados no Capítulo 3 e a sua arquitetura e o seu processo de configuração e utilização são detalhados.

No Capítulo 5 são apresentados dois estudos de caso em que o gerador de aplicação configurável Captor é utilizado para a construção de sistemas em dois domínios diferentes: uma família simples de sistemas para bóias náuticas, descrita anteriormente por Weiss e Lai (1999) e sistemas para a gestão de recursos de negócios, definidos anteriormente por Braga (2003); Braga et al. (1999b).

A Captor foi configurada para apoiar o domínio de persistência, mas tal configuração não é mostrada em detalhes. O exemplo mostrado no Capítulo 3 foi baseado nessa configuração, mas é mostrado de uma forma independente de implementação. Trechos da configuração baseados na implementação da Captor são mostrados no Capítulo 4.

Finalmente, no Capítulo 6 são apresentadas as conclusões deste trabalho.

CAPÍTULO

2

Reúso de software

2.1 Considerações iniciais

Um dos principais conceitos explorados neste trabalho é o reúso. O reúso de software é 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).

A reutilização de artefatos de software é um dos meios pelos quais a engenharia de software pode trazer benefícios de qualidade e produtividade durante o processo de desenvolvimento de sistemas computacionais. Existem diversas técnicas disponíveis para o desenvolvedor alcançar múltiplas formas de reúso sobre artefatos de software durante todas as fases do processo de construção de sistemas. Essas técnicas 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. Dentre as técnicas de reúso disponíveis na literatura podem ser citadas: padrões de software, frameworks de modelos e *template packages* (D'Souza e Wills, 1999), bibliotecas de componentes e frameworks de software orientados a objetos.

Este Capítulo apresenta algumas das principais formas de reúso de artefatos de software encontradas na literatura e está organizado da seguinte forma: na Seção 2.2 são apresentados os padrões de software e alguns exemplos encontrados na literatura; na Seção 2.3 são apresentados os frameworks de software orientados a objetos e alguns exemplos encontrados na literatura; na

Seção 2.4 são apresentados os geradores de código; finalmente, na Seção 2.5 são apresentados os principais conceitos da engenharia de software de linhas de produtos e o processo FAST (Weiss e Lai, 1999).

2.2 Padrões de Software

Construir software não é uma tarefa fácil. Desenvolvedores experientes possuem maior facilidade na atividade de construção de software por possuirem conhecimento de soluções recorrentes que podem ser aplicadas em diversas situações similares. Essa reutilização de soluções recorrentes pode ser documentada adequadamente no formato de padrões. Um padrão pode ser visto como a descrição de uma solução de um problema recorrente em um determinado ambiente, acompanhado do núcleo de sua solução de uma forma que facilite a sua utilização diversas vezes, sem no entanto, implementar a solução da mesma forma duas vezes (Alexander, 1979).

Na engenharia de software, padrões encapsulam as melhores soluções baseadas em anos de desenvolvimento de aplicações, observação e experiência. Para encontrar a melhor solução, o desenvolvedor deve entender o problema, o contexto e as forças que governam esse problema (Harrison et al., 1999). Dessa forma, os padrões ajudam a construir sistemas confiáveis seguindo os passos de outras construções de sistemas de sucesso (Harrison et al., 1999).

Os padrões de software identificam e especificam abstrações de mais alto nível do que classes, instâncias ou componentes (Gamma et al., 1995). Normalmente um padrão de software descreve componentes, classes ou objetos e os detalhes de suas responsabilidades e relacionamentos (Buschmann et al., 2000). Padrões documentam a experiência comprovada de projeto, além de fornecer um vocabulário comum para o entendimento dos princípios de um projeto (Gamma et al., 1995).

Os padrões de software devem ser documentados em um formato adequado. Diversas formas para documentar padrões foram propostas na literatura. Alexander (1979), em seu trabalho *The Timeless way of Building*, define que cada padrão deve ser representado na forma de uma regra que estabelece uma relação entre um contexto, um sistema de forças que aparece nesse contexto e uma configuração, a qual permite que as forças se equilibrem no referido contexto. A *Gangue dos Quatro* (*GoF*, 1995) descreve os padrões baseados em um gabarito, que é formado pelos seguintes componentes: nome do padrão e classificação, intenção, nome alternativo, motivação, aplicabilidade, estrutura, participantes, colaborações, conseqüências, implementação, exemplos de código, usos conhecidos e padrões relacionados. Buschmann (2000), utiliza a tríade: contexto, problema e solução para descrever os principais componentes que formam a descrição dos padrões. O formato utilizado por Alexander é conhecido como forma *alexandriana* ou forma canônica e o formato utilizado pela *Gangue dos Quatro* é conhecido como formato *GoF* (Appleton, 2000).

Os padrões de software têm sido explorados para a documentação de soluções em diversas áreas da engenharia de software e podem ser categorizados de acordo com as suas principais características. Alguns padrões ajudam a implementar aspectos particulares de uma linguagem de programação e outros são utilizados em domínios específicos. As principais categorias de padrões de software encontradas na literatura são: padrões de projeto, padrões de análise, padrões arquiteturais, padrões de código e padrões de usabilidade. A seguir são apresentados os principais tipos de padrões explorados neste trabalho.

2.2.1 Padrões de projeto

Padrões de projeto fornecem um esquema para o refinamento de subsistemas ou componentes de software e o relacionamento entre eles. Eles descrevem uma estrutura recorrente de comunicação de componentes, que resolve um problema de projeto genérico em um contexto particular (Gamma et al., 1995). Padrões de projeto são independentes de linguagem de programação e de paradigma. Alguns fornecem um meio pelo qual pode-se decompor serviços ou componentes complexos, e outros fornecem um meio efetivo de se promover a cooperação entre esses componentes (Buschmann et al., 2000).

Como os padrões de projeto são o resultado da documentação de projetos de desenvolvedores experientes, eles ajudam na busca dos objetos adequados para as funcionalidades necessárias em uma aplicação, na determinação correta da granularidade dos objetos, na determinação das interfaces dos objetos, no aumento da facilidade de reúso e no desenvolvimento voltado para mudanças, entre outros pontos positivos que podem ser coletados da experiência dos desenvolvedores.

Como exemplo, pode-se citar o padrão de projeto método-gabarito (do inglês *Template Method*), que ajuda na definição do esqueleto de um algoritmo, deixando a definição de partes variáveis desse algoritmo para as sub-classes (Gamma et al., 1995). Este padrão é bastante utilizado no projeto de frameworks e um resumo desse padrão é apresentado na Figura 2.1.

2.2.2 Padrões arquiteturais

A arquitetura de software de um programa ou sistema computacional é a estrutura ou conjunto de estruturas que abrangem a definição dos componentes de software, as propriedades externamente visíveis desses componentes e a relação entre eles (Pressman, 2001). Um padrão arquitetural representa o esquema da estrutura organizacional de sistemas de software. Ele fornece um conjunto de subsistemas predefinidos, especificando as suas responsabilidades e as regras para organizar os seus relacionamentos (Buschmann et al., 2000).

Padrões arquiteturais são gabaritos para arquiteturas de software concretas. Eles especificam as propriedades estruturais de toda uma aplicação e possuem impacto na arquitetura dos

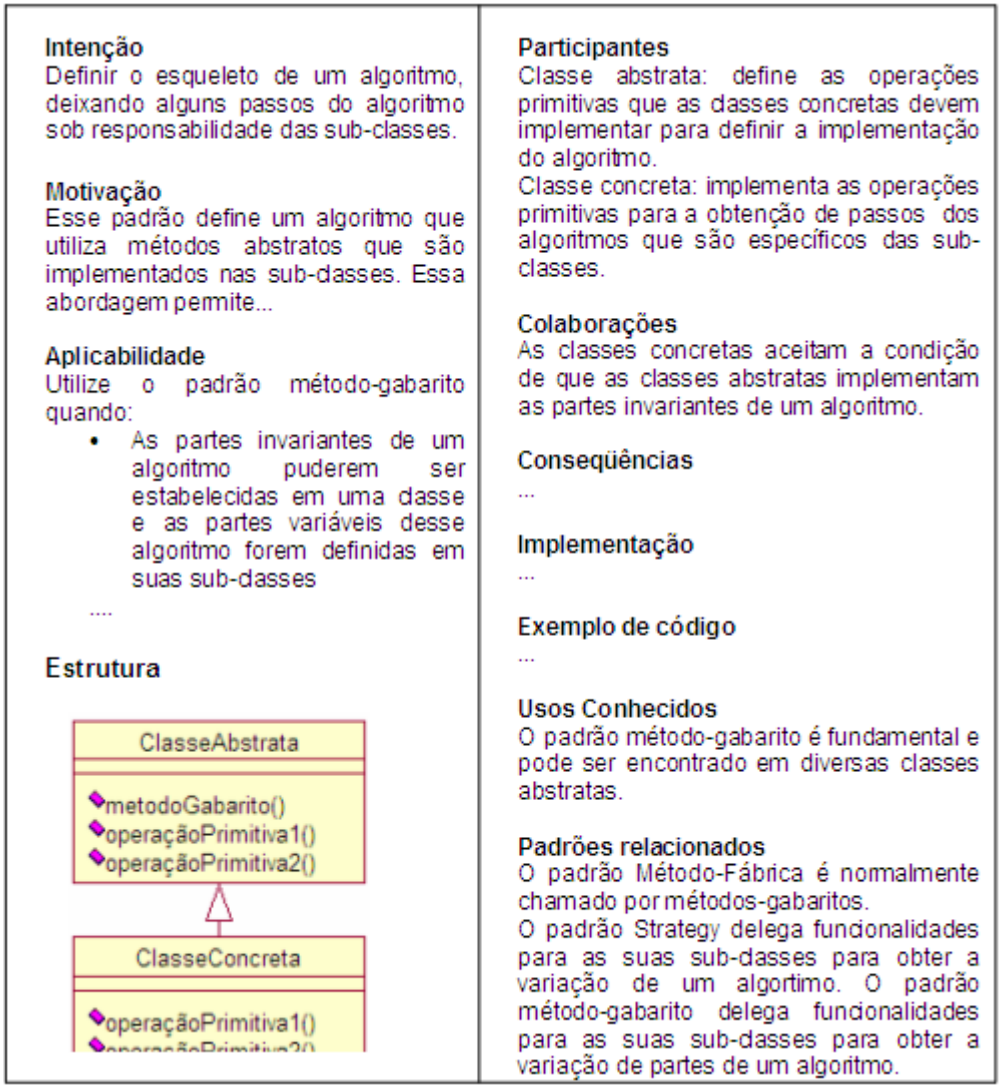

Figura 2.1: Padrão de projeto método-gabarito (Gamma et al., 1995)

subsistemas. A seleção de um padrão arquitetural é uma decisão de projeto fundamental durante o desenvolvimento de um sistema de software (Buschmann et al., 2000). Os padrões arquiteturais não são necessariamente utilizados isoladamente, e um sistema pode ser implementado utilizando um ou mais padrões em sua arquitetura.

Um dos padrões arquiteturais mais conhecidos é o padrão Modelo-Visão-Controlador (do inglês *Model-View-Controller*) ou MVC, bastante utilizado em sistemas que requerem interfaces gráficas com o usuário. A arquitetura MVC é modelada para apoiar a produção de aplicações de software com foco no elemento humano introduzido pelo usuário. Esse padrão divide uma aplicação em três partes: o modelo, a visualização e o controle (Goldberg, 1995). Essa divisão ajuda na separação de interesses entre o modelo da aplicação (ou classes de negócio) e os componentes relacionados com as telas gráficas responsáveis pela visualização do usuário. O controle e a coordenação entre o modelo e a visualização é realizado pelos componentes controladores.

2.2.3 Linguagem de padrões

Nenhum padrão é uma entidade isolada e cada padrão é apoiado por outros padrões (Alexander, 1979). Uma coleção de padrões relacionados forma um vocabulário para o entendimento e comunicação de idéias. Essa coleção pode ser integrada em um conjunto coesivo que revela as estruturas inerentes e os relacionamentos das partes constituintes de um objetivo compartilhado (Appleton, 2000). Quando uma coleção de padrões é estruturada dessa forma, eles formam uma linguagem, denominada linguagem de padrões, que define um conjunto de abstrações, as quais formam um vocabulário e um conjunto de regras. Essas regras formam uma gramática que permite a combinação dessas abstrações para a formação de expressões de uma linguagem (Greenfield e Short, 2004).

Uma linguagem de padrões fornece sugestões sobre como os padrões podem ser combinados para a solução de problemas específicos. Essas sugestões descrevem o processo de aplicação dos padrões, ou podem definir restrições que a aplicação possui, como por exemplo, a ordem na qual os padrões devem ser aplicados (Greenfield e Short, 2004).

Existem diversas linguagens de padrões propostas na literatura, tais como a linguagem *Crossing Chasms* (Brown e Whitenack, 1995) que foi desenvolvida para ajudar na integração de objetos com sistemas gerenciadores de banco de dados relacionais; a linguagem Tropyc (Braga et al., 1999a) fornece os padrões para serviços relacionados à segurança, tais como confidencialidade de dados, integridade de dados e autenticação de envio e a linguagem GRN que é uma linguagem que fornece apoio na Gestão de Recursos de Negócios (Braga, 2003; Braga et al., 1999b; Braga e Masiero, 2001, 2002).

2.2.3.1 Linguagem de padrões de Gestão de Recusos de Negócio (GRN)

A linguagem de padrões GRN é descrita em detalhes nesta Seção porque a motivação deste trabalho é baseada no projeto de pesquisa que desenvolveu essa linguagem, e também porque no Capítulo 5, ela é utilizada em um estudo de caso que apresenta a instanciação automática de frameworks por meio de geradores de aplicação configuráveis.

Essa linguagem é formada por quinze padrões de análise, alguns dos quais são aplicações ou extensões de padrões recorrentes propostos na literatura. Contudo, a linguagem de padrões GRN está localizada em um patamar de abstração superior em relação a esses padrões recorrentes, já que é aplicável a um domínio mais específico e contém valor semântico inerente a uma família de aplicações desse domínio. Ela foi concebida para auxiliar os engenheiros de software a desenvolver aplicações que lidem com gestão de recursos de negócios - mais especificamente - aplicações nas quais seja necessário registrar transações de aluguel, comercialização ou manutenção dos bens ou serviços. Braga utiliza a palavra “transação” com o mesmo significado

definido por Coad et al. (1997), ou seja, um evento significativo que precisa ser lembrado pelo sistema ao longo do tempo.

O aluguel de recursos enfoca principalmente a utilização temporária de um bem ou serviço, como por exemplo uma fita de vídeo ou o tempo de um especialista. A comercialização de recursos enfoca a transferência de propriedade de um bem, como por exemplo uma venda ou leilão de produtos. A manutenção de recursos enfoca o reparo ou conservação de um determinado produto, utilizando mão-de-obra e peças para execução, como é o exemplo de uma oficina de reparo de eletrodomésticos.

A Figura 2.2 mostra as dependências entre os padrões da linguagem de padrões GRN e a ordem na qual eles são geralmente aplicados. Essas dependências são também apresentadas e, eventualmente, complementadas em uma seção específica de cada padrão, a seção "Próximos padrões". Os principais padrões da linguagem são indicados por uma linha mais grossa. São eles: LOCAR O RECURSO, COMERCIALIZAR O RECURSO e MANTER O RECURSO. O uso desses padrões não é mutuamente exclusivo. Na verdade, existem aplicações nas quais eles podem ser usados conjuntamente, como por exemplo em uma locadora de veículos que também compra, vende e conserta seus próprios carros.

O primeiro padrão a ser aplicado é IDENTIFICAR O RECURSO. Os padrões ITEMIZAR A TRANSAÇÃO DO RECURSO, PAGAR PELA TRANSAÇÃO DO RECURSO e IDENTIFICAR O EXECUTOR DA TRANSAÇÃO são mostrados dentro de uma caixa, denotando que são aplicáveis a todas as situações nas quais uma seta chega até a borda dessa caixa. A seta sem origem que chega ao padrão 11 significa que esse é o primeiro padrão a ser verificado, seguido dos padrões 12 e 13.

Como ilustração da linguagem GRN, a seguir é apresentada a descrição do primeiro padrão da linguagem, extraído de Braga et al. (1999b):

Padrão 1: IDENTIFICAR O RECURSO

Contexto

Seu sistema de negócio lida com uma ou mais das seguintes transações: pedidos, compras, vendas, locações, aluguéis, atribuições, reservas, consertos ou manutenções. Essas transações referem-se, por exemplo, a produtos de uma loja, fitas de vídeo em uma locadora, livros em uma biblioteca, tempo de um especialista em uma clínica médica ou carros em uma oficina mecânica. Todos esses casos tratam de recursos de negócio gerenciados por sistemas específicos.

Problema

Como representar os recursos de negócio envolvidos nas transações processadas pelo sistema?

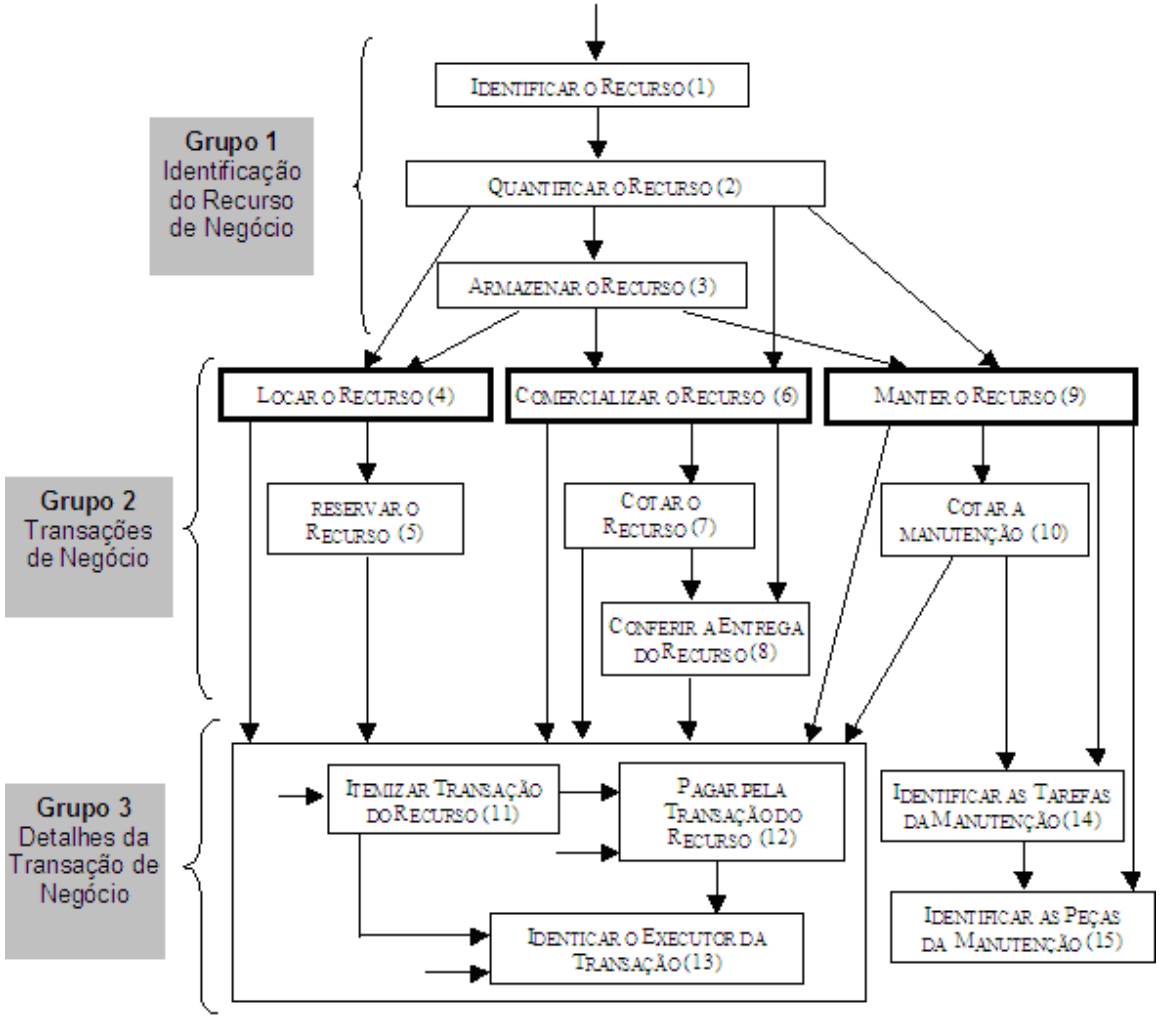

Figura 2.2: Relacionamento entre os padrões da linguagem de padrões GRN (Braga, 2003)

Influências

- Recursos de negócio têm, em geral, atributos ou qualidades em comum. Guardar as informações pertinentes a cada recurso em particular é importante para as empresas envolvidas em seu gerenciamento

- Recursos de negócio são freqüentemente classificados em diversas categorias. Por exemplo, em uma locadora de vídeos, os filmes podem ser agrupados em categorias como Aventura, Suspense, Romance, Comédia, etc. Essa qualificação é útil na obtenção de relatórios expressivos como, por exemplo, qual é o tipo de filme preferido pelos clientes e que, portanto, merece maior investimento na aquisição. A qualificação de recursos pode ser obtida adicionando um atributo à classe que representa o recurso, de modo que a categoria seja considerada um atributo do recurso. Essa abordagem funciona bem quando a categoria é simplesmente uma descrição do grupo, sem comportamento próprio.

- Quando existem atributos e métodos comuns, inerentes ao grupo delimitado por esse atributo, justifica-se a separação em duas classes. O espaço necessário para manter esses atributos para todo recurso e a redundância obtida são indesejáveis. Entretanto, a separação pode aumentar o tempo de processamento, já que uma classe precisará referenciar a outra e essa referência deve ser cuidada pelo sistema. Isso deve ser levado em consideração na otimização do desempenho do sistema.

- Se o recurso possuir apenas alguns atributos, esses podem ser colocados nas classes que representam as transações efetuadas pela organização. Isso facilita a implementação, embora limite a evolução do software e cause redundância de informações.

Estrutura

A Figura 2.3 mostra o diagrama de classes para o padrão IDENTIFICAR O RECURSO.

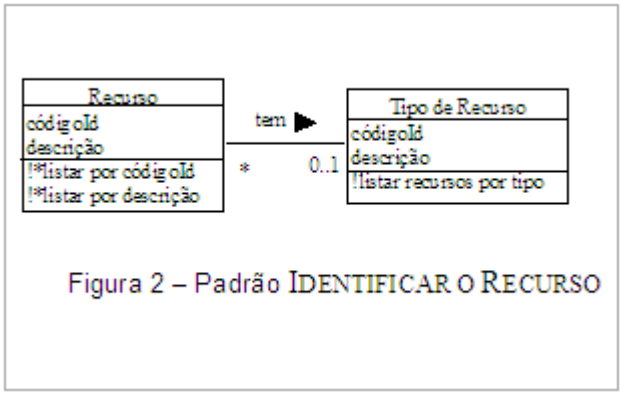

Figura 2.3: Diagrama de classes para o padrão IDENTIFICAR O RECURSO

Participantes

Recurso: representa o recurso de negócio gerenciado pela aplicação, definindo todas as suas características importantes. O atributo *codigoId* é utilizado com freqüência para identificar unicamente cada recurso e o atributo *descricao* para descrevê-lo. Outros atributos podem ser adicionados de acordo com a instância particular do recurso. Além dos métodos básicos da classe, operações especiais são fornecidas para listar os recursos por *codigoId* ou por descrição (ordem alfabética), que são relatórios requisitados com freqüência pela aplicação.

Tipo de Recurso: representa as categorias ou tipos de recurso. Essa classe é opcional, já que em alguns sistemas os recursos não são categorizados.

Exemplo

A Figura 2.4 mostra uma instanciação do padrão IDENTIFICAR O RECURSO para um sistema de locadora de vídeos.

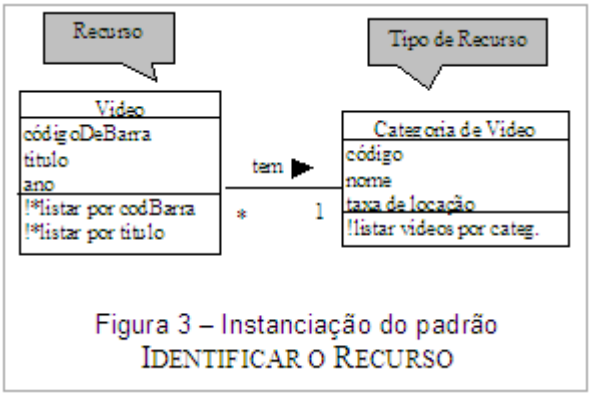

Figura 2.4: Diagrama de classes de um exemplo para o padrão IDENTIFICAR O RECURSO

Variantes

Para melhorar a categorização, o recurso pode ser classificado por mais do que um grupo ou em diversos níveis de agrupamento. Johnson denomina esses dois casos de objetos com múltiplos tipos (*multiple type objects*) e objetos com tipos aninhados (*nested type objects*), respectivamente (Johnson e Woolf, 1998). No primeiro caso, cria-se uma nova classe para cada categoria desejada, ligada à classe Resource. Por exemplo, em um sistema de locadora de vídeos, além da qualificação dos filmes mencionada acima, outra qualificação poderia ser adicionada para agrupar os filmes segundo sua freqüência de locação (“A” para locado freqüentemente, “B” para locado moderadamente e “C” para raramente locado). Isso poderia permitir, por exemplo, a definição de preços de locação de acordo com esses níveis. No segundo caso, criam-se novas classes de acordo com o grau de qualificação desejado, como em um sistema de contabilidade. Por exemplo, um carro pode ser categorizado por seu modelo, que, por sua vez, pode ser categorizado por fabricante.

Próximos padrões

Depois de IDENTIFICAR O RECURSO (1), tente QUANTIFICAR O RECURSO (2).

2.3 Frameworks de Software Orientados a Objetos

Pequenos componentes de software, tais como sub-rotinas ou bibliotecas de objetos, são muito utilizados por desenvolvedores. Esses artefatos permitem o reúso de pequenas partes do sistema e podem diminuir consideravelmente os esforços de desenvolvimento. O problema de construir software reutilizando apenas pequenos componentes de software é que o desenvolvedor ainda tem muito trabalho a ser feito, como construir uma super-estrutura arquitetural que concentre esses componentes menores dentro do sistema (Biggerstaff e Perlis, 1989). O custo

de construir essa super-estrutura é muito alto e pode ser reduzido utilizando-se frameworks orientados a objetos.

Um framework [1] é um conjunto de classes que contém o projeto abstrato de soluções para uma família de problemas relacionados, propiciando o reúso com granularidade maior do que classes (Johnson e Foote, 1988). Dessa forma, um framework pode ser visto como o esqueleto de uma aplicação que pode ser adaptado por um desenvolvedor (Fayad et al., 1999) para uso em um sistema particular.

Os frameworks são compostos de partes fixas e partes variáveis. As partes fixas são denominadas pontos-fixos (do inglês *frozen-spots*) e são estáveis e imutáveis durante o uso do framework em diversas aplicações. As partes variáveis são denominadas pontos variáveis (do inglês *hot-spots*) e devem ser adaptadas de forma particular em cada instanciação do framework.

Uma característica marcante dos frameworks é a inversão de controle. Essa característica se manifesta durante o fluxo de execução da aplicação. Com o uso de bibliotecas de funções, o controle de execução é de responsabilidade do programa principal e este deve chamar os métodos ou procedimentos contidos na aplicação ou em bibliotecas externas. Com o uso de frameworks, o controle passa do programa principal para o framework, que deverá controlar o fluxo de execução da aplicação e delegar o controle para as demais partes do sistema.

Uma característica importante nos frameworks é o método utilizado para adaptar as partes variáveis de sua estrutura. Um framework é denominado caixa-branca se para adaptá-lo for utilizada herança, em outras palavras, o desenvolvedor deve criar subclasses a partir das classes do framework e adicionar comportamento específico aos pontos variáveis de sua estrutura, implementando métodos-gancho (do inglês *hook-methods*) nessas novas classes (Johnson e Foote, 1988). Quando um framework permite que suas partes variáveis sejam modificadas por meio de um conjunto de componentes especificados por interfaces bem definidas e inseridos por meio de composição de objetos ao framework, ele é denominado caixa-preta. Um framework caixa-cinza é uma mistura de um framework caixa-branca e caixa preta. Frameworks caixa cinza permitem sua adaptação por meio de herança assim como por meio de interfaces bem definidas (Yassin e Fayad, 2000).

Os frameworks podem ser utilizados em diferentes aplicações de software e em cada aplicação pode ser utilizado um ou mais frameworks. Cada framework pode possuir uma determinada função em um projeto de software orientado a objetos. A funcionalidade que o framework fornece para o desenvolvedor da aplicação pode estar restrita a serviços básicos, tais como persistência, distribuição e gerenciamento de transações ou pode ser mais abrangente, contendo funcionalidades para um amplo domínio de aplicação, como gerenciamento de recursos de ne-

[1] O termo framework será utilizado no resto deste trabalho com o significado de framework de software orientado a objetos.

gócio e engenharia financeira. De acordo com o escopo, um framework pode ser classificado em três categorias (Fayad et al., 1999):

- **Frameworks de infra-estrutura de sistema**
  Este tipo de framework simplifica o desenvolvimento de aplicações, fornecendo serviços básicos de sistemas de software como persistência, interface gráfica com o usuário e ferramentas de processamento de linguagens.

- **Frameworks de integração de *middleware***
  Este tipo de framework é utilizado na integração de aplicações e componentes distribuídos. Frameworks de integração de *middleware* são projetados para melhorar a habilidade de desenvolvedores de software em modularizar, reutilizar e estender a infra-estrutura de software para se trabalhar em ambientes distribuídos.

- **Frameworks de aplicação**
  Frameworks que constituem uma aplicação genérica para um domínio em particular são chamados de frameworks de aplicação (Pree, 1995). Esses frameworks abrangem domínios de aplicação maiores, tais como o gerenciamento de recursos de negócio, engenharia financeira e telecomunicações. Comparados com frameworks de integração de *middleware* e de infra-estrutura, frameworks de aplicação empresarial são mais caros de desenvolver ou comprar. Entretanto, esse tipo de framework pode fornecer um retorno substancial de investimento, já que eles podem ser utilizados no desenvolvimento de diversas aplicações completas para o usuário final (Fayad et al., 1999).

O uso de frameworks pode ser visto como uma evolução das técnicas de reúso, se for comparado com os primeiros esforços alcançados por meio das funções e bibliotecas de procedimentos nas épocas iniciais da ciência da computação (Mattsson e Bosch, 1997). Um framework bem projetado deve conter como pontos fixos todo o comportamento comum encontrado na família de problemas para a qual ele fornece cobertura, e como pontos variáveis apenas o que for específico de cada instância no domínio. Se esta atividade for realizada de maneira que os pontos fixos apoiem o maior número de aplicações possíveis e os pontos variáveis possam ser facilmente adaptados para um grande número de aplicações, o desenvolvedor da aplicação vai ser beneficiado com um ótimo grau de reúso entre aplicações dentro do domínio coberto pelo framework.

2.3.1 Exemplos de frameworks relevantes

Existem diversos frameworks disponíveis na literatura e na Web. O framework IBM San Francisco (Monday et al., 2000) é um framework de aplicação empresarial desenvolvido pela

IBM para fornecer apoio a diversas transações de negócio, tais como Contas a Pagar, Contas a Receber e Gestão de Pedidos; o Hotdraw (Johnson, 1992) é um framework para o desenvolvimento de editores gráficos e pode ser utilizado para especializar editores de duas dimensões, tais como editores esquemáticos e projeto de programas. Os elementos dos desenhos podem ter restrições entre si, podem reagir a comandos do usuário e podem ser animados. O editor especializado pode ser uma aplicação completa ou parte de um sistema maior. Os frameworks utilizados neste trabalho, em conjunto com o gerador de aplicações, são apresentados a seguir:

GREN

O framework GREN (Braga, 2003) é um framework de aplicação empresarial e foi desenvolvido com base na linguagem de padrões GRN (Braga, 2003) (ver Seção 2.2.3.1). O GREN é um framework caixa-branca e permite a criação de aplicações no domínio de sistemas de gestão de recursos de negócio.

A arquitetura do GREN foi definida em três camadas: a camada de persistência, a camada de negócios e a camada de interface gráfica com o usuário (GUI), conforme ilustrado na Figura 2.5.

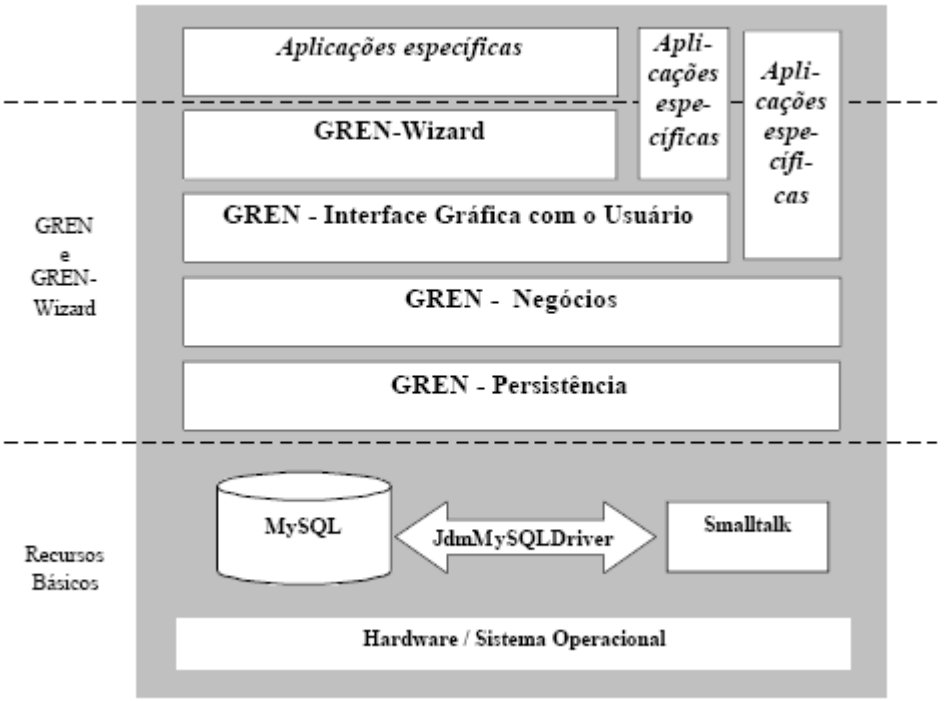

Figura 2.5: Arquitetura do GREN

A camada de persistência possui classes para cuidar da conexão com a base de dados, gerenciamento dos identificadores dos objetos e persistência dos objetos. A camada de negócios comunica-se com a camada de persistência sempre que precisa armazenar permanentemente um objeto. Dentro da camada de negócios existem diversas classes derivadas diretamente dos padrões de análise que compõem a linguagem de padrões GRN, ou seja, as classes e relacionamentos contidos em cada padrão possuem a implementação correspondente nesta camada. A camada de interface gráfica com o usuário contém as classes responsáveis pela entrada e saída de dados, como formulários de interface, janelas e menus, que permitem a interação do usuário final com o sistema. Comunica-se com a camada de negócios para obtenção de objetos a serem mostrados na interface com o usuário e para devolver informações a serem processadas pelos métodos da camada de negócios.

Aplicações específicas podem ser construídas a partir da camada de interface gráfica com o usuário, usando (por meio de herança ou referência a objetos) classes de todas as camadas do GREN, ou podem ser construídas com base na camada de negócios, caso a camada de interface gráfica com o usuário não seja reutilizada a partir do GREN, mas implementada separadamente.

O projeto da hierarquia de classes começou pela construção de um modelo de classes geral, englobando as classes de todos os quinze padrões da GRN, e depois fazendo uso dos mecanismos de especialização e generalização para refinar esse modelo. A seguir foram utilizados padrões de projeto, como por exemplo o *Strategy*, o *Template Method* e o *Factory Method* (Gamma et al., 1995), para conseguir a flexibilidade necessária para atender aos pontos variáveis do framework.

A Figura 2.6 mostra parte do diagrama de classes de projeto do GREN. Nesse exemplo em particular, são mostradas as classes relacionadas à classe *Resource* (Recurso). A classe *PersistentObject* foi criada para modelar a camada de persistência, conforme o padrão *Persistence-Layer* (Yoder et al., 1998). As classes *StaticObject* e *QualifiableObject* foram criadas durante a etapa de generalização/especialização, visto que diversas classes podem herdar seu comportamento. O padrão de projeto *Strategy* (Gamma et al., 1995) foi utilizado para modelar um dos pontos variáveis do GREN. Cada objeto da classe *Resource* refere-se a um objeto da classe *QuantificationStrategy* da Figura 2.6, o qual é responsável por seus aspectos de quantificação, permitindo que o framework implemente as quatro diferentes soluções requeridas.

Para a implementação das classes utilizou-se a linguagem Smalltalk, mais especificamente o ambiente VisualWorks Non-Commercial 5i.4 (Cincom, 2001). A base de dados (relacional) utilizada na persistência dos objetos é a MySQL (MySQL, 2001). A primeira versão do GREN possui aproximadamente 160 classes e 30 mil linhas de código.

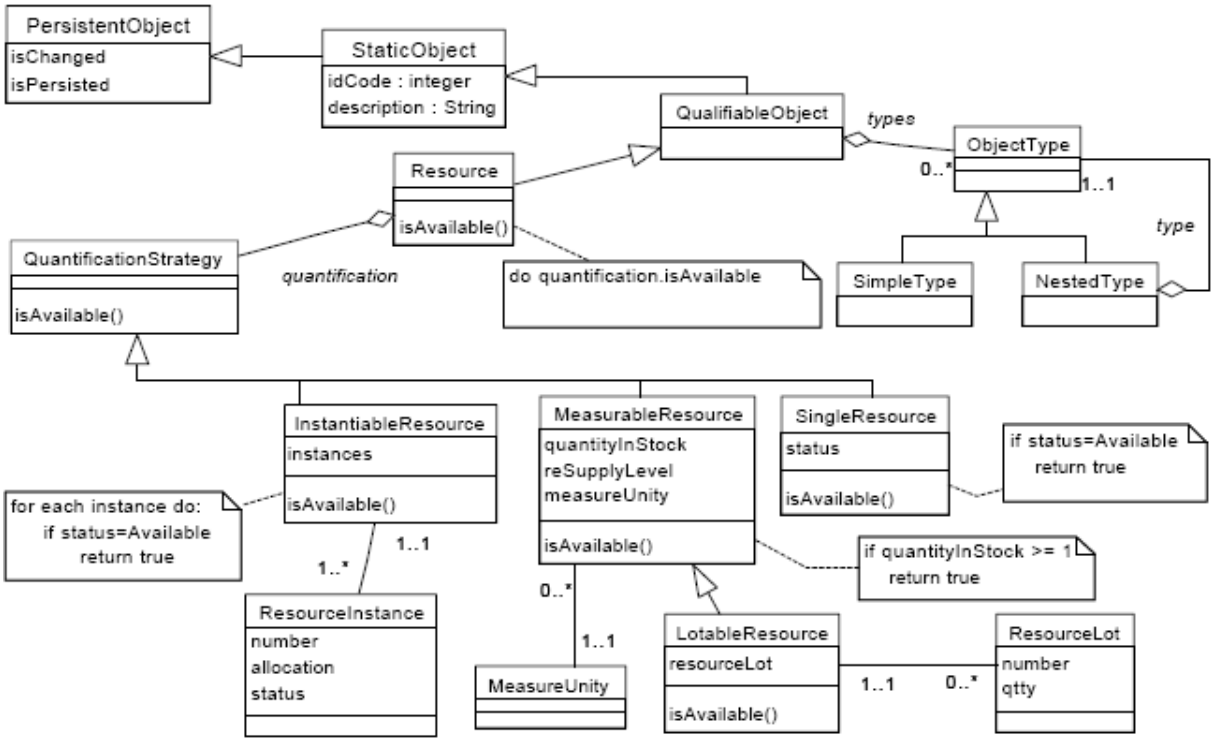

Figura 2.6: Exemplo de algumas classes do GREN

2.4 Geração de código

A automação da produção de artefatos de software é uma atividade que pode diminuir o tempo e o custo de desenvolvimento de aplicações. Existem diversas técnicas que podem ser utilizadas para a geração de software. Essas técnicas têm o objetivo de fornecer meios para o engenheiro especificar uma aplicação em uma linguagem abstrata de alto nível e, a partir dessa especificação, uma ferramenta realiza a construção automática de artefatos, tais como código-fonte, casos de teste e documentação.

Dentre as técnicas de automação disponíveis na literatura estão as linguagens específicas de domínio, a arquitetura baseada em modelos e os geradores de aplicação.

As linguagems específicas de domínio ou DSL (do inglês *Domain-Specific Language*) são linguagens de programação ou linguagens de especificação executáveis que oferecem, por meio de notações e abstrações apropriadas, poder de expressar soluções em um problema de domínio particular (Deursen et al., 2000). DSLs são linguagens especializadas e orientadas ao problema, elas podem ser textuais, tais como a linguagem SQL (do inglês *Structured Query Language*), ou podem ser gráficas, tais como uma especificação gráfica em uma ferramenta GUI (do inglês *Graphical User Interface*) (Tolvanen et al., 2002; Czarnecki e Eisenercker, 2002; Greenfield e Short, 2004). As DSLs normalmente são apoiadas por um compilador que gera artefatos a partir de uma especificação (Deursen e Klint, 1997).

A arquitetura baseada em modelos ou MDA (do inglês *Model Driven Architecture*), é uma que utiliza modelos para direcionar diretamente o entendimento, projeto, construção, operação, manutenção e modificação do sistema. O MDA mantém o foco na separação entre a especificação e a funcionalidade do sistema, e entre a especificação e a implementação do sistema em uma determinada plataforma de desenvolvimento (Miller e Mukerji, 2003). Para atingir esses objetivos, o MDA define uma arquitetura que fornece um conjunto de regras para estruturar especificações baseadas em modelos UML (Booch et al., 2000) e obter a geração automática de artefatos por meio desses modelos.

Os geradores de aplicação são objetos de estudo deste trabalho e são detalhados com maior profundidade na Seção 2.4.1.

2.4.1 Geradores de aplicação

A automação pode transformar as atividades rotineiras de codificação, teste e documentação de um artefato de software em uma atividade automática, na qual o desenvolvedor insere uma especificação abstrata em um gerador de aplicação, e as atividades rotineiras são realizadas automaticamente pela ferramenta. Com o uso de geradores, o desenvolvedor deve inserir apenas as informações sobre "o que" deve ser feito, e a ferramenta deve decidir "como" essas informações são transformadas em código fonte (Smaragdakis e Batory, 2000; Cleaveland, 2001).

O termo gerador de aplicação 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). Por exemplo, um *wizard* 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, uma vez que recebem uma informação em um nível de abstração, tais como a linguagem Java ou C++, e transformam essa informação em uma linguagem de mais baixo nível, tais como código objeto, bytecode ou código de máquina (Czarnecki e Eisenercker, 2002).

Como Weiss(1999) descreve, existem duas possibilidades para construir um gerador de aplicação: construir um compilador ou um compositor. Construir um compilador significa criar um analisador léxico, sintático e semântico para uma linguagem, enquanto que construir um compositor significa criar um projeto de software, derivar um conjunto de gabaritos (do inglês *templates*) a partir desse projeto, criar um mapeamento entre a especificação e esses gabaritos, para em seguida uma ferramenta utilizar as especificações junto com os gabaritos para gerar artefatos.

De maneira geral, um gerador de aplicação[2] realiza as seguintes tarefas (Czarnecki e Eisenercker, 2002):

- Realiza a validação da especificação de entrada e relata erros ou avisos de inconsistências.
- Completa a especificação utilizando as configurações-padrão, caso seja necessário.
- Realiza otimizações
- Gera a implementação

Os geradores de aplicação ajudam uma organização a construir múltiplos produtos de uma família com mais facilidade do que pela 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 codificação que utiliza geradores durante a engenharia de aplicação 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 (Smaragdakis e Batory, 2000; Cleaveland, 2001; Czarnecki e Eisenercker, 2002).

Existem duas classificações para os tipos de transformações que um gerador pode fazer: 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 (Czarnecki e Eisenercker, 2002). Um diagrama esquemático das transformações horizontais, verticais e oblíquas é apresentado na Figura 2.7.

[2] A partir desse ponto, o termo gerador deverá assumir o significado de uma ferramenta que implementa essas funcionalidades ou parte dessas funcionalidades.

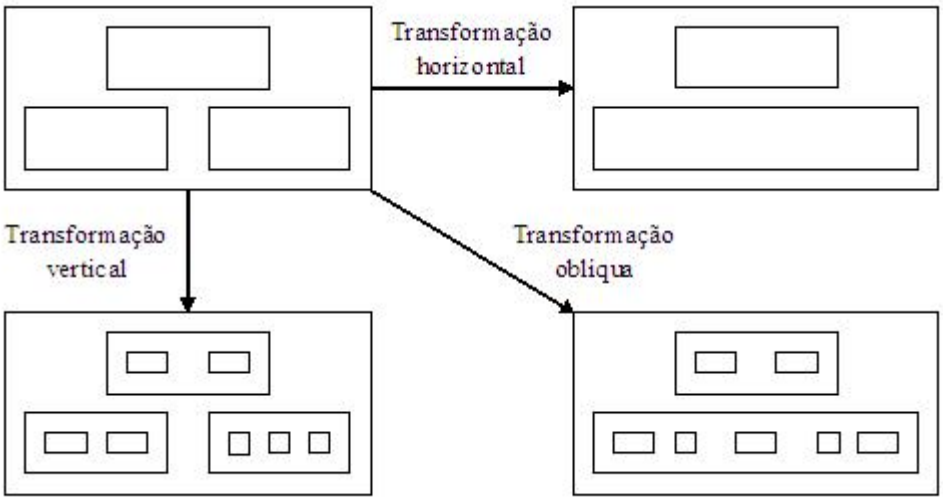

Figura 2.7: Transformações vertical, horizontal e oblíqua (Czarnecki e Eisenercker, 2002)

2.4.2 Alteração do software gerado

Após o software ser gerado, testado e implantado, o sistema pode precisar de algum tipo de alteração. A alteração dos artefatos gerados por uma ferramenta pode representar um problema maior do que a alteração de aplicações desenvolvidas manualmente.

Se o gerenciamento de configuração e a proteção dos arquivos gerados da edição manual não forem realizadas durante as atividades de alteração, o sistema pode se desgastar rapidamente. Como apresentado na Seção 2.4.1, durante a geração de artefatos, o engenheiro insere uma especificação abstrata em uma ferramenta e essa ferramenta gera os artefatos. Na Figura 2.8 é apresentado o processo de alteração dos artefatos gerados.

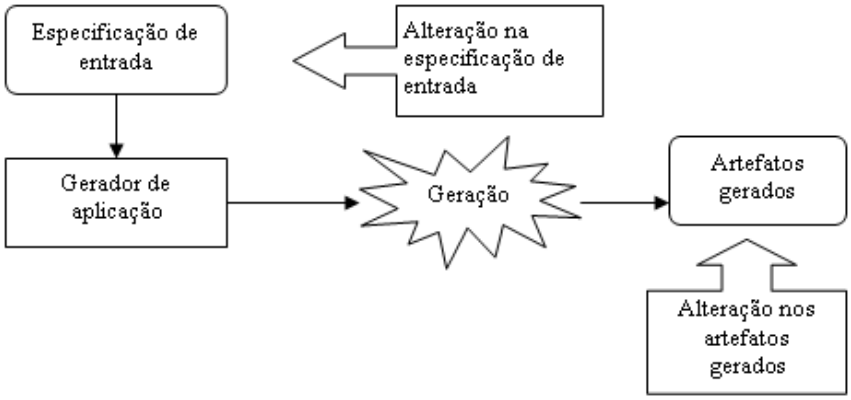

Figura 2.8: Manutenção nos artefatos gerados

Dentro do processo de alteração, as modificações podem ser necessárias na especificação ou nos artefatos gerados. Se a manutenção for realizada na especificação, a única atividade necessária é a realização da geração dos novos artefatos com uma versão compatível da mesma

ferramenta e implantação desses artefatos. Se a manutenção for necessária nos artefatos gerados, o engenheiro deve controlar as versões dos artefatos. A falta de controle pode resultar na perda das informações inseridas manualmente, principalmente nos casos em que o software é gerado automaticamente em um momento posterior à edição manual.

O gerenciamento de configuração e o controle de versões é uma atividade fundamental para os sistemas de software que são gerados por ferramentas de automação. Além dessas atividades, o engenheiro pode utilizar diversas técnicas para permitir a alteração do software gerado de forma mais eficaz. Nesta Seção são apresentadas duas técnicas de geração que permitem que o sofware gerado possa ser modificado sem a perda de informações inseridas manualmente com a re-geração dos artefatos: métodos *delegate* e zonas de segurança. A técnica de zonas de segurança é aplicada na construção do gerador Captor, descrito no Capítulo 4.

Métodos *delegate*

Métodos *delegate* (Greenfield e Short, 2004) são técnicas utilizadas para permitir a alteração do código gerado sem a necessidade da modificação direta desses artefatos. O código gerado é projetado para receber alteração em pontos específicos e delegar as funcionalidades que podem precisar de alteração para outros artefatos. Na Figura 2.9 apresentam-se quatro maneiras de se construir software gerado com métodos *delegate*.

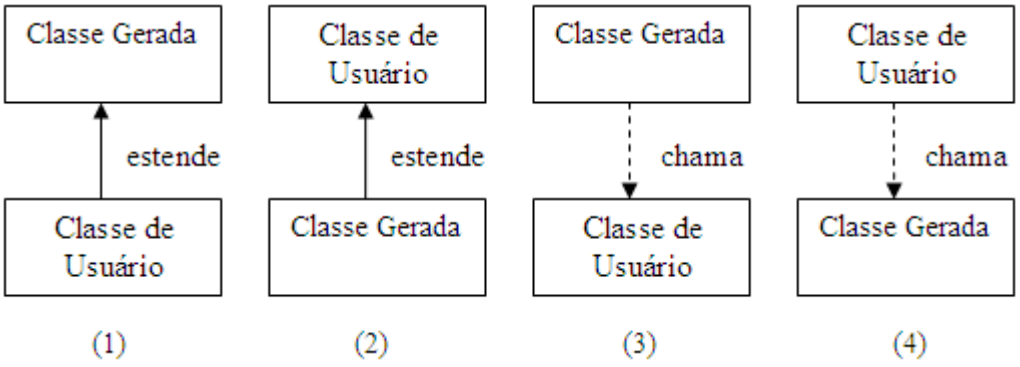

Figura 2.9: Métodos delegate

As classes geradas (Figura 2.9) são classes produzidas automaticamente por um gerador e as classes de usuário são classes editadas manualmente por um programador. As classes de usuário são utilizadas para adicionar funcionalidades para os quais o gerador não fornece apoio.

Na primeira abordagem, as classes geradas são estendidas por uma classe fornecida pelo desenvolvedor. Esta pode adicionar ou alterar funcionalidades por meio da adição de novos atributos e métodos, sem interferir diretamente no funcionamento da classe gerada. A segunda abordagem é uma inversão na hierarquia da primeira, ou seja, as classes geradas estendem as classes fornecidas pelo desenvolvedor.

Na terceira abordagem as classes geradas possuem chamadas para métodos específicos das classes do desenvolvedor. Dessa forma, para obter a implementação do sistema, o desenvolvedor deve fornecer classes com interfaces bem determinadas para serem utilizadas pelas classes geradas. Com essa técnica, é necessário que o projetista do gerador saiba com antecedência quais vão ser os pontos necessários para realizar a manutenção do sistema e projete a interface dos componentes do sistema de forma a obter a maior flexibilidade possível. A quarta abordagem é uma inversão na hierarquia da terceira, ou seja, nessa abordagem as classes do usuário fazem chamadas para os métodos das classes geradas.

Zonas de segurança

Zonas de segurança (Jack Herrington, 2005) 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. Na Figura 2.10 é ilustrado o processo de geração de artefatos com zonas de segurança.

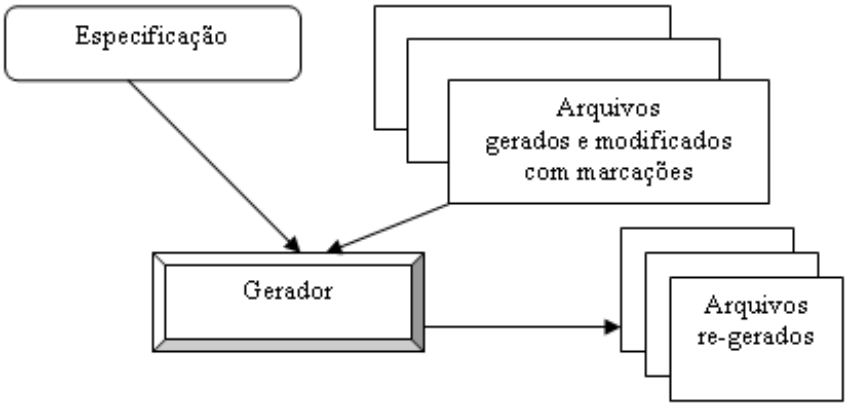

Figura 2.10: Geração de artefatos com zonas de segurança

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.

Essa abordagem permite uma grande flexibilidade porque, diferentemente dos métodos *delegate*, ela permite que o desenvolvedor insira modificações em qualquer posição dos artefatos gerados, por exemplo, em sistemas orientados a objetos, além da sobreposiçao e inserção de novos métodos e atributos nas classes geradas, o desenvolvedor pode modificar o corpo dos métodos das classes geradas.

2.4.3 Exemplos de geradores

Os geradores de aplicação podem ser classificados de duas formas diferentes (Masiero e Meira, 1993): domínio único - aplicação única (DUAU) e múltiplos domínios - múltiplas aplicações (MDMA). Os do tipo DUAU sempre criam o mesmo tipo de produto no mesmo domínio enquanto os MDMA podem ser adaptados para fornecer apoio em domínios diferentes e aplicações diferentes dentro desses domínios.

Exemplos típicos de geradores DUAU são os utilitários LEX e YACC, que geram analisadores léxicos e sintáticos, respectivamente. Um exemplo típico de gerador MDMA é o Draco (Czarnecki e Eisenercker, 2002), que possui um mecanismo para especificação de domínio e criação de regras de refinamento para transformar especificações de um domínio para outros domínios conhecidos do Draco. Diferentemente do gerador apresentado nesta dissertação, que usa uma LMA de alto nível para especificar aplicações, o Draco trabalha diretamente com transformações na árvore sintática que representa o domínio, o que o torna ao mesmo tempo muito geral e poderoso e também complexo. Por essa razão, os geradores de aplicação MDMA similares ao Draco não foram estudados e apresentados nesta Seção com maior nível de detalhes. A técnica utilizada neste trabalho para a geração de artefatos é a composição (ver Seção 2.4.1).

Diversos geradores de aplicação DUAU foram estudados para a realização deste trabalho e dentre eles, pode-se citar: o gerador apresentado por Rausch (2001), o gerador AndroMDA (Bohlen et al., 2006) e o GREN-Wizard (Braga e Masiero, 2003).

O gerador apresentado por Rausch (2001) é baseado em XML e gabaritos (do inglês *templates*). Como apresentado na Figura 2.11, ele recebe como entrada um arquivo de XML contendo uma especificação de software e um conjunto de gabaritos e, após o processamento desses gabaritos com o arquivo de XML, os arquivos de saída são produzidos. Esse gerador produz um sistema que utiliza uma biblioteca de componentes, ou seja, o gerador recebe uma especificação de XML que descreve como esses componentes devem ser parametrizados e relacionados, para em seguida com base nos gabaritos, gerar o código da aplicação.

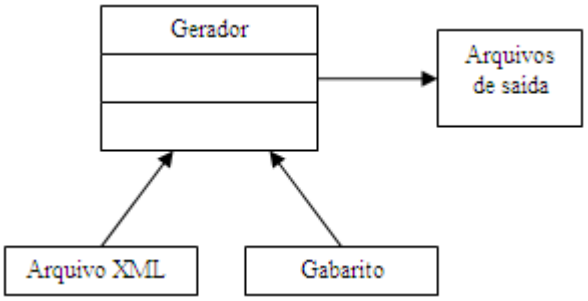

Figura 2.11: Gerador baseado em XML e gabaritos

O gerador AndroMDA (Bohlen et al., 2006) que é um gerador de código que recebe como entrada documentos UML no formato XMI (OMG, 2005) e gera aplicações para diversas pla-

taformas e tecnologias existentes, dentre elas: J2EE, Java Server Faces, Enterprise Java Beans (Java, 2006), XMLSchema (W3C, 2006), entre outros. Esse gerador utiliza as técnicas de MDA (Miller e Mukerji, 2001, 2003; Siegel, 2001) e padrões de transformação abertos gerenciados pela entidade OMG (*Object Management Group*) (OMG, 2006).

O GREN-Wizard (Braga e Masiero, 2003) é uma ferramenta para auxiliar a instanciação do framework GREN (descrito na seção 2.3.1), com base na linguagem de padrões GRN (descrita na seção 2.2.3.1). O GREN-Wizard foi projetado de forma que seus usuários possam gerar, apenas com o conhecimento sobre a GRN, aplicações específicas no domínio de gestão de recursos de negócios.

O GREN-Wizard é alimentado com informações sobre o uso da GRN na modelagem da aplicação específica e retorna como resultado o código-fonte, em Smalltalk (Goldberg, 1995), das classes especializadas a partir do GREN. Ele cria, também, as tabelas na base de dados MySQL (MySQL, 2001), de acordo com os padrões da GRN que foram utilizados. Para executar a aplicação resultante basta juntar-se ao código do GREN o código das classes geradas pelo GREN-Wizard.

A interação do engenheiro de aplicações com a GUI do GREN-Wizard é baseada nos conceitos da GRN, ou seja, deve-se informar quais padrões são usados, quais variantes são mais adequados a cada aplicação específica e quais classes desempenham cada papel no variante escolhido. Após aplicar um determinado padrão, deve-se escolher qual o padrão seguinte a aplicar, desde que se respeitem as possibilidades de navegação pré-estabelecidas pelo engenheiro de domínio que construiu a linguagem de padrões. O GREN-Wizard é utilizado paralelamente à GRN, isto é, utiliza-se a GRN para modelar o sistema, produzindo-se um modelo de análise e um histórico de padrões e variantes aplicados. Essa informação é utilizada para preencher as telas do GREN-Wizard e produzir o código Smalltalk necessário para adaptar o GREN à aplicação específica.

2.5 Engenharia de software de linhas de produtos

A engenharia de software está em constante conflito com os interesses do mercado. De um lado, a alta demanda por produtos e serviços pressiona os fornecedores de software a desenvolver sistemas de forma rápida e eficiente. Por outro lado, os clientes exigem sistemas de alta qualidade e complexidade, que não podem ser construídos de forma eficiente com os métodos tradicionais (Weiss e Lai, 1999).

Dentro desse ambiente de mercado, é observado que muitos dos sistemas desenvolvidos apresentam mais similaridades do que diferenças (Greenfield e Short, 2004). Mesmo com essa oportunidade de reúso entre aplicações, a maior parte do software desenvolvido atualmente é

construído de forma manual a partir de zero, com o uso de métodos intensivos que exigem um investimento alto, tornando o preço final do produto muito elevado (Greenfield e Short, 2004).

Por outro lado, uma organização pode utilizar métodos voltados para o desenvolvimento de uma família de produtos, ou seja, a organização pode utilizar métodos para o desenvolvimento de 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. De todas as formas, Weiss e Lai (1999) definem uma linha de produtos como um processo voltado para o desenvolvimento de uma família de produtos projetada para se obter vantagens dos seus aspectos comuns e das suas variabilidades previsíveis.

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 os 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). Com essas técnicas, uma organização pode atingir níveis mais elevados de produtividade e construir sistemas sob demanda de forma mais eficaz.

2.5.1 Processo de engenharia de software de linhas de produtos

Diversos autores (Weiss e Lai, 1999; Gomaa, 2005; Greenfield e Short, 2004) dividem o processo de desenvolvimento de engenharia de software de linhas de produtos em duas etapas. Essa divisão tem a finalidade de separar as atividades de análise, projeto e implementação genéricas de domínio das atividades de desenvolvimento de aplicações. Na primeira etapa, o engenheiro deve analisar o domínio, capturar os aspectos similares e os aspectos variáveis das aplicações, projetar e codificar artefatos para apoiar o processo de desenvolvimento de aplicações. Na segunda etapa, o engenheiro realiza a elicitação dos requisitos de aplicações dentro do domínio escolhido e desenvolve essas aplicações utilizando os artefatos criados durante a primeira etapa. As atividades genéricas do processo da engenharia de software de linhas de produtos são apresentadas na Figura 2.12.

Um domínio pode ser definido como uma área de conhecimento ou atividade caracterizada por um conjunto de conceitos e terminologia comum utilizados por especialitas nessa área (Booch et al., 2000). Um domínio pode ser associado com uma área de negócio, uma coleção de problemas ou uma coleção de aplicações (Harsu, 2002).

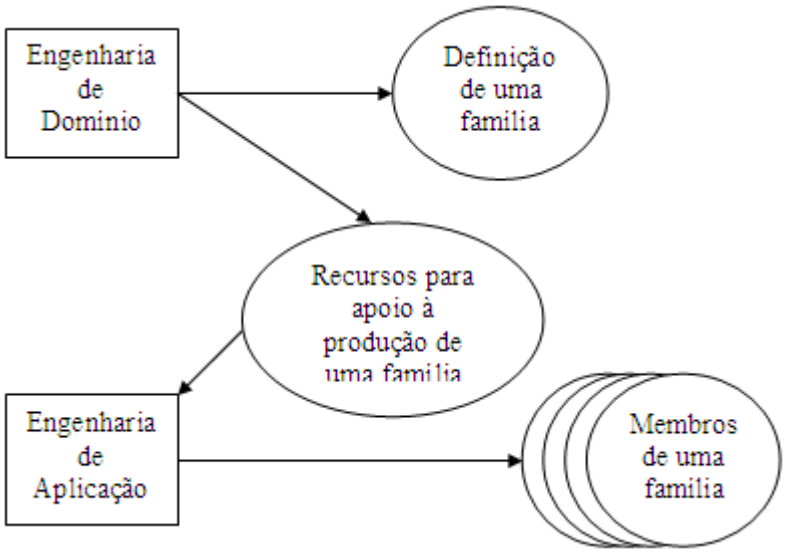

Figura 2.12: Processo de desenvolvimento de linhas de produtos (Weiss e Lai, 1999)

A primeira etapa de desenvolvimento da linha de produtos, às vezes denominada engenharia de domínio (Weiss e Lai, 1999) e outras vezes denominada desenvolvimento da linha de produtos (Greenfield e Short, 2004), é realizada para desenvolver artefatos para apoiar o reúso das aplicações semelhantes da família escolhida. Esse processo coleta, organiza e armazena experiências passadas da construção de sistemas ou partes de sistemas em um domínio particular na forma de artefatos reutilizáveis e formaliza maneiras adequadas de reutilizar esses artefatos na construção de novos sistemas.

De forma genérica, a primeira etapa compreende a análise, projeto e implementação de domínio (Czarnecki e Eisenercker, 2002; Weiss e Lai, 1999; Harsu, 2002; Greenfield e Short, 2004). A análise de domínio seleciona e define o foco do domínio, coletando as informações relevantes e integrando essas informações em um modelo denominado modelo de domínio. O projeto de domínio desenvolve uma arquitetura para uma família de sistemas em um domínio. A implementação de domínio desenvolve e refina o processo padrão de engenharia de aplicação e desenvolve o ambiente que apóia o processo de engenharia de aplicação.

Como ilustrado na Figura 2.12, a engenharia de domínio cria a definição de uma família e todos os recursos para o apoio à produção dessa família. Esses recursos podem ser compostos por padrões de processo, modelo de domínio, componentes de software, frameworks, manuais e geradores de aplicação.

Após a criação desses artefatos, uma organização estará apta a desenvolver diversos sistemas por meio da atividade denominada engenharia de aplicação. Essa atividade pode ser descrita como o processo de construção de sistemas baseado nas saídas produzidas pela engenharia de domínio (Czarnecki e Eisenercker, 2002) e nos resultados da análise de requisitos realizada de forma específica para cada sistema.

Durante a atividade de elicitação de requisitos para uma nova aplicação concreta, os modelos de domínio são utilizados para descrever as necessidades dos clientes. As necessidades que não estiverem contidas no modelo de domínio necessitam de desenvolvimento individualizado. A engenharia de aplicação compreende as fases de análise, projeto e implementação e essas atividades são desenvolvidas utilizando-se as facilidades de produção criadas nas fases de engenharia de domínio.

2.5.1.1 Processo FAST

Neste trabalho é utilizado como referência o processo de engenharia de software de linhas de produtos desenvolvido por Weiss e Lai (1999), que foi utilizado para construir um dos estudos de caso deste trabalho. Esse processo, denominado FAST, (do inglês *Family-Oriented Abstraction, Specification and Translation*), é o resultado do trabalho desses autores na criação de linhas de produtos na empresa Lucent Technologies, onde foi mostrado que a adoção desse processo pode reduzir o tempo e o custo de desenvolvimento de um membro de uma família de 60% a 80%.

A primeira atividade do processo é a qualificação de domínio. A realização dessa atividade tem o objetivo de determinar a viabilidade econômica da adoção da linha de produtos no domínio de aplicação escolhido. Se for avaliado que é vantajoso economicamente a criação da linha de produtos, o processo continua com a realização da engenharia de domínio e engenharia de aplicação.

A engenharia de domínio é realizada em um processo interativo. O resultado desse processo é a criação ou refinamento de um ambiente de engenharia de aplicação e um processo sistematizado para utilizar esse ambiente. Durante a engenharia de domínio, as atividades são divididas em duas etapas: análise de domínio e implementação de domínio.

A etapa de **análise de domínio** é realizada para se obter um conjunto de especificações para o ambiente de engenharia de aplicação e é dividida em seis atividades: 1) *definição do modelo de decisão* que consiste na definição ou refinamento de um conjunto de decisões que o engenheiro de aplicação deve resolver para descrever e construir um produto; 2) *análise de aspectos similares*: consiste na criação da análise de similaridade que define as fronteiras da família. Essa atividade consiste nas seguintes sub-atividades: definição de uma terminologia padrão, definição das similaridades e variabilidades que um produto da família pode apresentar e parametrização das variabilidades, ou seja, a definição dos parâmetros de variação que quantificam as variabilidades de um membro da família. Cada parâmetro é definido por um nome e pode assumir um valor diferente para cada membro da família criado; 3) *projeto de domínio*: consiste na criação ou refinamento de um projeto comum para todos os membros da família e na criação de um mapeamento entre a linguagem de modelagem de aplicações e os componentes especificados no projeto; 4) *projeto da linguagem de modelagem de aplicações*

(AML, do inglês *Application Modelling Language*), que é uma linguagem utilizada para especificar membros de uma família. Essa linguagem deve permitir que o engenheiro expresse os parâmetros de variação definidos durante a análise de similaridades. 5) criação do *processo de engenharia de aplicação padrão*: realizado para definir como as decisões realizadas pelos engenheiros de aplicação são incorporadas em um processo padrão para o domínio, em particular, o processo deve definir como essas decisões são apoiadas por um conjunto de ferramentas que vão constituir o ambiente de engenharia de aplicação; 6) *projeto do ambiente de engenharia de aplicação*: consiste em um conjunto de ferramentas que são utilizadas para analisar os modelos da aplicação e gerar o código com base nesses documentos.

As atividades da etapa de **implementação de domínio** são realizadas com o objetivo de construir o ambiente de engenharia de aplicação para o domínio. Esse ambiente é construído com base nas especificações criadas durante a análise de domínio ou modelos de domínio. A implementação de domínio está dividida nas seguintes atividades: 1) *implementação do ambiente de engenharia de aplicação*, realizada com o objetivo de desenvolver ferramentas para o ambiente de engenharia de aplicação. Essa atividade pode ser dividida em: implementação de uma biblioteca de gabaritos, implementação de ferramentas de geração e implementação de ferramentas de análise; 2) *documentação do ambiente de engenharia de aplicação*, em que é criada toda a documentação necessária para manter e utilizar o ambiente de engenharia de aplicação. Essa documentação pode conter o manual do usuário para as ferramentas e o material de treinamento e documentação necessária para mudar o ambiente assim que a família evoluir.

Após a realização das atividades da engenharia de domínio e criação de todos os artefatos necessários (os modelos de domínio), a organização pode realizar o processo de engenharia de aplicações. A engenharia de aplicação é um processo interativo utilizado para construir aplicações para satisfazer as necessidades dos clientes. Cada aplicação é considerada um membro de uma família e é definida pelo modelo da aplicação. O ambiente de engenharia de aplicação é utilizado para analisar as aplicações e gerar código e documentação. As atividades do processo de engenharia de aplicação são: modelagem da aplicação, produção da aplicação e fornecimento de apoio de entrega e operação.

A atividade de *modelagem de aplicações* consiste na modelagem das necessidades dos clientes para uma aplicação utilizando especificações no formato da linguagem de modelagem de aplicações. O engenheiro de aplicação pode analisar o modelo utilizando as ferramentas do ambiente de engenharia de aplicação e gerar código e documentos com base nesse modelo. A *produção da aplicação* é a atividade na qual o código e a documentação são gerados com base no modelo da aplicação. Se a aplicação não pode ser totalmente gerada, algumas partes devem ser desenvolvidas manualmente. O *fornecimento de apoio de entrega e operação* consiste na entrega da aplicação para o cliente e fornecimento de apoio operacional para o sistema.

Durante o processo FAST, diversos papéis realizam as etapas descritas. De forma geral, os engenheiros de domínio são responsáveis pela construção e evolução da família, por garantir que o investimento financeiro na família continue a valer a pena e os seus esforços são distribuídos para apoiar o projeto e desenvolvimento de aplicações dessa família. Os engenheiros de aplicação são responsáveis por produzir membros da família para satisfazer o contrato com os clientes e normalmente os seus esforços são direcionados para projetos individuais.

2.6 Considerações finais

Foram apresentados as principais formas de reúso utilizadas neste trabalho. Os padrões de software permitem reutilizar o conhecimento dos desenvolvedores experientes de soluções recorrentes utilizadas em projetos passados em novos projetos. Os frameworks de software orientados a objetos são utilizados para reutilizar super-estruturas arquiteturais e permitem o reúso de grandes parcelas de código em uma aplicação.

Mesmo utilizando as formas de reúso apresentadas, uma organização deve realizar muito trabalho manual configurando e compondo os artefatos reutilizáveis em uma aplicação completa. Esse trabalho representa uma parcela significativa do trabalho realizado e muitas vezes é realizado manualmente de forma repetitiva. Essas tarefas repetitivas podem ser automatizadas por ferramentas de geração que recebem uma especificação de alto nível de abstração e realizam essas tarefas automaticamente.

As três primeiras formas de reúso podem ser aplicadas em projetos de software que utilizam processos de desenvolvimento tradicionais, tais como os orientados a objetos ou métodos baseados em componentes, e os artefatos reutilizados são normalmente compostos para se obter uma aplicação completa. Por outro lado, o processo definido na engenharia de software de linhas de produtos permite a reutilização de artefatos em um domínio de aplicação, permitindo a construção de membros de uma família de aplicação com menor esforço e consequentemente menor custo para uma organização.

A engenhenharia de software de linhas de produtos se diferencia dos métodos convencionais, porque é utilizada na construção de famílias de aplicações e normalmente utiliza modelos na geração de aplicações, tais como o realizado pela arquitetura dirigida a modelos (do inglês *Model Driven Architecture*) (Miller e Mukerji, 2001), uso de linguagens específicas de domínio (Deursen et al., 2000), uso de padrões de projeto e arquiteturais para a transformação dos modelos, uso sistemático do reúso (Greenfield e Short, 2004; Gomaa, 2005) e principalmente, o foco na mudança de paradigma, da construção de produtos individuais para a construção de uma família de produtos.

A industrialização dos meios de produção de diversos produtos já foi alcançada por outras disciplinas de engenharia. Desde o início da revolução industrial, com a criação de máquinas

de tosquiar lã até as mais novas e robotizadas fábricas de construção automobilística, diversos dos conceitos sobre linhas de produtos foram aplicados com sucesso.

Alguns autores (Gomaa, 2005; Weiss e Lai, 1999; Greenfield e Short, 2004; Días-Herrera, 2000) apontam a engenharia de software de linhas de produtos como uma técnica excelente para uma organização alcançar o reúso sitemático e realizar o desenvolvimento de membros de uma família de aplicações em menos tempo e com menor custo do que os métodos de engenharia de software tradicionais.

É importante notar que o desenvolvimento de software tradicional representa uma parcela significativa do mercado, uma vez que muitos dos produtos de software desenvolvidos atualmente são, por natureza, feitos sob medida para clientes especifícos e domínios específicos. Porém, a construção de todos os sistemas que são reproduzidos diversas vezes dentro de um mesmo domínio, pode sofrer uma mudança de paradigma com a utilização de diversos dos conceitos sobre linhas de produtos. O uso desses conceitos pode elevar a produtividade e aumentar o poder competitivo das organizações que disputam determinados segmentos de mercados.

CAPÍTULO

3

Geradores de aplicação configuráveis

3.1 Considerações iniciais

Como foi apresentado na Seção 2.4, os geradores de aplicação são ferramentas que aceitam uma especificação de aplicações de software, validam essa especificação e geram artefatos. Em geral essas ferramentas produzem artefatos para um domínio específico e, conforme mencionado no Capítulo 1, elas podem ser custosas e complexas de desenvolver, o que pode não justificar sua construção para domínios em que o custo de desenvolver um gerador, somado ao custo de desenvolver aplicações com esse gerador, seja maior do que o custo de desenvolver as mesmas aplicações sem o uso do gerador.

Por outro lado, os geradores de aplicação configuráveis são geradores que podem ser configurados para fornecer apoio em domínios diferentes[1]. A principal vantagem dessa abordagem em relação à primeira é que as atividades de configuração de um gerador de aplicação configurável podem ser menos custosas do que a construção completa de um gerador de aplicação específico, diminundo o custo de uso do gerador e permitindo a utilização das técnicas de geração para domínios que normalmente não apresentam viabilidade econômica para investir nessas ferramentas.

[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), e também para denotar famílias de aplicações, que são definidas como conjuntos de itens que possuem aspectos comuns e variabilidades previsíveis (Weiss e Lai, 1999).

Este Capítulo apresenta os geradores de aplicação configuráveis e está organizado da seguinte forma: na Seção 3.2 é apresentada uma visão geral dos geradores de aplicação e das etapas de seu ciclo de vida, na Seção 3.3 é apresentada uma visão geral da etapa de desenvolvimento dos geradores configuráveis, na Seção 3.4 é apresentado o processo de engenharia de domínio utilizado para se obter a configuração do gerador configurável, na Seção 3.5 é apresentado o processo de engenharia de aplicação utilizado para produzir aplicações com o gerador configurável, finalmente, na Seção 3.6, são apresentadas as considerações finais deste Capítulo.

3.2 Visão geral dos geradores de aplicação configuráveis

Um gerador de aplicação configurável (GAC) é uma ferramenta que deve ser configurada para apresentar as mesmas características de um gerador de aplicação específico, ou seja, receber uma especificação, armazená-la em meio persistente, permitir a edição e re-edição dessa especificação, validá-la e transformá-la em artefatos de software.

O ciclo de vida de um gerador de aplicação configurável possui basicamente três etapas: desenvolvimento, engenharia de domínio e engenharia de aplicação. Na etapa de desenvolvimento, o gerador de aplicação configurável é projetado e implementado; na etapa de engenharia de domínio, um domínio de aplicação é analisado, 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) (ver Seção 3.4.1.4) para expressar aplicações nesse domínio e o GAC é configurado para produzir automaticamente aplicações; finalmente, na etapa de engenharia de aplicação, o GAC configurado é utilizado para a geração de aplicações[2].

Os GAC's podem, após o seu desenvolvimento, ser configurados para produzir artefatos em diversos domínios, e cada instância configurada do GAC pode produzir artefatos para diversas aplicações. Como exemplo, ilustra-se na Figura 3.1 o desenvolvimento de um GAC, a configuração desse GAC nas instâncias G1, G2 e GN e a utilização desse GAC na produção de diversas aplicações. A instância G1 pode produzir artefatos para as aplicações T1, T2 e Tn no domínio T, a instância G2 pode produzir artefatos para as aplicações P1, P2 e Pn no domínio P, e assim sucessivamente em cada domínio no qual o gerador é configurado.

Quando uma organização tem à disposição um GAC e precisa desenvolver um projeto de software em um domínio específico, ela deve avaliar a possibilidade de utilizar o GAC conforme o processo apresentado no diagrama de atividades da Figura 3.2. Na primeira atividade, deve-se

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

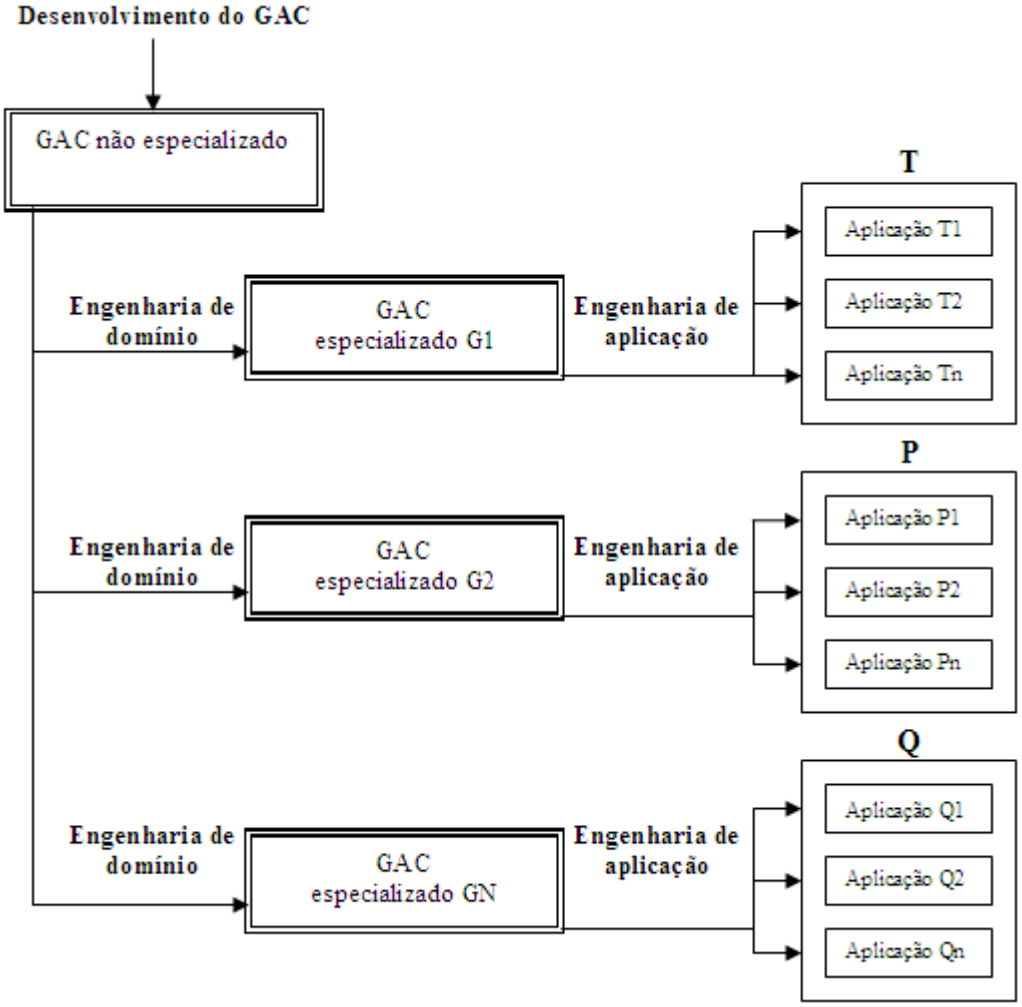

Figura 3.1: Exemplo do ciclo de vida de um gerador configurável

verificar se o GAC já fornece apoio para o domínio escolhido. Em caso positivo, a organização pode utilizar o GAC para gerar artefatos nesse domínio. Se o GAC apoiar parcialmente o domínio escolhido, a organização deve verificar a viabilidade técnica de adaptar o GAC nesse domínio ou utilizar o GAC parcialmente configurado no desenvolvimento de aplicações nesse domínio. Para realizar a atividade de adaptação do GAC, a organização deve verificar se todas as atividades da etapa de engenharia de domínio e aplicação (apresentadas nas Seções 3.4 e 3.5, respectivamente) podem ser realizadas. Em caso positivo, deve ser realizada a estimativa do custo da adaptação da ferramenta e a qualificação do domínio. A estimativa do custo da adaptação deve calcular aproximadamente o esforço necessário para alterar a configuração da ferramenta para ela fornecer apoio ao desenvolvimento da aplicação no domínio escolhido e a qualificação de domínio deve ser realizada para verificar a viabilidade econômica da utilização de GAC's. Na qualificação de domínio, a organização deve estimar quantas aplicações serão desenvolvidas após a configuração do GAC no domínio escolhido e se o custo de desenvolver essas aplicações com o GAC somado ao custo de adaptar o GAC é menor do que o custo

de desenvolver as mesmas aplicações sem o GAC. Em caso positivo, deve-se adaptar o GAC e utiliza-lo na geração de artefatos no domínio escolhido, caso contrário, a organização não deve utilizar o GAC. Para realizar a atividade de utilização do GAC parcialmente configurado, o engenheiro de aplicação deve utilizar o GAC para gerar alguns artefatos e implementar manualmente o restante dos artefatos necessários para a construção da aplicação. Se o GAC não fornecer apoio ao domínio escolhido, a organização deve estimar o custo de configurar o GAC e qualificar o domínio. Caso o domínio seja qualificado de forma positiva, a organização deve configurar e utilizar o GAC na produção de artefatos para o domínio escolhido, caso contrário não deve utilizá-lo.

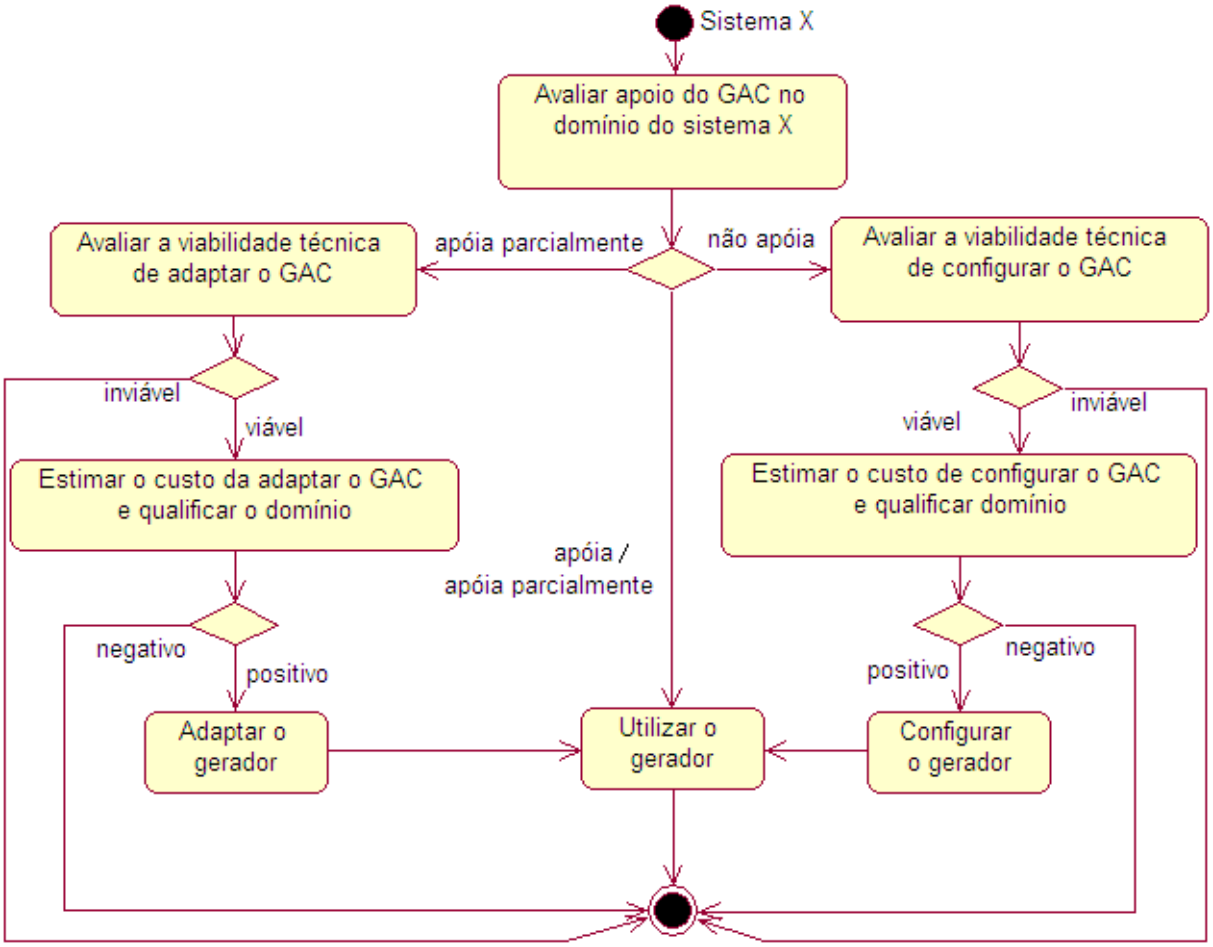

Figura 3.2: Avaliação do gerador configurável

Os principais papéis envolvidos no processo de desenvolvimento de software com GAC's são o engenheiro de domínio e o engenheiro de aplicação. O engenheiro de domínio é responsável por desenvolver e configurar o GAC e o engenheiro de aplicação é responsável por utilizar o GAC configurado durante o processo de desenvolvimento de aplicações de software.

3.3 Visão geral da etapa de desenvolvimento

Após a análise de diversos geradores de aplicação específicos, pode-se notar que essas ferramentas são modularizadas de forma muito semelhante. Alguns desses módulos são encontrados em todos os geradores analisados e outros são encontrados apenas em alguns deles. A seguir, apresenta-se uma lista dos principais módulos dos geradores de aplicação:

1. Módulo de interface com o usuário.
2. Módulo de validação da especificação.
3. Módulo de geração de artefatos.
4. Módulo de gerenciamento de projetos.
5. Módulo de persistência da especificação.
6. Módulo de apoio na atividade de depuração da especificação.
7. Módulo de zonas de segurança.

O módulo de **interface com o usuário** tem a responsabilidade de apresentar uma interface que permita que o engenheiro de aplicação disponibilize para o gerador uma especificação de software utilizando uma linguagem de modelagem de aplicação (LMA). Esse módulo pode ser implementado de diversas formas, dentre elas pode-se citar: um programa que aceita na linha de comando argumentos que representam os dados da especificação (ou o caminho do arquivo que contém a especificação), um editor de gráficos em que seja possível desenhar, manipular e relacionar conceitos de um domínio, uma interface em que seja possível inserir diagramas e documentos (por exemplo, diagramas UML) ou uma interface composta por formulários com elementos gráficos. O módulo de **validação da especificação** tem a responsabilidade de analisar a especificação e verificar a sua corretitude. O módulo de **geração de artefatos** tem a responsabilidade de gerar artefatos com base na especificação. O módulo de **gerenciamento de projetos** tem a responsabilidade de gerenciar os dados de um ou mais projetos criados com o gerador. Esses dados podem conter a localização dos caminhos dos diretórios da especificação e dos arquivos gerados, a especificação, os arquivos gerados e propriedades específicas. O módulo de **persistência da especificação** tem a responsabilidade de realizar a persistência dos dados da especificação das aplicações (ou instância da LMA) com o objetivo de permitir que o engenheiro de aplicação possa reeditar essa especificação e regerar os artefatos dessa especificação em um momento posterior. O módulo de **apoio na atividade de depuração da especificação** tem a responsabilidade de ajudar o engenheiro de aplicação a localizar possíveis

erros contidos na especificação. O módulo de **zonas de segurança** (ver Seção 2.4.2) tem a responsabilidade de, durante a regeração de artefatos, verificar modificações manuais inseridas nos artefatos gerados e inserir essas modificações nos artefatos regerados.

Como geradores diferentes recebem e validam especificações utilizando LMA's em formatos diferentes gerando artefatos de maneiras diferentes, os engenheiros devem utilizar técnicas diferentes para projetar e implementar os módulos de geradores específicos. Por outro lado, se um grupo de geradores que apoiam domínios distintos utilizar o mesmo formato para representar as suas LMA's e o mesmo conjunto de técnicas para validá-las e gerar artefatos, alguns módulos comuns podem ser reutilizados por esses geradores e a diferença entre eles serão apenas os módulos específicos do domínio.

De maneira geral, os módulos comuns que podem ser reutilizados em geradores de domínios diferentes são: gerenciamento de projetos, persistência da especificação, apoio na atividade de depuração da especificação e zonas de segurança. Os módulos que são variáveis no domínio são: interface com o usuário, validação da especificação e geração de artefatos.

Para realizar a substituição de módulos específicos de domínio, o engenheiro pode reconstruir os módulos do zero ou pode construir um framework para instanciar esses módulos com mais facilidade. De qualquer forma, a substituição dos módulos específicos de domínio para a obtenção de geradores que apoiam domínios diferentes é uma atividade que pode reduzir o custo de implementação de um gerador, porque reutiliza os módulos comuns e o conhecimento das técnicas de implementação dos módulos específicos de domínio.

Embora a substituição dos módulos específicos de domínio possa reduzir o custo da obtenção de um gerador específico, o engenheiro de domínio ainda tem que realizar algumas atividades custosas, tais como entender o projeto modular e implementar os módulos específicos de domínio. Para facilitar ainda mais o trabalho de "especialização" de um gerador para um domínio específico, o engenheiro de domínio pode projetar esses módulos de forma que eles possam ser configurados para realizar as suas funcionalidades específicas, ou seja, projetar esses módulos de forma que eles executem as suas funcionalidades de acordo com diretivas contidas em arquivos de configuração. Com essa abordagem, as atividades de implementação e substituição dos módulos específicos de domínio são substituídas pela atividade de configuração de módulos configuráveis. Os módulos configuráveis utilizados neste trabalho são detalhados na Seção 3.4 e as técnicas de implementação desses módulos são detalhadas na Seção 4.2.

A etapa de desenvolvimento de geradores configuráveis pode ser realizada de forma interativa, incremental, prototipação, cascata ou qualquer outro modelo de processo que o engenheiro de domínio escolher e, basicamente, deve conter as seguintes atividades: análise, projeto, implementação e testes. Na atividade de **análise**, o engenheiro de domínio deve avaliar vários geradores de aplicação, analisar os aspectos similares, analisar os aspectos variáveis e criar um documento com a visão geral do gerador: a avaliação de vários geradores, a análise dos aspectos

similares e dos aspectos variáveis devem ser realizadas de uma forma parecida com a que foi apresentada nesta Seção. O documento de visão geral deve apresentar brevemente as principais funcionalidades que estarão disponíveis para os usuários, os principais módulos e o processo configuração e utilização do gerador. Na atividade de **projeto**, o engenheiro de domínio deve definir a arquitetura, criar a estrutura modular, definir quais módulos serão configuráveis e como eles serão configurados. Na atividade de **implementação**, o gerador é codificado e, finalmente, na atividade de **testes**, o gerador é testado e validado.

Dependendo do formato de LMA que o GAC aceita, da forma como ele valida essa LMA e da técnica utilizada na geração de artefatos, o GAC deve ser projetado e implementado de maneiras diferentes. Por esse motivo, as atividades da etapa de desenvolvimento dos geradores configuráveis não são detalhadas nesta Seção. No Capítulo 4, é apresentado o gerador de aplicação configurável Captor e o detalhamento da sua etapa de desenvolvimento.

3.4 Engenharia de domínio

A realização da etapa de engenharia de domínio proposta nesta Seção tem o objetivo de criar todos os artefatos necessários para configurar o GAC para apoiar um domínio específico. Nessa etapa, o engenheiro de domínio deve realizar as seguintes atividades de domínio: análise, projeto e implementação, configuração do GAC e documentação. Nas Seções seguintes, as atividades da engenharia de domínio são detalhadas.

3.4.1 Análise de domínio

A análise de domínio é definida como o processo em que a informação utilizada no desenvolvimento de sistemas de software é identificada, capturada e organizada com o objetivo de fazê-la reusável durante a criação de novos sistemas (Prieto-Diaz, 1990). A análise de domínio pode ser aplicável a vários paradigmas de engenharia de software e, atualmente, existem diversas técnicas descritas na literatura para realizá-la (Pressman, 2001; Czarnecki e Eisenercker, 2002; Weiss e Lai, 1999; Prieto-Diaz, 1990; Valerio et al., 1997; Greenfield e Short, 2004).

Neste trabalho não é criado nem utilizado como referência nenhum método de análise de domínio específico. O engenheiro de domínio deve notar que, independentemente do método escolhido para realizar essa atividade, as principais sub-atividades que devem ser relizadas são: descrição do domínio, definição da terminologia comum, análise de aspectos similares e variáveis e criação da linguagem de modelagem de aplicações.

A **descrição do domínio** é utilizada para fornecer uma visão geral do domínio que está sendo analisado. A **definição de uma terminologia comum** serve para refinar e ampliar a comunicação de idéias durante o processo. A **análise de aspectos similares e variáveis** é

realizada para estabelecer os aspectos similares de todas as aplicações construídas no domínio e elicitar as variabilidades que essas aplicações devem apresentar entre si. Finalmente, a **criação da linguagem de modelagem de aplicações** (LMA) tem o objetivo de produzir uma linguagem de alto nível de abstração que é utilizada para descrever aplicações no domínio escolhido.

Nas Seções a seguir, as sub-atividades da análise de domínio e o restante das atividades da engenharia de domínio são exemplificadas no domínio de persistência de dados ou, mais especificamente, mapeamento de objetos para o sistema relacional[3].

3.4.1.1 Descrição do domínio

A descrição do domínio fornece uma visão geral do domínio que está sendo analisado. Essa descrição é realizada no inicío da análise de domínio e é utilizada para estabelecer as fronteiras gerais do domínio analisado. A descrição do domínio de persistência é apresentada a seguir:

Descrição do domínio de persistência

Muitas aplicações utilizam a persistência de dados para armazenar informações em um dispositivo de *hardware* com a finalidade de recuperar essas informações em um momento posterior. No mapeamento de objetos para o sistema relacional, uma classe representa uma tabela do banco de dados e o conjunto de atributos dessa classe representa uma tupla dessa tabela. Adicionalmente, as classes persistentes devem fornecer métodos acessores e modificadores para manipular os atributos e métodos adicionais para manipular os dados na base, tais como métodos para inserção, alteração, remoção e busca de objetos.

3.4.1.2 Definição da terminologia comum

A definição de uma terminologia comum tem o objetivo de refinar e ampliar a comunicação de idéias durante o processo. A terminologia comum para o domínio de persistência é apresentada na Tabela 3.1.

Tabela 3.1: Terminologia comum do domínio de persistência

Termo:	**Significado**
Entidade	Uma “coisa” ou “objeto” do mundo real que pode ser identificada de forma unívoca em relação a todos os outros objetos (Silberschatz et al., 1999).
Tabela	Uma estrutura de dados que representa uma entidade no sistema gerenciador de banco de dados relacional.
Classe persistente	Uma classe que manipula os dados de uma tabela.

[3]Este domínio foi escolhido por ser bem simples e conhecido.

3.4.1.3 Análise de aspectos similares e variáveis

Um domínio pode ser representado por aplicações que possuem um conjunto de itens comuns e variabilidades previsíveis. Um aspecto comum é uma característica comum para todas as aplicações do domínio e uma variabilidade é uma característica que pode diferenciar uma aplicação de outra no domínio analisado.

Os aspectos comuns e variáveis do domínio de persistência são apresentados nas Tabelas 3.2 e 3.3, respectivamente.

Tabela 3.2: Aspectos similares

Aspecto similar	**Descrição**
Métodos acessores	Todas as classes persistentes possuem métodos de acesso aos seus atributos no formato get*NomeDoAtributo*() em que a cadeia de caracteres *NomeDoAtributo* deve ser substituída pelo nome do atributo.
Métodos modificadores	Todas as classes persistentes possuem métodos modificadores de seus atributos no formato set*NomeDoAtributo*(*tipo parâmetro*) em que a cadeia de caracteres *NomeDoAtributo* deve ser substituída pelo nome do atributo e a cadeia de caracteres *tipo* deve ser substituída pelo tipo do atributo.
Conexão com a base de dados	Todas as classes devem ter um atributo do tipo *Connection* que deve ser instanciado no construtor da classe.

Tabela 3.3: Variabilidades identificadas

Variabilidade	**Descrição**
Nome da tabela	O nome da tabela do banco de dados que possui uma classe persistente relacionada.
Nome da classe	O nome da classe persistente que representa uma tabela na base de dados.
Atributos da classe	O nome dos atributos da classe persistente.
Atributos da tabela	O nome dos atributos da tabela do banco de dados.
Tipos dos atributos	O tipo dos atributos da classe e da tabela.

3.4.1.4 Criação da linguagem de modelagem de aplicações

A criação da LMA tem o objetivo de produzir uma linguagem de alto nível de abstração, que é utilizada para descrever aplicações no domínio escolhido. Instâncias dessa linguagem (ou especificações) devem ser inseridas no gerador por um engenheiro de aplicação, o gerador deve validar essa instância baseado em regras específicas e gerar artefatos com base na instância validada e um conjunto de gabaritos[4].

[4]O desenvolvimento de gabaritos é apresentado na Seção 3.4.4.1

As LMAs podem ser definidas por uma gramática formal, assim como as linguagens de programação, ou podem ser definidas de outras formas, tais como: um conjunto de gráficos que podem ser reunidos para representar uma especificação, um conjunto de parâmetros que definem as partes variáveis de uma aplicação ou telas gráficas em que o usuário insere um conjunto de dados e esses dados são transformados para um formato apropriado.

Para representar aplicações no domínio de persistência, pode-se criar uma linguagem declarativa que permite que o engenheiro de aplicação especifique quais são os artefatos (classes, interfaces e *scripts* de criação de tabelas) que devem ser criados para representar os dados armazenados no banco relacional. Essa linguagem pode ser descrita de diversas formas: a) representação por meio de formulários, conforme modelo apresentado na Figura 3.3; b) como um diagrama, conforme apresentado na Figura 3.4; ou c) no formato XML, conforme apresentado na Figura 3.5.

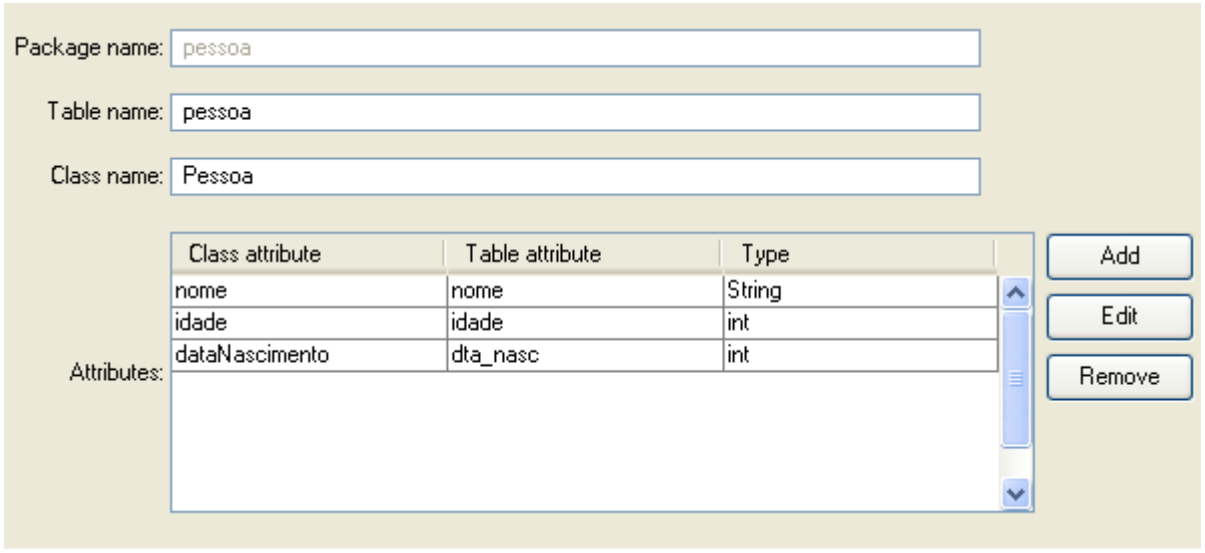

Figura 3.3: Representação da LMA por meio de formulários

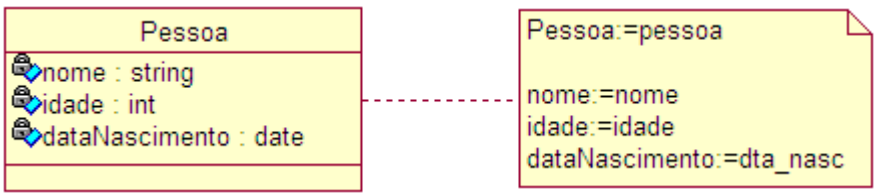

Figura 3.4: LMA no formato de um diagrama UML

Os exemplos das Figuras 3.3, 3.4 e 3.5 podem ser utilizados para a geração dos mesmos artefatos porque eles possuem o mesmo meta-modelo, ou seja, todas as especificações possuem componentes que representam uma classe, seus atributos e a tabela na qual ela está relacionada.

```
<persistencia>

  <entidade>
    <nome_tabela>pessoa</nome_tabela>
    <nome_classe>Pessoa</nome_classe>
    <atributos>
      <atributo tipo="string">
        <att_tabela>nome</att_tabela>
        <att_classe>nome</att_classe>
      </atributo>
      <atributo tipo="int">
        <att_tabela>idade</att_tabela>
        <att_classe>idade</att_classe>
      </atributo>
      <atributo tipo="date">
        <att_tabela>dta_nasc</att_tabela>
        <att_classe>dataNascimento</att_classe>
      </atributo>
    </atributos>
  </entidade>

</persistencia>
```

Figura 3.5: LMA no formato do XML

3.4.2 Projeto de domínio

Normalmente, o código das aplicações pode ser dividido entre os artefatos reutilizáveis, tais como componentes ou frameworks, e artefatos específicos da aplicação, tais como: classes, interfaces e arquivos de configuração. Durante o projeto de domínio, o engenheiro de domínio deve descrever os artefatos que serão construídos para serem reutilizados durante o desenvolvimento de aplicações, deve definir uma arquitetura comum na qual esses artefatos serão utilizados e como as classes da aplicação devem utilizar os artefatos reutilizáveis nessa arquitetura.

Atualmente, existem diversas técnicas descritas na literatura para a realização do projeto de software (Larman, 2004; Daniels e Cheesman, 2000; Booch et al., 1999; Pressman, 2001; D'Souza e Wills, 1999; Braga, 2003; Weiss e Lai, 1999; Czarnecki e Eisenercker, 2002; Greenfield e Short, 2004). Neste trabalho não é criada e nem utilizada como referência nenhuma técnica de projeto específica. O engenheiro de domínio tem a liberdade de escolher a técnica ou método de projeto com o qual estiver mais familiarizado ou que achar mais conveniente para utilizar no domínio escolhido. As únicas recomendações deste trabalho com relação ao projeto dos artefatos são que sejam respeitados os conceitos de independência funcional, alta coesão e baixo acoplamento.

No domínio de persistência, durante o projeto, foi definido que cada entidade é representada por uma classe que deve conter os métodos acessores, modificadores, insert(), delete(), update()

e getByPrimaryKey(argumento) e uma interface com os métodos persistentes. O nome da classe persistente deve ser o mesmo da entidade que a classe representa, sempre iniciando com letras maiúsculas e, o nome da interface deve ser o mesmo nome da classe prefixado com ´´I".

Todas as classes persistentes precisam ter uma associação com a classe PConnection que realiza a conexão, autenticação e executa operações na base de dados. Essa classe deve ser projetada como um componente de software e deve fornecer uma interface com o banco de dados para os seus clientes. No seu construtor, ela recebe como argumento o caminho de diretório de um arquivo de texto que contém a URL da base de dados, o número da porta na qual o banco de dados aguarda conexões, o usuário e a senha. O diagrama de classes que representa as classes persistentes e o componente de conexão é exemplificado na Figura 3.6.

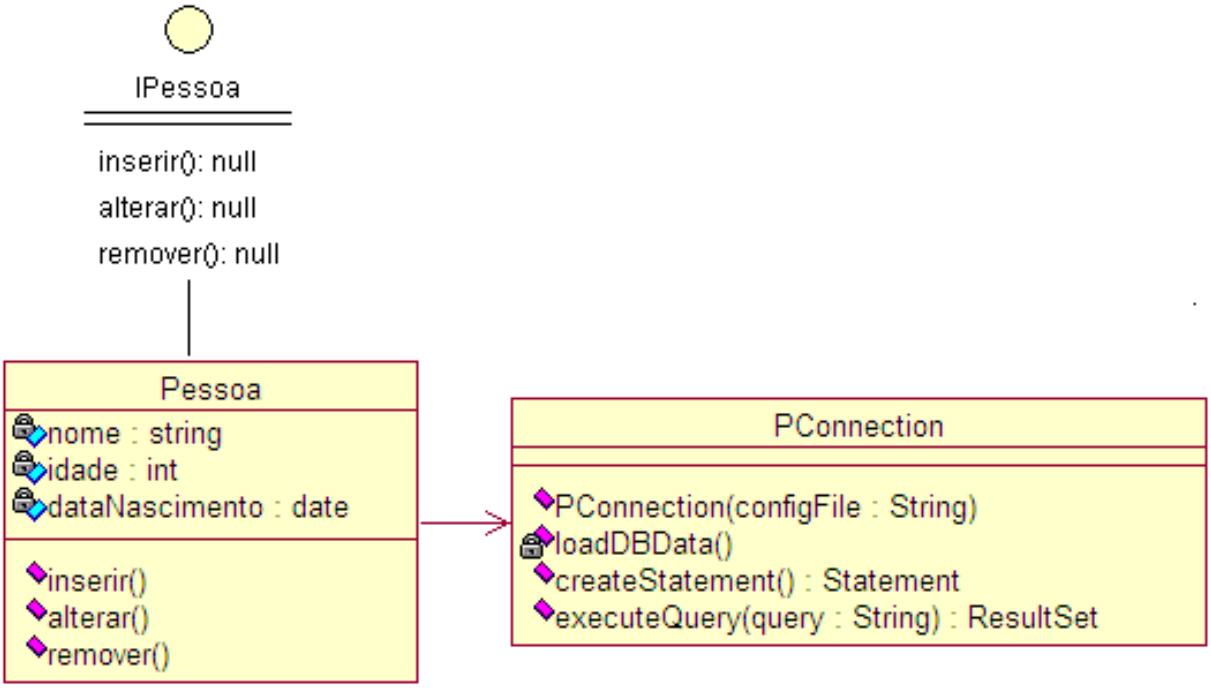

Figura 3.6: Artefatos do domínio de persistência

As classes persistentes devem ser executadas em uma arquitetura distribuída, similar à arquitetura apresentada na Figura 3.7, em que as classes persistentes de uma aplicação acessam o banco de dados por meio de conexões de rede.

3.4.3 Implementação de domínio

Durante a atividade de implementação de domínio, os artefatos reutilizáveis definidos na etapa de projeto de domínio são codificados e testados. No caso do domínio de persistência, é implementada a classe PConnection. Os demais artefatos serão gerados automaticamente por meio dos gabaritos fornecidos na etapa seguinte, que criarão as classes, interfaces e scripts da base de dados.

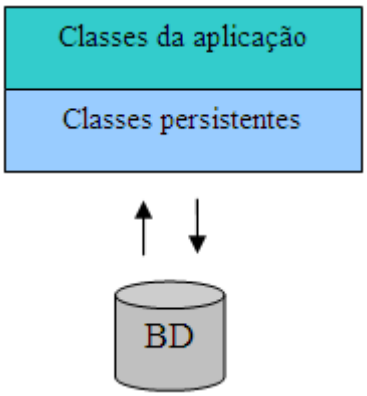

Figura 3.7: Arquitetura do domínio de persistência de dados

3.4.4 Configuração da ferramenta

Na atividade de configuração do GAC, o engenheiro de domínio deve criar um conjunto de arquivos de configuração e gabaritos que são utilizados pelo gerador para apresentar comportamento variável específico do domínio, ou seja, o GAC deve ser capaz de receber uma LMA, validar essa LMA e gerar artefatos com base nessa LMA e nos gabaritos.

O gerador deve possuir uma interface com o usuário configurável para que seja possível que o gerador possa receber LMA's de diferentes domínios. A configuração da interface do gerador é uma atividade dependente do tipo de LMA para a qual o GAC fornece apoio e de sua implementação e, por esse motivo, essa atividade não é detalhada nesta Seção. A configuração da interface gráfica de um GAC específico é apresentada na Seção 4.3.1.4.

A configuração da validação da LMA deve ser realizada de forma que o GAC possa checar a corretitude e a consistência de uma especificação. Tipicamente, o GAC deve ler um conjunto de regras e verificar se essas regras são respeitadas na especificação que o usuário inseriu no GAC. A configuração das regras de validação de uma especificação é uma atividade dependente do tipo da LMA usada e da implementação do GAC e, por esse motivo, essa atividade não é detalhada nesta Seção. A configuração das regras de validação de um GAC específico é apresentada na Seção 4.3.4.1.

A transformação da especificação em artefatos pode ser realizada por meio de compilação ou composição (Weiss e Lai, 1999). A configuração da transformação por meio de uma interface configurável de compilação é uma tarefa complexa, porque expõe os detalhes das técnicas de compilação, tais como análise léxica, sintática ou semântica para o engenheiro de domínio e, por este motivo, essa abordagem não será utilizada neste trabalho. Uma interface de configuração de transformação por meio de composição requer que o gerador disponibilize os dados das instâncias da LMA no formato apropriado e um motor de transformação interno utilize um conjunto de gabaritos para obter a geração de artefatos. O processo de desenvolvimento de

gabaritos é detalhado na Seção 3.4.4.1 e o mapeamento da LMA nos gabaritos é detalhado na Seção 3.4.4.2.

3.4.4.1 Desenvolvimento de gabaritos

Os gabaritos são documentos que contém o conteúdo fixo dos artefatos que devem ser gerados e marcações especiais que são substituídas durante o processo de geração. Como exemplo, na Figura 3.8, são apresentados um arquivo de XML representando parte de uma especificação de software (indicado pela letra a), o trecho de um gabarito (indicado pela letra b) e o resultado do processamento dessa especificação com o gabarito (indicada pela letra c). O processador de gabaritos do gerador deve manter o texto fixo contido no gabarito e substituir as marcações especiais por dados contidos na especificação.

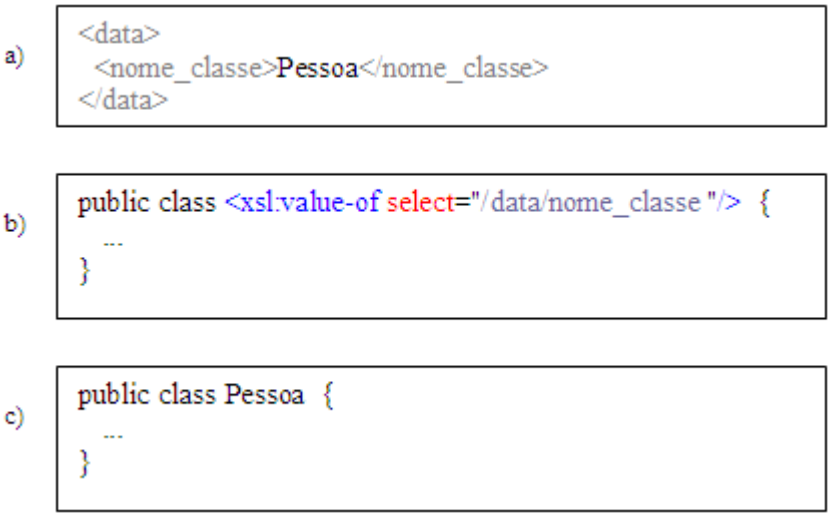

Figura 3.8: Exemplo simplificado de transformação de gabaritos

Para implementar os gabaritos em um gerador de aplicação, o engenheiro de domínio tem três alternativas: implementar os gabaritos diretamente no código do gerador, criar uma linguagem de gabaritos ou utilizar uma linguagem de gabaritos comercial.

A implementação direta no código, ilustrada na Figura 3.9 e descrita por Braga (2003) e Pazin e Penteado (2004), requer a criação de um método-gabarito para cada elemento dos artefatos que devem ser gerados. Essa abordagem tem a vantagem de ser mais fácil de desenvolver, porque o engenheiro de domínio não precisa aprender ou criar uma nova linguagem como nos outros casos, porém dificulta a alteração da saída da geração, porque acopla os gabaritos ao gerador.

A criação de uma nova linguagem de gabaritos, como realizado por Jack Herrington (2005) e Rausch (2001), é uma abordagem mais genérica do que a implementação direta no código, porque desacopla os gabaritos do gerador. Nessa abordagem, o engenheiro de domínio define

```
public void printClassDec(String classname)  {
  out.println("public class " + classname + " {…}");
}
```

Figura 3.9: Exemplo de gabarito inserido no código do gerador de aplicação

uma nova linguagem que deve conter os elementos-padrão de linguagens de gabaritos, tais como: cláusulas condicionais, laços de repetição e mecanismos para acessar os dados do arquivo utilizado como entrada (similar ao código indicado pela letra b da Figura 3.8), entre outros. Para o gerador realizar a geração de um arquivo de saída, ele deve receber como entrada um gabarito e os dados de entrada contendo uma especificação de software.

A utilização de linguagens de gabaritos comerciais tem as mesmas vantagens apresentadas da abordagem de criação de uma nova linguagem de gabaritos e ainda, por utilizar uma linguagem pronta, tem a vantagem de economizar esforço no processo.

Como se deve evitar codificar os geradores configuráveis durante a sua especialização para um domínio específico, os gabaritos devem ser implementados por uma linguagem própria ou por uma linguagem comercial. A decisão entre as duas abordagens deve levar em consideração os requisitos técnicos necessários para a geração e a preferência pessoal do engenheiro de domínio. Neste trabalho optou-se por usar uma linguagem comercial conforme será apresentado na Seção 4.3.4.2.

3.4.4.2 Mapeamento da LMA nos gabaritos

Tipicamente, a instância de uma LMA identifica unicamente uma aplicação, indicando seus pontos variáveis. Com base nessa instância da LMA, deve ser possível definir uma ou mais unidades implementacionais (tais como: classes, procedimentos, estruturas ou arquivos de configuração) que devem ser criadas na geração da aplicação. No exemplo do domínio de persistência de dados, as variabilidades referem-se ao nome e atributos da classe, nome e campos da tabela e mapeamento entre atributos da classe e campos da tabela. Assim, a instância da LMA deve definir essas variabilidades, que serão depois mapeadas para uma ou mais unidades implementacionais. Neste caso, suponha que o engenheiro de aplicação tenha preenchido os campos do formulário da Figura 3.3. Para mapear essas variabilidades em gabaritos, ele deve realizar o mapeamento conforme apresentado na Figura 3.10, em que pode-se ver as unidades implementacionais afetadas por cada variabilidade do domínio. Como exemplo de mapeamento de variabilidades para gabaritos, ao preencher o campo Class Name do formulário da Figura 3.3, o engenheiro de aplicação estará definindo uma das variabilidades do domínio. Isso refletirá em duas unidades implementacionais e, conseqüentemente, em dois gabaritos que usarão a in-

formação sobre essa variabilidade: a classe “Pessoa” propriamente dita e a interface “IPessoa”, que representa a interface de programação disponibilizada pela classe Pessoa.

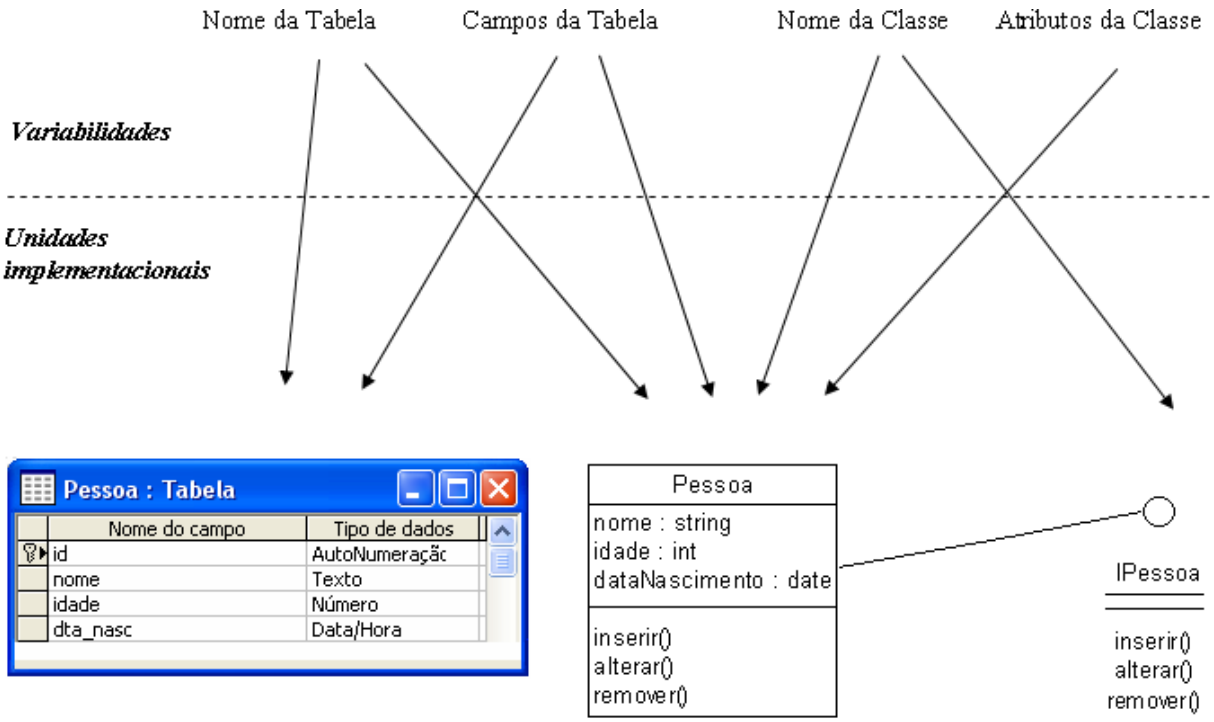

Figura 3.10: Mapeamento de variabilidades

O mapeamento da LMA em artefatos pode ser realizado graficamente como na Figura 3.10, ou como ilustrado na Tabela 3.4, por meio de uma representação tabular que relaciona as variabilidades com as unidades implementacionais por elas afetadas. Além disso, poderia ser acrescentada uma coluna nesta tabela para exibir os gabaritos correspondentes a cada unidade implementacional, que podem ser um ou mais gabaritos para cada unidade implementacional. Na seção 4.3.4.3 é apresentada uma linguagem por meio da qual é possível expressar esse mapeamento de maneira formal.

Tabela 3.4: Representação tabular do mapeamento das variabilidades

Unidades implementacionais alteradas	**Variabilidades**
Pessoa (classe)	Nome da Tabela, Campos da Tabela, Nome da Classe, Atributos da Classe
IPessoa (interface)	Nome da Classe
Pessoa (tabela)	Nome da Tabela, Campos da Tabela

As LMA’s são modeladas de acordo com as regras do domínio escolhido, tais como: regras de negócio, características opcionais e obrigatórias. As classes são modeladas de acordo com princípios do desenvolvimento de software, tais como: modularização, encapsulamento e

acoplamento. Em razão da diferença na modelagem, as variabilidades definidas na linguagem de alto nível não possuem necessariamente um mapeamento de um para um com os pacotes ou classes da implementação do sistema. Por exemplo: na arquitetura Web (cliente/servidor) as aplicações podem ser desenvolvidas com o padrão arquitetural de camadas (Buschmann et al., 2000). Nesse padrão, existe uma divisão modular entre as classes do sistema que são distribuídas em camadas. Em aplicações Web, normalmente essas camadas são divididas em: camada de apresentação, camada de negócios e camada de persistência (Booch et al., 1999). Nesse modelo, uma alteração de uma regra de negócio possivelmente irá atingir apenas a camada de negócios, mas uma alteração na forma e no conteúdo de uma entrada esperada do usuário, provavelmente terá um impacto em porções de todas as camadas, ou seja, o desenvolvedor poderá ter que alterar a camada de apresentação (telas gráficas), as regras de negócio (validações específicas) e a camada de dados (modificação nas classes persistentes).

Outra característica do software gerado é que, normalmente, as implementações possuem artefatos prontos, artefatos gerados automaticamente e artefatos criados manualmente. Os artefatos prontos podem ser partes da estrutura fixa (*frozen-spots*) de um framework, componentes de software ou trechos de artefatos reutilizados informalmente na implementação. Os artefatos gerados são todos os artefatos produzidos pelo gerador a partir da especificação. Finalmente, os artefatos criados manualmente são os artefatos criados para a implementação das funcionalidades para as quais o gerador não fornece apoio.

A boa separação entre os artefatos editados manualmente e os artefatos gerados automaticamente afeta diretamente a manutenibilidade do sistema, conforme descrito na Seção 2.4.2. Para que seja possível construir e manter o sistema sem a perda do controle das versões, deve existir uma maneira de identificar quais classes foram geradas automaticamente e quais classes foram editadas manualmente. Se as classes geradas automaticamente forem editadas manualmente e não forem protegidas, o trabalho irá se perder com a regeração dessas classes. Além disso, como é usual que os requisitos do sistema mudem antes que o sistema seja terminado (Steyaert et al., 1996; Tamai e Itou, 1993; Bell e Thayer, 1976; Berry, 1998; Allen, 1998; Pressman, 2001; Sawyer, 2004; Sommerville, 2003) existe uma possibilidade muito grande que o gerador de aplicação não produza todos os artefatos necessários para implementar as aplicações. Por esse motivo, é importante que durante essa atividade os diagramas de projeto sejam reavaliados para definir em quais pontos o sistema poderá ser modificado ou estendido. Quanto antes os pontos de modificação e extensão do sistema forem detectados, menor será o impacto dessas mudanças no processo de desenvolvimento das aplicações.

3.4.5 Documentação do domínio

Na atividade de documentação do domínio, o engenheiro de domínio deve criar um conjunto de documentos que descrevam o funcionamento dos artefatos reutilizáveis criados durante a atividade de implementação de domínio e um manual de uso do GAC configurado no domínio escolhido. O manual de uso deve detalhar os principais conceitos do domínio, a linguagem de modelagem de aplicações utilizada para expressar aplicações no domínio, as principais regras de validação da LMA, os artefatos que são gerados e como esses artefatos devem ser utilizados.

3.5 Engenharia da aplicação

A realização da etapa de engenharia de aplicação proposta nesta Seção tem o objetivo de construir aplicações com o uso de um GAC configurado para um domínio específico. Nessa etapa, o engenheiro de aplicação deve realizar as seguintes atividades: engenharia de requisitos, criação do modelo da aplicação, geração dos artefatos, implementação das funcionalidades não cobertas pelo gerador, teste e manutenção do sistema. Nas Seções seguintes, as atividades da engenharia da aplicação são detalhadas.

3.5.1 Engenharia de requisitos

Durante a engenharia de requisitos, o engenheiro de aplicação deve coletar as necessidades dos clientes e criar o documento de requisitos do sistema. Após a criação desse documento, o engenheiro deve realizar o processo de avaliação de uso dos GAC's apresentado na Seção 3.2. Se ele decidir utilizar o GAC, ele deve seguir com o processo de engenharia de aplicação proposto nesta Seção, caso contrário, ele deve utilizar outros métodos para implementar o sistema.

3.5.2 Criação do modelo da aplicação

O modelo da aplicação é uma instância da LMA que representa uma aplicação particular. O engenheiro de aplicação deve utilizar o GAC configurado em um domínio específico para criar o modelo de uma aplicação específica com base nas necessidades dos seus clientes. Como exemplo, no domínio de persistência, o engenheiro de aplicação deve criar um modelo que represente todas as entidades persistentes do sistema que está sendo construído. Por exemplo, ele pode preencher os formulários como o ilustrado na Figura 3.3.

3.5.3 Geração de artefatos

O gerador de aplicação configurável deve validar o modelo da aplicação e gerar artefatos com base nele. O processo de geração deve produzir diversos arquivos que representam a implementação e são armazenados em locais determinados pelo engenheiro de domínio ou engenheiro de aplicação.

No exemplo do domínio de persistência, os artefatos gerados são as classes e interfaces que implementam a persistência e o *script* SQL de criação das tabelas no banco de dados relacional. Como ilustração, com base no modelo apresentado na Figura 3.3, um gerador poderia produzir a interface de programação e a classe apresentadas nas Listagens 3.1 e 3.2, respectivamente, e o *script* de criação de tabelas apresentado na Listagem 3.3.

Listagem 3.1: Interface IPessoa

```
1 package pessoa;
2
3 public interface IPessoa  {
4
5  public boolean save();
6  public boolean getById(int id);
7  public boolean delete();
8
9 }
```

Listagem 3.2: Classe Pessoa

```
1 package pessoa;
2
3 import java.sql.*;
4
5 public class Pessoa  {
6  private int id;
7  private String nome;
8  private int idade ;
9  private int dataNascimento;
10  private MyConnection con;
11
12  public Pessoa(MyConnection con)  {
13   this.con = con;
14   id = 0;
15   nome = new String();
16   idade  = 0;
17   dataNascimento = 0;
18  }
19
20  //-----------------------------------------
21  //getters and setters
22
23  public int getId()  {
24   return id;
25  }
26  public void setId(int id)  {
```

```
27   this.id = id;
28  }
29  public String getNome()  {
30   return nome;
31  }
32  public void setNome(String nome)  {
33   this.nome = nome;
34  }
35  public int getIdade ()  {
36   return idade ;
37  }
38  public void setIdade (int idade )  {
39   this.idade  = idade ;
40  }
41  public int getDatanascimento()  {
42   return dataNascimento;
43  }
44  public void setDatanascimento(int dataNascimento)  {
45   this.dataNascimento = dataNascimento;
46  }
47
48  //-------------------------------------
49  //persistent methods
50
51  public boolean save()  {
52   Statement stmt = null;
53   String query = "INSERT INTO pessoa ";
54   query = query + "(id, nome, idade, dta_nasc) ";
55   query = query + " VALUES ";
56   query = query + "(ID_, 'nome_', idade _, dataNascimento_)";
57
58   query = query.replaceFirst("ID_", new Integer(id).toString());
59   query = query.replaceFirst("nome_", nome);
60   query = query.replaceFirst("idade _",\\
61           new Integer(idade ).toString());
62   query = query.replaceFirst("dataNascimento_",\\
63           new  Integer(dataNascimento).toString());
64
65   try {
66    stmt = con.createStatement();
67    stmt.executeUpdate(query);
68    try {
69     stmt.close();
70     return true;
71    } catch (SQLException ex) {
72     System.out.println(ex);
73     return false;
74    }
75   }catch(Exception ex)  {
76    System.out.println(ex);
77    return false;
78   }
79  }
80
81  public boolean getById(int id)  {
82   Statement stmt = null;
83   ResultSet rs = null;
```

```
84
85   String query = "SELECT * FROM pessoa WHERE ID = ID_";
86   query = query.replaceFirst("ID_", new Integer(id).toString());
87
88   try {
89    stmt = con.createStatement();
90    rs = stmt.executeQuery(query);
91
92    while (rs.next() )  {
93     this.id = rs.getInt("ID");
94     nome = rs.getString("nome");
95     idade  = rs.getInt("idade");
96     dataNascimento = rs.getInt("dta_nasc");
97    }
98
99    try {
100    stmt.close();
101    return true;
102   } catch (SQLException ex) {
103    System.out.println(ex);
104    return false;
105   }
106
107  }catch(Exception ex)  {
108   System.out.println(ex);
109   return false;
110  }
111 }
112
113 public boolean delete()  {
114  Statement stmt = null;
115
116  String query = "DELETE FROM pessoa WHERE ID = ID_";
117  query = query.replaceFirst("ID_", new Integer(id).toString());
118
119  try {
120   stmt = con.createStatement();
121   stmt.executeUpdate(query);
122
123   try {
124    stmt.close();
125    return true;
126   } catch (SQLException ex) {
127    System.out.println(ex);
128    return false;
129   }
130
131  }catch(Exception ex)  {
132   System.out.println(ex);
133   return false;
134  }
135 }
136
137 // Safe-zone
138 //----------------------------------------
139 // START-SAFE(newMethods)
140 // END-SAFE
```

```
141 }
```

Listagem 3.3: Script de criação da tabela Pessoa

```
1
2 drop table IF EXISTS pessoa;
3 CREATE TABLE pessoa (id int,
4                      nome  varchar(30),
5                      idade int,
6                      dta_nasc int);
```

3.5.4 Implementação das funcionalidades não cobertas pelo gerador

Um gerador pode gerar todos os artefatos necessários para implementar uma aplicação ou pode gerar apenas parte desses artefatos. Em domínios para os quais o gerador de aplicação gera a maior parte dos artefatos, o engenheiro de aplicação deverá produzir manualmente apenas pequenas modificações e personalizações, já em domínios em que o gerador gera apenas parte dos artefatos, como é o caso do exemplo de persistência, o engenheiro de aplicação deverá produzir manualmente grande parte da aplicação e integrar os artefatos gerados com os artefatos produzidos manualmente.

3.5.5 Testes e Manutenção

O processo de testes pode ser realizado antes, durante e depois do processo de engenharia da aplicação. Neste trabalho não é adotada nenhuma técnica ou método de testes específicos, deixando para o engenheiro de aplicação a liberdade de realizar essa atividade da forma que ele achar mais conveniente.

A manutenção do sistema pode ser realizada em diversas partes da aplicação, depois de pronta, para a correção de erros, adição de novas funcionalidades, etc. Para realizar a manutenção pode ser necessário alterar o modelo da aplicação e regerar a aplicação com o GAC, podem ser necessárias alterações nos artefatos implementados manualmente, podem ser necessárias alterações manuais nos artefatos gerados ou podem ser necessárias alterações na configuração de domínio do GAC.

3.5.5.1 Manutenção no modelo da aplicação

Esse tipo de manutenção deve ser realizada com o apoio do GAC. O engenheiro de aplicação deve abrir o GAC com os dados da aplicação que precisa ser mantida, alterar o modelo da aplicação, regerar os artefatos, testá-los, colocá-los sob controle de configuração e integrá-los com os outros artefatos do sistema.

3.5.5.2 Manutenção nos artefatos implementados manualmente

Nesse tipo de manutenção, o engenheiro de aplicação deve modificar os artefatos implementados manualmente, testá-los, colocá-los sob controle de configuração e integrá-los com os outros artefatos do sistema.

3.5.5.3 Manutenção manual nos artefatos gerados

Nesse tipo de manutenção, o engenheiro de aplicação deve alterar manualmente os artefatos gerados e proteger essas alterações de possíveis perdas em um novo processo de geração. Se o GAC oferecer apoio para geração de artefatos com zonas de segurança (ver Seção 2.4.2 e Seção 4.3.4.2), o engenheiro de aplicação deve realizar as alterações dentro das zonas de segurança. Dessa forma, os artefatos poderão ser regerados em novo processo de manutenção sem a perda dessas alterações manuais. Caso o GAC não apoie o desenvolvimento manual com zonas de segurança, para evitar a perda das alterações manuais, o engenheiro deve marcar o arquivo de forma que seja indicado que ele não poderá mais ser regerado pelo GAC em um novo processo de manutenção.

3.5.5.4 Manutenção na configuração de domínio

A alteração na configuração do domínio deve ser realizada pelo engenheiro de domínio e tem o objetivo de adaptar o domínio configurado no GAC da seguinte forma: alterar a LMA, alterar as validações da LMA ou alterar os gabaritos para que sejam gerados artefatos diferentes.

Em qualquer um dos casos, o engenheiro de domínio deve estabelecer um processo de gerenciamento de versões de configuração de domínio. Caso as atualizações também sejam aplicáveis para projetos legados desenvolvidos com o GAC na configuração original, o engenheiro de domínio deve definir um processo para adaptar os dados dos projetos legados armazenados pelo GAC na nova configuração de domínio. Caso as atualizações não sejam aplicáveis a sistemas legados desenvolvidos com o GAC na configuração original, o engenheiro de domínio deve manter as duas versões da configuração de domínio.

3.6 Considerações finais

Neste Capítulo apresentou-se de maneira genérica, os geradores de aplicação configuráveis, a sua etapa de desenvolvimento, configuração e utilização. A configuração do GAC é realizada durante a etapa da engenharia de domínio, na qual um domínio de aplicação é analisado, artefatos para apoiar o desenvolvimento de aplicações nesse domínio são projetados e implementados, o GAC é configurado para gerar artefatos e o domínio é documentado. A utilização

do GAC é realizada durante a etapa de engenharia de aplicações, na qual os requisitos dos clientes são elicitados, o modelo da aplicação é criado, os artefatos são gerados, as funcionalidades não cobertas pelo gerador são implementadas, os artefatos são testados e mantidos.

Algumas atividades das etapas de engenharia de domínio e engenharia de aplicação podem ser realizadas de maneiras diferentes e outras devem ser realizadas exatamente como foram descritas neste Capítulo. As atividades que podem ser realizadas de maneiras diferentes são as atividades em que foi deixado para o engenheiro de domínio ou engenheiro de aplicação escolher as técnicas ou métodos que ele quiser, como é o caso das atividades: análise do domínio, projeto e implementação de domínio, documentação de domínio, engenharia de requisitos, implementação das funcionalidades não cobertas pelo gerador e testes. As atividades que devem ser realizadas da mesma forma como foram apresentadas são as atividades: configuração da ferramenta, criação do modelo da aplicação e geração de artefatos. O processo de configuração e utilização dos GAC's foi apresentado dessa forma para garantir uma maior flexibilidade no uso dos GAC's e permitir que o engenheiro de domínio e o engenheiro de aplicação possam utilizar e/ou adaptar o processo de configuração e utilização dos GAC's em projetos de software que utilizam métodos e processos de desenvolvimento distintos, tais como: métodos orientados a objetos, linhas de produtos e frameworks baseados em linguagens de padrões.

CAPÍTULO

4

O Gerador de aplicação configurável Captor

4.1 Considerações iniciais

No Capítulo 3 foram apresentados de maneira genérica os geradores de aplicação configuráveis e a sua etapa de desenvolvimento, a etapa de engenharia de domínio, na qual eles são configurados e a etapa de engenharia de aplicação, na qual eles são utilizados. Algumas das atividades dessas etapas, tais como: as atividades da etapa de desenvolvimento, a criação da LMA, configuração da ferramenta e criação do modelo da aplicação, podem ser realizadas de maneiras diferentes, dependendo da forma como o gerador configurável é implementado.

Neste Capítulo é apresentado de forma específica o gerador de aplicação configurável Captor, a sua etapa de desenvolvimento, engenharia de domínio e aplicação. O Capítulo está organizado da seguinte forma: na Seção 4.2 são apresentadas as atividades da etapa de desenvolvimento da Captor, na Seção 4.3 são apresentadas as atividades da etapa de engenharia de domínio na qual a Captor é configurada, na Seção 4.4 são apresentadas as atividades da etapa de engenharia de aplicação na qual a Captor é utilizada para produzir artefatos, finalmente, na Seção 4.5, são apresentadas as considerações finais deste Capítulo.

4.2 Etapa de desenvolvimento da Captor

O gerador de aplicação configurável Captor (do ingles ***C***onfigurable ***Ap***plication Genera***tor***) foi inicialmente projetado para ser um gerador configurável por linguagens de padrões e, na primeira fase do desenvolvimento, seus requisitos foram definidos com base no gerador de aplicação específico baseado em linguagens de padrões GREN-Wizard (Braga e Masiero, 2003) (apresentado na Seção 2.4.3). Terminado o desenvolvimento do gerador configurável por linguagem de padrões, os autores decidiram que a ferramenta deveria ser configurada não apenas por linguagens de padrões, mas também por outras linguagens de modelagem de aplicações que podem ser definidas para domínios específicos. Esse processo levou a uma segunda fase de desenvolvimento, que culminou na criação do gerador de aplicação configurável Captor e na definição das atividades das etapas de engenharia de domínio e engenharia de aplicação com geradores configuráveis. Nesta monografia será apresentado somente o gerador de aplicação configurável construído à partir da segunda fase do desenvolvimento.

Para a realização da etapa de desenvolvimento da Captor, foram realizadas as atividades de análise, projeto, implementação e testes. Essas atividades foram realizadas de maneira interativa e incremental, similar ao modelo de processo apresentado na Seção 3.3, e são detalhadas nas próximas Seções.

4.2.1 Análise

Durante a atividade de análise, diversos geradores de aplicação ou ferramentas que realizam geração foram avaliados, dentre eles: o gerador GREN-Wizard (Braga e Masiero, 2003), o gerador baseado em gabaritos apresentado por Rausch (2001), AndroMDA (Bohlen et al., 2006), Albatross, P2 (Batory e Geraci, 1996), P3 (Batory et al., 1998), MOMoc (Bichler, 2003), GenDoc, JavaDoc, Microsoft Visual Studio, Borland JBuilder, Borland Delphi, Eclipse, SQLServer, MS FrontPage, DreamWeaver, Rational Rose e MVCase (Almeida et al., 2002). Dentre essas ferramentas, algumas são projetadas especialmente para gerar artefatos e outras apresentam funcionalidades primárias específicas e capacidade de geração secundária, como são os casos dos últimos seis itens. Como resultado dessa avaliação e da generalização dessas ferramentas, foi criado o documento de visão geral apresentado na próxima Seção, que serviu de base para o projeto e implementação da Captor.

4.2.1.1 Visão geral da Captor

A Captor é um gerador de aplicação configurável que pode ser configurado para diversos domínios e fornecer apoio no desenvolvimento de diversas aplicações dentro de um mesmo

domínio. Para utilizar a Captor, o engenheiro de domínio deve configura-la e o engenheiro de aplicação deve utiliza-la para a construção de aplicações em um domínio particular.

A configuração é realizada por meio da modularização da especificação em formulários ou mais especificamente, criação de linguagens de modelagem de aplicações baseadas em formulários. Cada formulário apresenta um conjunto de campos, representados por elementos gráficos, tais como: caixas de texto, caixas de seleção, tabelas, entre outros. A interface gráfica com o usuário (GUI) da Captor é configurada para apresentar comportamento variável, ou seja, ela é configurada para modificar a sua aparência para receber especificações baseadas em formulários de diferentes domínios. O engenheiro de domínio, por meio de um arquivo de configuração no formato XML, define quantos formulários devem ser utilizados em uma especificação, como esses formulários devem ser organizados e quais as regras de validação de cada campo de cada formulário. A Captor lê o arquivo XML utilizado para configurar a GUI e definir as regras de validação dos elementos da especificação, e disponibiliza os formulários adequados para a edição da especificação. Quando solicitado pelo usuário, ela pode conferir as regras de validação e persistir os dados da especificação inseridos em sua interface gráfica no formato XML.

Para completar a configuração do domínio, o engenheiro de domínio deve construir um conjunto de gabaritos para transformar a especificação inserida na GUI e persistida no formato XML para artefatos de software, tais como: código-fonte, casos de testes e documentos. Os gabaritos utilizados pela ferramenta para transformar os dados da especificação em artefatos devem ser construídos com a linguagem de gabaritos XSL (XSL, 2005) e a Captor fornece um mecanismo configurável de seleção de gabaritos durante a geração [1].

Após a realização da configuração da GUI, o engenheiro de aplicação pode abrir a Captor no domínio recém-configurado com o objetivo de inserir a especificação de uma aplicação particular. Nesse momento, a Captor fornece um mecanismo que realiza a persistência da especificação inserida em sua GUI para o formato XML (salvamento do projeto) e carregamento desses dados do meio persistente para a GUI (abertura de projeto), sem a necessidade de nenhuma configuração especial.

Para utilizar a Captor, o engenheiro de aplicação deve criar um projeto que, na ferramenta é representado por um conjunto de arquivos que contém informações sobre o domínio do projeto, nome do projeto, caminhos de diretório dos arquivos utilizados para armazenar a especificação do projeto e de armazenamento dos arquivos gerados, a especificação do projeto e os arquivos gerados. A Captor deve apoiar o gerenciamento completo de projetos, ou seja, criação, salvamento, fechamento, abertura e remoção.

Além das funcionalidades apresentadas, a Captor oferece apoio na criação de gabaritos com zonas de segurança, pré e pós processamento dos artefatos gerados e mecanismos de depuração

[1] Esse mecanismo é detalhado na Seção 4.3.4.3

de erros na especificação[2]. As zonas de segurança são utilizadas para armazenar as modificações manuais realizadas nos artefatos gerados e inseri-las novamente nos artefatos re-gerados. O pré e pós-processamento dos artefatos gerados devem processar os artefatos gerados antes e depois da geração (cópia, remoção, alteração, controle de configuração, *backup*, etc). Finalmente, o mecanismo de depuração de erros deve informar para o engenheiro de aplicação detalhes dos erros encontrados pelo módulo de validação da especificação.

4.2.2 Projeto

A arquitetura da Captor é apresentada na Figura 4.1. A Captor é composta por quatro módulos: gerenciamento de interface (GUI), gerenciamento de domínio, motor de transformação de gabaritos e gerenciamento de projeto. Esses módulos são apresentados nas Seções a seguir.

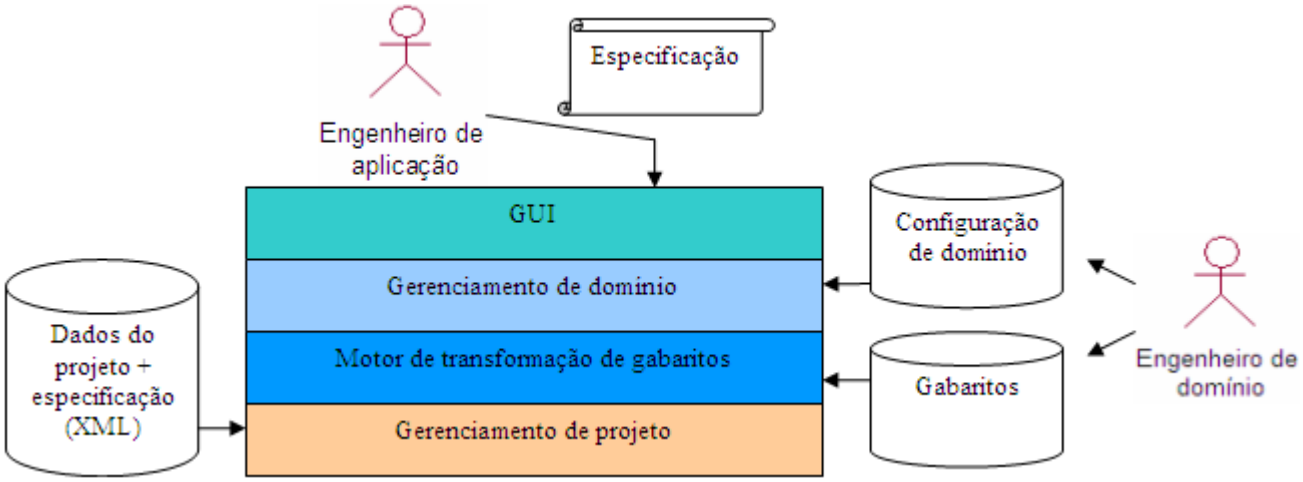

Figura 4.1: Arquitetura da Captor

4.2.2.1 Gerenciamento de projeto

O módulo de gerenciamento de projetos é responsável pela manutenção dos dados dos projetos criados na Captor, ou seja, criação manutenção e remoção de projetos. Esse módulo fornece a interface `IProjectManager`, é implementado pelo pacote `projectmanager` e é ilustrado na Figura 4.2.

Na Figura 4.3 é apresentado o diagrama de classes que implementa o pacote `projectmanager`. A classe `ProjectManager` realiza a interface `IProjectManager` e está implementada de acordo com o padrão de projeto Facade (Gamma et al., 1995). Essa classe fornece uma interface única para disponibilizar os serviços do módulo de gerenciamento de projetos e delega as responsabilidades desses serviços para classes auxiliares.

[2]Esses itens são detalhados nas próximas Seções.

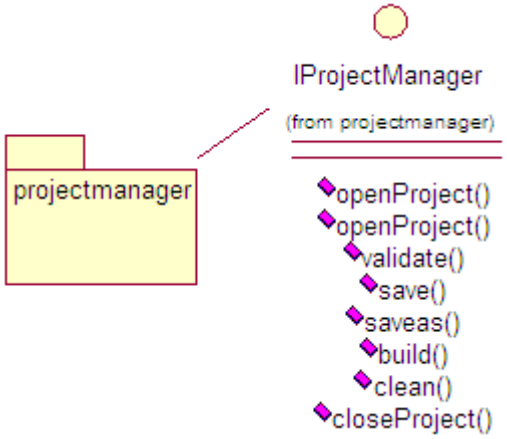

Figura 4.2: Módulo de gerenciamento de projetos

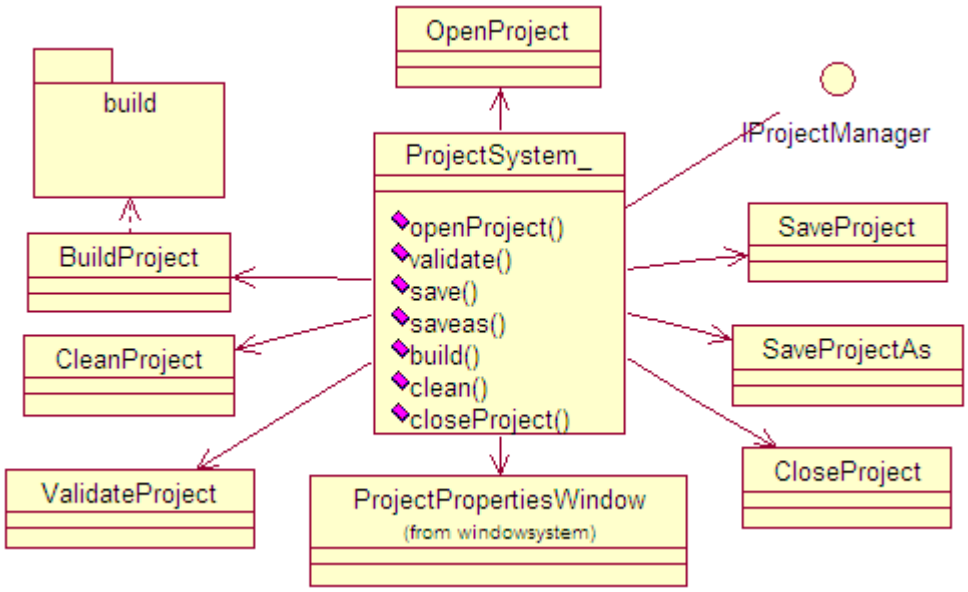

Figura 4.3: Classes que implementam o módulo de gerenciamento de projetos

As funcionalidades do gerenciamento de projeto são: abrir, fechar, compilar, limpar, salvar, salvar como, validar e mostrar propriedades de um projeto e, essas funcionalidades são implementadas pelas classes: `OpenProject`, `CloseProject`, `BuildProject`, `CleanProject`, `SaveProject`, `SaveAsProject`, `ValidateProject` e `ProjectPropertiesWindow`, respectivamente.

4.2.2.2 Gerenciamento de domínio

A Captor deve ter a capacidade de ser configurada para apoiar diversos domínios. Para cada domínio em que a Captor oferecer apoio, ela deve ler e interpretar um conjunto de informações de domínio, contidas em arquivos de configuração no formato XML, utilizado para apresen-

tar um comportamento variável, tais como: ser capaz de receber especificações de diferentes linguagens de modelagem de aplicações, validar essas especificações e gerar artefatos.

O módulo `DomainManager`, apresentado na Figura 4.4, é responsável por abrir os arquivos de configuração de um domínio e criar uma estrutura de dados no modelo da aplicação que represente o domínio escolhido pelo usuário e que possa ser utilizado pelos outros módulos (o formato dos arquivos de configuração são apresentados na Seção 4.3.4).

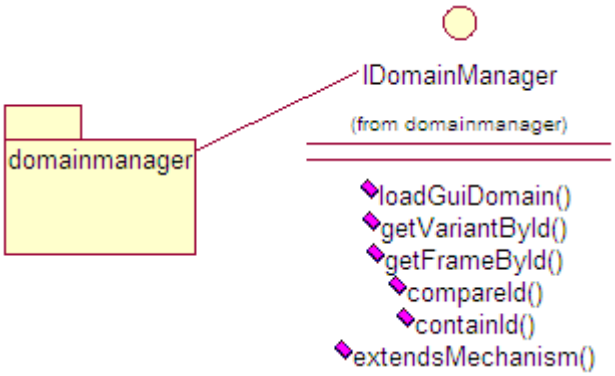

Figura 4.4: Módulo de gerenciamento de domínio

4.2.2.3 Gerenciamento de interface GUI

O módulo de gerenciamento de interface (GUI) tem a responsabilidade de apresentar as janelas para o usuário. As janelas disponíveis são: principal, criação de novo projeto, propriedades do projeto, de ajuda, de ajuste de preferências e de validação da configuração GUI de um domínio.

A janela principal deve apresentar comportamento variável de acordo com os dados fornecidos pelo módulo de gerenciamento de domíno por meio dos arquivos de configuração de domínio.

4.2.2.4 Transformação de gabaritos

Durante a geração de artefatos, o módulo de gerenciamento de projetos tem a responsabilidade de selecionar os gabaritos que devem ser gerados e utilizar o módulo de transformação de gabaritos, implementado no pacote `build`, para produzir os artefatos.

O fluxo de execução do módulo de transformação de gabaritos é apresentado na Figura 4.5. O módulo `build` deve receber como entrada um arquivo de gabarito e um arquivo XML com os dados da especificação (ou instância da LMA), e gerar um artefato de saída utilizando o componente de transformação de gabaritos XSL fornecido pela Sun Microsystems (Java, 2006). Se o artefato de saída já tiver sido gerado antes, o módulo `build` deve analisar esse arquivo,

buscar por zonas de segurança, armazenar o conteúdo das zonas de segurança e colocar esse contéudo no arquivo gerado.

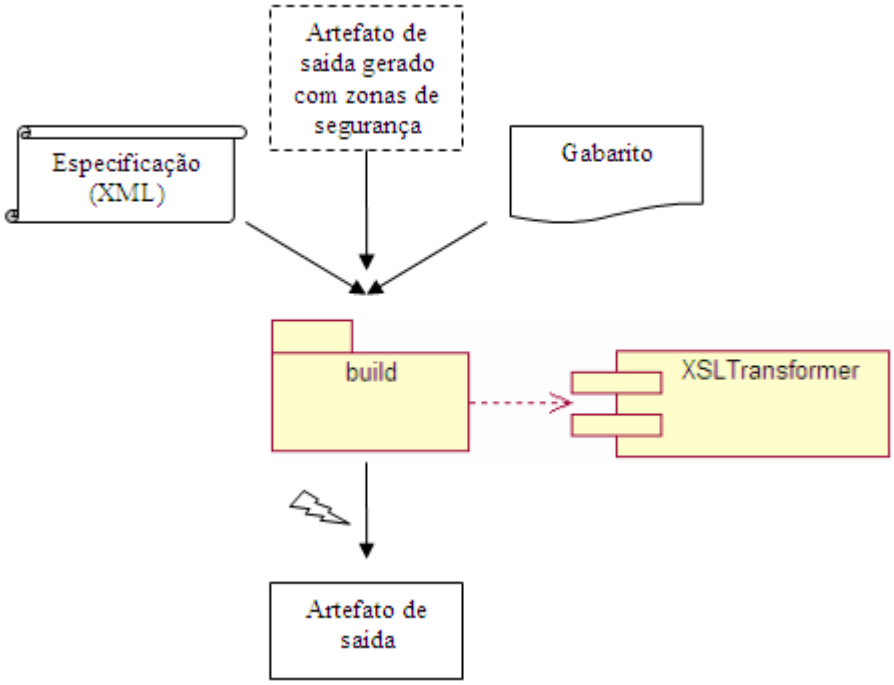

Figura 4.5: Módulo de transformação de gabaritos

4.2.3 Implementação e Testes

Na atividade de implementação e testes, os módulos da Captor foram implementados, testados e integrados em um processo iterativo e incremental. Na primeira iteração, escolheram-se algumas funcionalidades que foram implementadas e testadas. Com o término dessas atividades, os artefatos de análise e projeto foram revisados e atualizados. Para dar continuidade ao processo, uma nova iteração foi planejada e realizada e assim sucessivamente, até a Captor ser completamente implementada. Durante a implementação, alguns módulos da Captor passaram por testes com o framework de testes JUnit (Gamma e Beck, 2005).

4.3 Engenharia de domínio

Conforme apresentado no Capítulo 3, os geradores configuráveis precisam ser configurados para serem utilizados em um domínio particular. Nesta Seção é apresentada, de forma específica, a etapa de engenharia de domínio na qual a Captor pode ser configurada para fornecer apoio em um domínio particular. Algumas atividades, tais como o projeto, implementação e documentação de domínio devem ser realizadas da mesma forma como foram apresentadas no

Capítulo 3, e por esse motivo, elas não são detalhadas nesta Seção. As atividades específicas, tais como a criação da LMA e configuração da Captor, são apresentadas em detalhes nas próximas Seções.

4.3.1 Análise de domínio

Conforme apresentado na Seção 3.4.1, as sub-atividades da análise de domínio são: descrição do domínio, definição da terminologia comum, análise de aspectos similares e variáveis, e criação da linguagem de modelagem de aplicações.

4.3.1.1 Descrição do domínio

A sub-atividade de descrição de domínio não depende da implementação do gerador configurável e deve ser realizada de acordo com a Seção 3.4.1.1.

4.3.1.2 Definição da terminologia comum

A sub-atividade de definição de terminologia comum não depende da implementação do gerador configurável e deve ser realizada de acordo com a Seção 3.4.1.2.

4.3.1.3 Análise dos aspectos similares e variáveis

A sub-atividade de análise dos aspectos similares e variáveis não depende da implementação do gerador configurável e deve ser realizada de acordo com a Seção 3.4.1.3.

4.3.1.4 Criação da linguagem de modelagem de aplicações

Na Captor, são utilizadas LMA's declarativas que são especificadas em um conjunto de formulários organizados hierarquicamente em forma de árvore. Cada formulário contém um conjunto de campos, representados por elementos gráficos (do inglês *widgets*), tais como: caixas de texto, caixas de seleção, etiquetas e tabelas. Para ilustrar esses formulários e os seus relacionamentos, apresenta-se na Figura 4.6 um formulário com três campos disponibilizando uma caixa de texto, uma caixa de seleção para a edição de dois dados simples e uma tabela para a edição de dados que devem ser indexados. Na Figura 4.7, apresenta-se a forma hierárquica de uma árvore com três formulários.

Os formulários da Captor podem apresentar pequenas variações, dependendo das escolhas do usuário e das necessidades dos clientes. Cada formulário da linguagem de modelagem de aplicação é definido pelo nome e pelo relacionamento que ele possui com outros formulários. Os elementos que cada formulário contém são definidos por uma estrutura chamada variante. Se

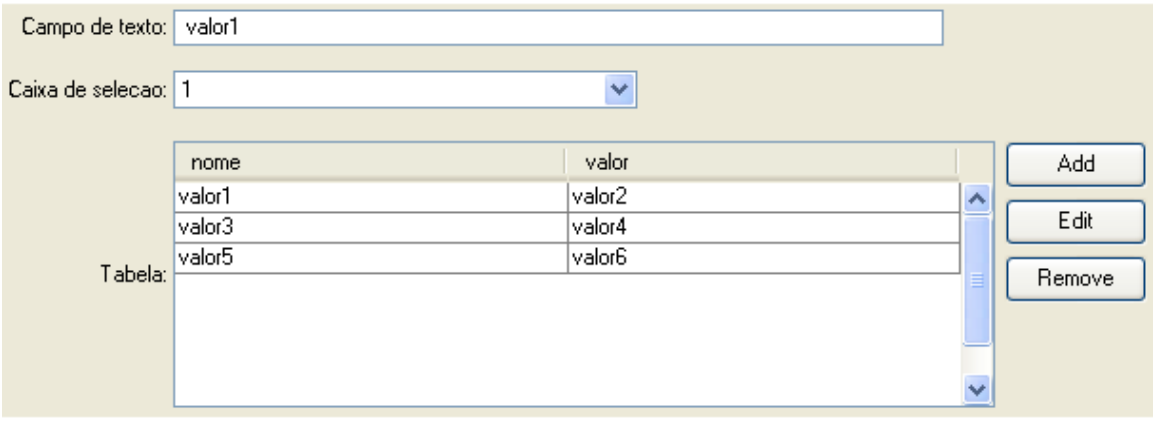

Figura 4.6: Representação gráfica de um formulário exemplo

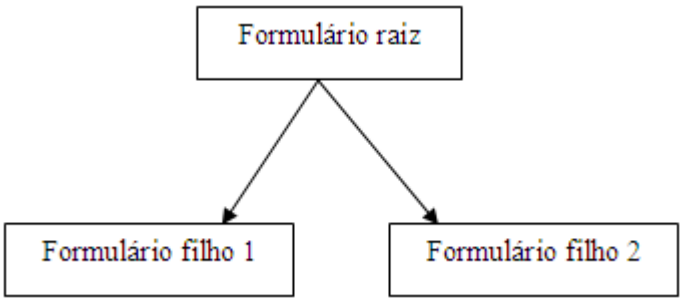

Figura 4.7: Árvore com três formulários

o engenheiro de domínio disponibilizar mais de uma variante para um formulário, o engenheiro de aplicação poderá escolher essa variante em tempo de modelagem da aplicação, e a Captor deve mostrar o mesmo formulário (mesmo nome e relacionamentos), mas com um conjunto de campos diferentes. Se o engenheiro de domínio disponibilizar apenas uma variante para um formulário, a Captor deve mostrar essa variante para o engenheiro de aplicação e não disponibilizar nenhum mecanismo de troca de variantes durante a especificação desse formulário. Como ilustração dessa situação, é apresentada na Figura 4.8 a árvore de formulários da Figura 4.7, mas com duas variantes do "formulário filho 1". Com essa árvore, o usuário pode, em tempo de modelagem da aplicação, escolher entre o formulário filho 1.1 e o formulário filho 1.2.

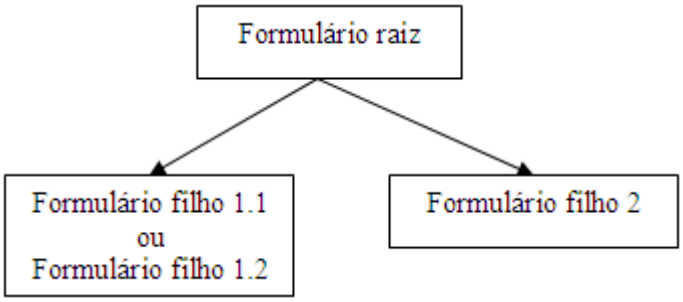

Figura 4.8: Árvore com três formulários e duas variantes para o formulário filho 1

Exemplo de criação de uma LMA utilizando o modelo de árvores de formulários:

O primeiro passo na criação de uma LMA em forma de árvore é definir qual vai ser a estrutura hierárquica da árvore. Como exemplo, no domínio de persistência, a forma hierárquica dos formulários pode ser organizada como apresentado na Figura 4.9.

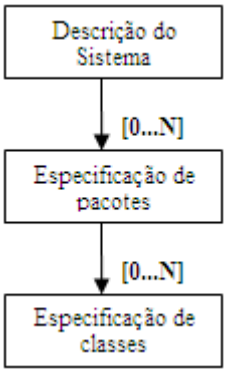

Figura 4.9: Exemplo de árvore com três formulários

Esses formulários são disponibilizados pela Captor conforme ilustrado na Figura 4.10. O formulário "descrição de sistema" é definido como formulário raiz, por ser o único formulário do diagrama da Figura 4.9 que não possui um pai. Esse formulário é utilizado para armazenar informações que descrevem o sistema que está sendo modelado e documentar a especificação na própria Captor. Ele possui relacionamento apenas com um formulário filho denominado "especificação de pacotes" e esse relacionamento é definido com cardinalidade mínima de 0 e máxima de N, ou seja, o formulário de descrição de sistema deve ter no mínimo zero e no máximo N formulários filhos do tipo "especificação de pacotes". O formulário de especificação de pacotes é utilizado para modularizar a geração das classes persistentes em pacotes diferentes e esse formulário pode ter como filho o formulário "especificação de classes" com cardinalidade mínima 0 e máxima N. O formulário de especificação de classes é utilizado para descrever as classes persistentes do sistema e relacionar essas classes com tabelas de um banco relacional.

O usuário pode navegar pelos formulários da especificação por meio do painel de navegação de formulários (localizado do lado esquerdo da ferramenta da Figura 4.10). Clicando com o botão esquerdo do mouse em um item da árvore, a ferramenta disponibiliza o formulário para edição no painel central e clicando com o botão direito do mouse nos itens da árvore, a Captor disponibiliza um menu *popup* para o usuário inserir formulários filhos no item selecionado, remover o formulário selecionado e alterar a ordem de visualização dos formulários, entre outras funcionalidades. O painel de baixo apresenta três caixas de texto (`Console`, `Error` e `Warning`) que são utilizadas pela Captor para informar ao usuário informações sobre o projeto, tais como: mensagens sobre os arquivos que são gerados, mensagens de erros detectados

na especificação e mensagens de avisos sobre os detalhes da geração. Conforme ilustrado no painel de navegação de formulários da Figura 4.10, cada nó "iteração" da árvore deve armazenar a especificação de uma aplicação, permitindo que o engenheiro de domínio modele e gere artefatos para várias aplicações em um domínio com apenas uma janela aberta.

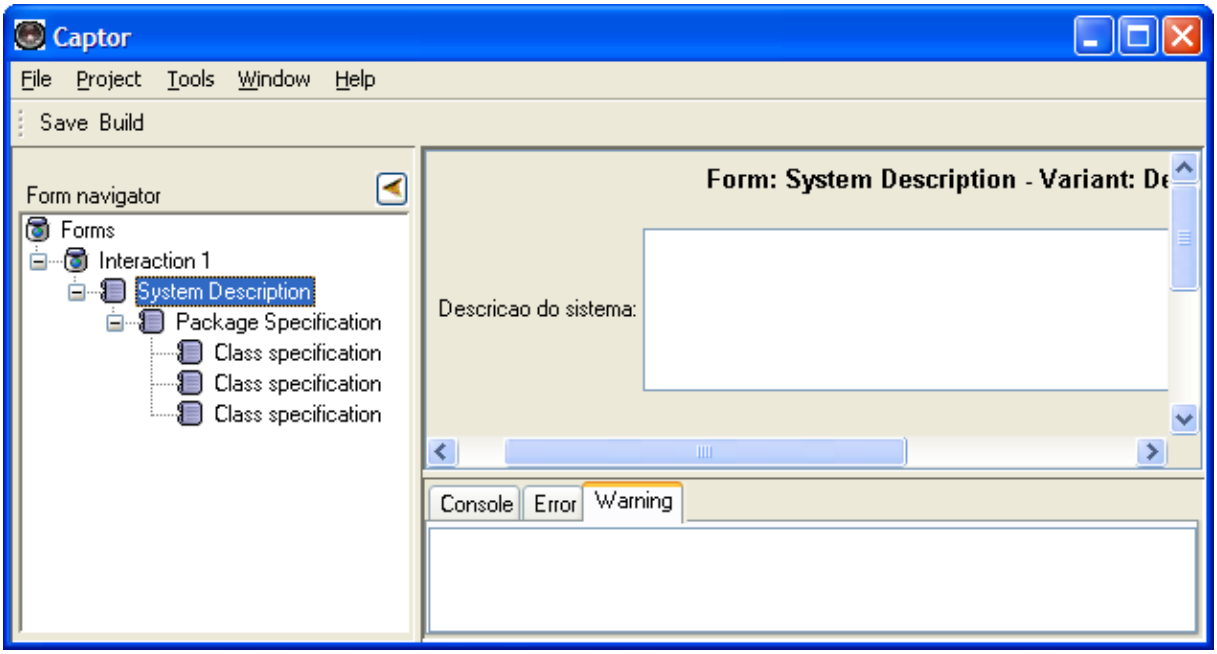

Figura 4.10: Formulários do domínio de persistência

Após a definição da estrutura hierárquica, o engenheiro de domínio deve definir a forma como o formulário deve se apresentar ao usuário, ou seja, o engenheiro deve definir quais campos cada formulário contém e quais os valores válidos dos dados inseridos nesses campos.

Os campos de um formulário são organizados linha a linha, ou seja, em cada linha[3] do formulário são apresentadas caixas de texto, caixas de seleção, tabelas, etiquetas, botões e outros elementos gráficos. A Figura 4.11 apresenta um formulário de especificação de classe com quatro campos e consequentemente quatro linhas, contendo como elementos gráficos três caixas de texto, quatro etiquetas, três botões e uma tabela com três colunas.

O conjunto de elementos gráficos contidos em uma linha, denominados "elementos de formulário", apresentam comportamento genérico, ou seja, durante a etapa de configuração, o engenheiro de domínio deve parametrizar esses componentes com o objetivo de fazer com que eles apresentem comportamento específico. Essa parametrização é feita por meio de um conjunto de parâmetros obrigatórios e, um conjunto de parâmetros opcionais. Os parâmetros são utilizados para realizar modificações na aparência dos elementos, definir regras de validações específicas de domínio nos dados inseridos nos elementos, entre outras especializações possíveis. Como

[3]Não necessariamente refere-se aqui a uma linha física do formulário, mas a um grupo de elementos relacionados posicionados próximos um do outro.

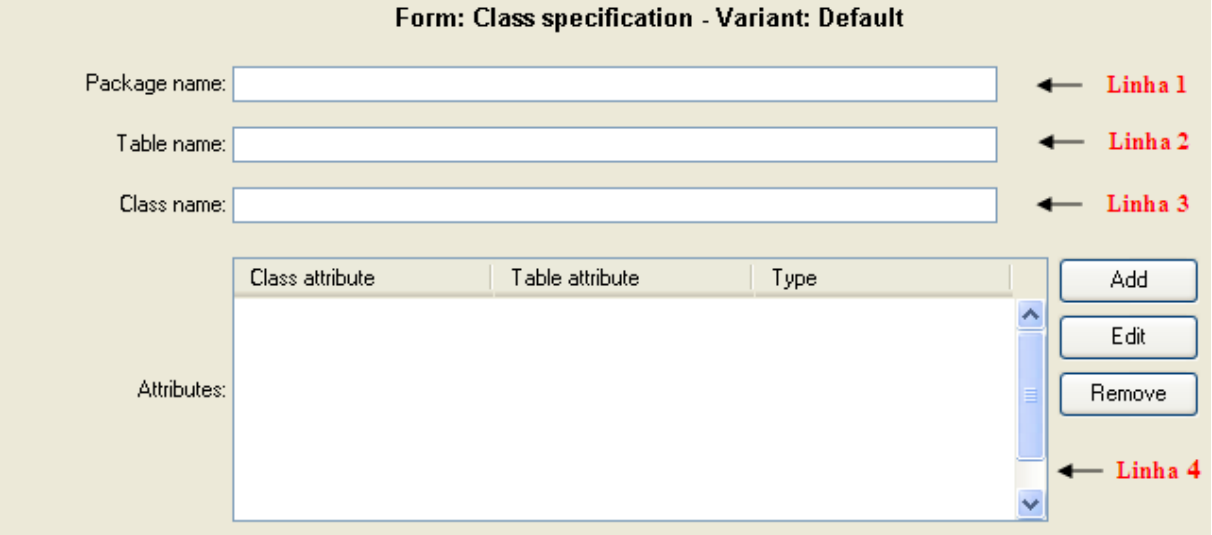

Figura 4.11: Representação gráfica de um formulário com quatro elementos

ilustração, a representação tabular do formulário da Figura 4.11 e a sua parametrização são apresentadas na Tabela 4.1.

Tabela 4.1: Representação tabular do formulário da Figura 4.11

Elemento	**Nome**	**Parâmetro**	**Valor**
TextPanel	Package name	id	packageName
		label	Package name:
		use	required
		regexp	([a-zA-Z]+[a-zA-Z-0-9]+)
TextPanel	Table name	id	tableName
		label	Table name:
		use	required
		regexp	[A-Za-z]+([a-zA-Z-_0-9]+)
TextPanel	Class name	id	className
		label	Class name:
		use	required
		regexp	[A-Z]+([a-zA-Z-0-9]+)
TablePanel	Attributes	id	attributes
		label	Attributes
		colname1	Class attribute
		colname2	Table attribute
		colname3	Type
		regexp3	[[int]*|[String]*|[bool]*]

No exemplo da Tabela 4.1, os três primeiros elementos (`TextPanel`) são representados pelos elementos de formulário `TextPanel`. Esse elemento contém uma etiqueta e uma caixa de texto, e é parametrizado com quatro parâmetros: id, label, use e regexp. O primeiro parâme-

tro serve para definir um identificador único para esse elemento, o segundo serve para configurar um valor para a etiqueta posicionada ao lado esquerdo da caixa de texto, o terceiro indica que o campo de texto não pode ter um valor nulo (uso requerido) e o quarto indica que o valor da caixa de texto deve estar no formato da expressão regular "([a-zA-Z]+[a-zA-Z-0-9]+)", ou seja, este valor deve ser formado por uma cadeia de caracteres iniciando com uma letra, seguido de um ou mais caracteres de letras e números.

O quarto elemento é representado pelo elemento de formulário `TablePanel`. Esse elemento contém uma etiqueta, uma tabela com número variável de colunas e um conjunto de botões, e é parametrizado com seis parâmetros: id, label, colname1, colname2, colname3 e regexp3. Os dois primeiros são similares aos descritos para o elemento TextPanel, o terceiro, quarto e quinto parâmetros são utilizados para indicar que devem ser mostradas três colunas com nomes `Class attribute`, `Table attribute` e `Type` respectivamente. O sexto parâmetro é utilizado na validação dos dados inseridos na terceira coluna da tabela e indica que ela só pode conter os valores "String", "int" ou "bool".

A criação de uma LMA no formato de árvore de formulários requer a criação de dois tipos de artefatos: o diagrama de formulários hierárquico e as representações tabulares de cada formulário da linguagem. Para realizar essa tarefa, o engenheiro de domínio deve consultar a documentação da Captor, disponível junto com o pacote de instalação no site `http://www.labes.icmc.usp.br/captor`, na qual é apresentado o conjunto de elementos de formulário disponíveis e os parâmetros que cada elemento aceita. Se o engenheiro de domínio precisar de elementos de formulários mais especializados, ele pode desenvolver os seus próprios elementos utilizando os recursos da Captor.

4.3.2 Projeto de domínio

A atividade de projeto de domínio não depende da implementação do gerador configurável e deve ser realizada de acordo com a Seção 3.4.2.

4.3.3 Implementação de domínio

A atividade de implementação de domínio não depende da implementação do gerador configurável e deve ser realizada de acordo com a Seção 3.4.3.

4.3.4 Configuração da ferramenta

Para melhor entender a configuração da Captor, é necessário entender o fluxo de execução durante sua utilização, que é dividido em duas etapas: edição da especificação e transformação da especificação em artefatos. Como ilustrado na Figura 4.12, na primeira etapa o engenheiro

de aplicação deve inserir a especificação de uma aplicação na GUI da Captor e, em seguida, ela deve validar a especificação e persisti-la no formato XML.

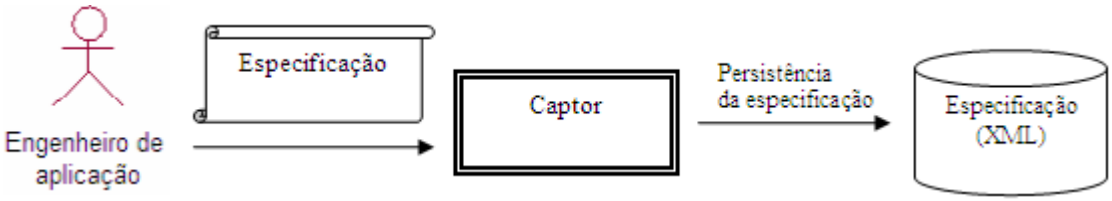

Figura 4.12: Primeira etapa do fluxo de execução da Captor

Como ilustrado na Figura 4.13, na segunda etapa a Captor utiliza a especificação persistida no formato XML e um conjunto de gabaritos para gerar artefatos de software.

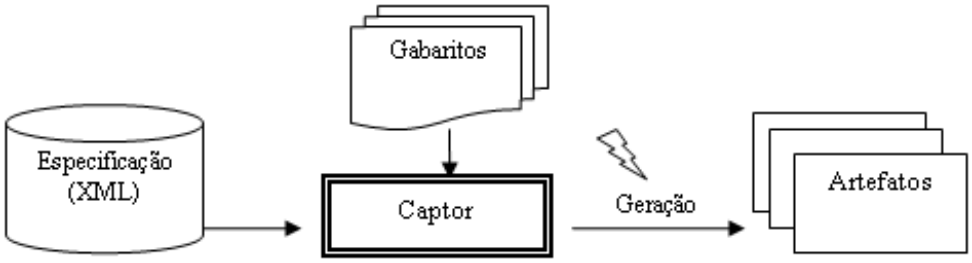

Figura 4.13: Segunda etapa do fluxo de execução da Captor

Como ilustrado na Figura 4.14, durante a utilização, quando for solicitada a geração da aplicação, a Captor deve processar um conjunto de gabaritos para transformá-los em artefatos. Dependendo do conteúdo da especificação, a Captor deve selecionar um conjunto de gabaritos para ser processado uma ou mais vezes. Por exemplo, no domínio de persistência, a Captor deve criar dois arquivos de saída (representando uma classe persistente e sua interface) para cada especificação de classe contida no modelo da aplicação e um arquivo de SQL com os *scripts* de criação de tabelas na base de dados para cada especificação.

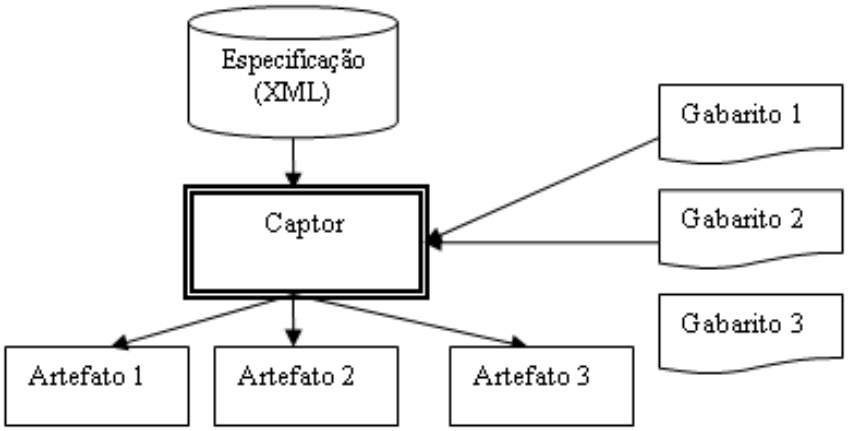

Figura 4.14: Seleção de gabaritos durante a geração de artefatos

A Captor fornece apoio para a seleção de gabaritos por meio da linguagem, criada neste trabalho, denominada MTL (do inglês *Mapping Transformation Language*). Para cada domínio em que a Captor é configurada, o engenheiro de domínio deve fornecer um arquivo de mapeamento de transformação de gabaritos. Esse arquivo é utilizado para realizar diversas asserções no modelo da aplicação, selecionar os gabaritos que devem ser utilizados para gerar os artefatos e indicar em que local cada arquivo gerado deve ser armazenado.

Além disso, a Captor e a ferramenta Ant (Davidson et al., 2005), podem ser configuradas para realizar um conjunto de pré e pós-processamentos específicos nos artefatos gerados e nos artefatos reutilizáveis desenvolvidos para apoiar o domínio, tais como: cópia dos artefatos reutilizáveis para o mesmo diretório (ou diretórios diferentes) dos artefatos gerados, remoção de arquivos, compilação automática de arquivos-fonte Java, implantação automática em servidores web ou servidores de aplicação, *commit* em sistemas de controle de versão como o CVS, entre outros.

Na Figura 4.15 são ilustrados os arquivos necessários para realizar a configuração da Captor. O arquivo de configuração de interface gráfica define como a interface com o usuário vai ser adaptada para receber uma instância de uma LMA específica (ou especificação) e quais são os critérios de validação desses dados. Os gabaritos são arquivos de texto que contém a parte fixa dos artefatos que devem ser gerados e instruções de como processar, com base nos dados da especificação, as partes variáveis desses artefatos. O arquivo de MTL é utilizado pela Captor para selecionar, dependendo dos dados da especificação, quais gabaritos devem ser utilizados na geração. Finalmente, os arquivos de pré e pós-processamento realizam processamentos específicos nos artefatos do domínio.

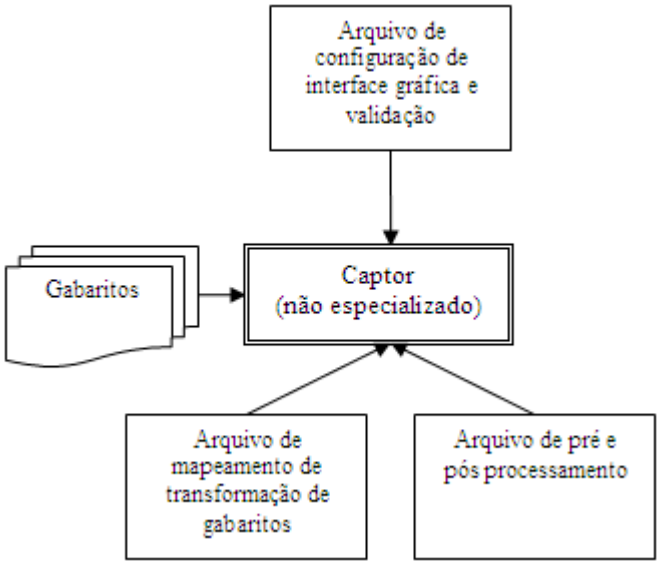

Figura 4.15: Implantação da configuração no gerador configurável

A atividade de configuração da Captor é dividida em quatro sub-atividades: configuração da interface e regras de validação, desenvolvimento dos gabaritos, criação do arquivo de mapeamento de transformação de gabaritos e criação dos arquivos de pré e pós-processamento, descritas a seguir. Cada uma dessas atividades tem o objetivo de criar uma das entradas da Figura 4.15.

4.3.4.1 Configuração da interface e das regras de validação da especificação

A Captor apresenta interface gráfica com comportamento adaptável, ou seja, quando a Captor é iniciada para a edição de uma LMA em um domínio específico, ela lê um arquivo de configuração com as definições dos formulários e das regras de validação dos formulários e disponibiliza os elementos gráficos necessários na tela.

O arquivo de configuração é definido no formato XML e possui diversas marcações para definir os formulários, os elementos dos formulários, as regras de validação dos dados inseridos nos formulários e o relacionamento entre os formulários.

Para facilitar a tarefa do engenheiro de domínio e esconder esses detalhes de implementação, a Captor oferece uma LMA (ou meta-LMA) para realizar a especificação de LMAs na própria Captor. Para inserir os dados de um novo domínio na meta-LMA, o engenheiro de domínio deve realizar as mesmas etapas de utilização realizadas em outros domínios, ou seja, criar um projeto, editar e validar a especificação desse projeto, e gerar artefatos para esse projeto. Assim, a Captor recém-instalada, que ainda não tenha sido configurada para nenhum domínio específico, já posssui pelo menos essa meta-LMA que deve ser usada para criar novos domínios.

Para iniciar a criação de um novo domínio, o engenheiro de domínio deve selecionar o menu Arquivo->Novo>Projeto. A seleção desse menu abre o *wizard* apresentado na Figura 4.16. Na primeira tela do *wizard*, o engenheiro de domínio deve selecionar a opção `New Captor Project` da árvore de seleção de domínios e clicar no botão `Next`. As telas seguintes recolhem informações sobre o projeto que deve ser criado, tais como: caminho de diretório de armazenamento dos arquivos do projeto e caminho do diretório no qual os arquivos gerados devem ser armazenados.

Como ilustrado na Figura 4.17, após a criação do projeto, a Captor criará diversos elementos gráficos na interface da janela principal e automaticamente mostrará os formulários mínimos que devem ser editados para a especificação de uma LMA, ou seja, o formulário raiz e todos os filhos do formulário com cardinalidade mínima maior que zero. O mínimo que se espera no domínio de criação de LMAs é o formulário de descrição de projeto, um formulário para a definição do formulário raiz e um formulário para a definição da variante padrão do formulário raiz (considere-se que cada formulário tem pelo menos uma variante, que é a padrão).

É importante notar que todos os formulários da especificação podem ser acessados pela árvore localizada no painel de navegação de formulários indicado pela letra “A” da Figura 4.17.

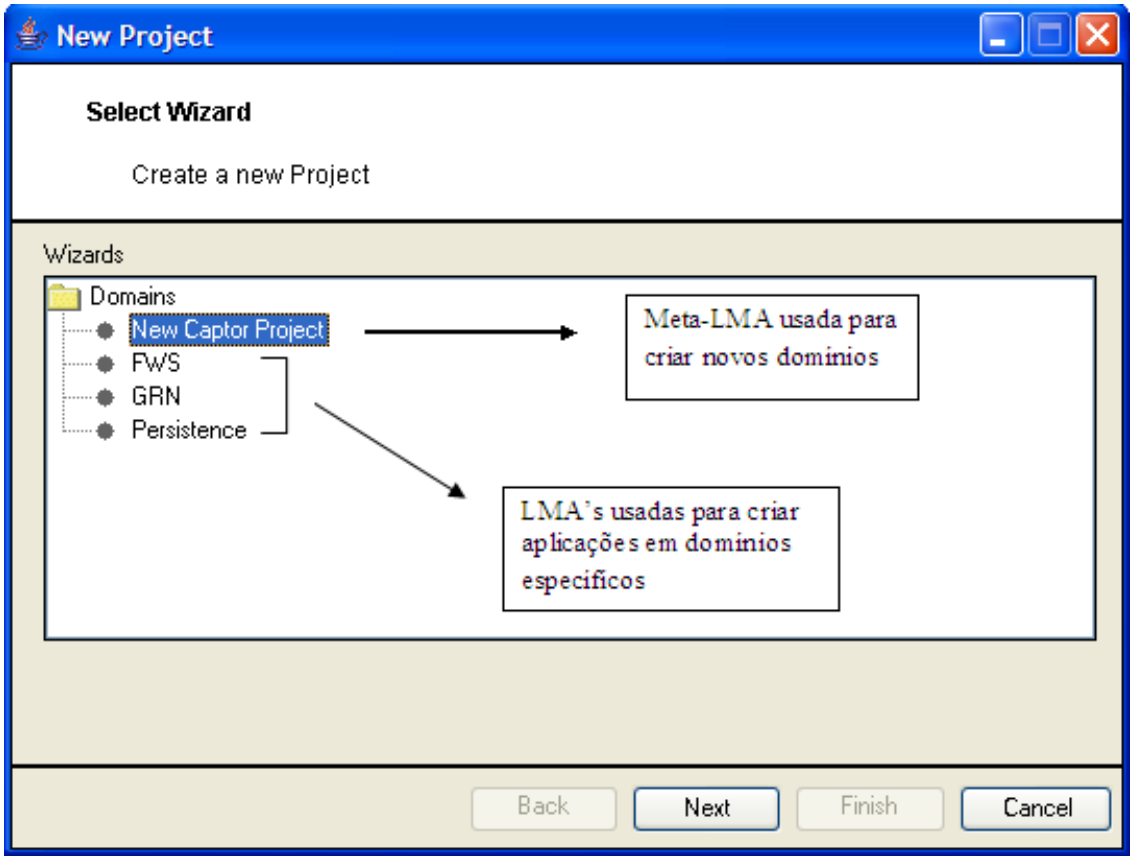

Figura 4.16: Criação de um novo projeto

Clicando com o botão esquerdo do *mouse* nos itens dessa árvore, o conteúdo dos formulários é exibido no painel central.

Para iniciar o processo de edição, o engenheiro de domínio deve selecionar o formulário raiz e inserir uma breve descrição do projeto na caixa de texto localizada no formulário de descrição de projeto indicada pela letra "B" da Figura 4.17.

Para continuar o processo, o engenheiro de domínio deve utilizar o diagrama hieráquico de formulários, construído na etapa de definição da LMA, para contabilizar quantos formulários são necessários na especificação. Sabendo esse número, ele deve adicionar esses formulários na Captor para que sejam referenciados mais adiante. Como indicado pela letra "A" da Figura 4.18, para adicionar um formulário na especificação, o engenheiro de domínio deve clicar com o botão direito no formulário de descrição de projetos `Captor Project` e selecionar o menu `Insert Form->New Form`. Na Figura 4.19, a letra "A" ilustra o resultado da inserção de três formulários na especificação. A Captor adiciona automaticamente um identificador nos formulários, assim o primeiro formulário recebe o identificador 1.1, o segundo 2.1, o terceiro 3.1 e assim sucessivamente. No caso de algum formulário ter mais de uma variante, o segundo número do identificador é incrementado, por exemplo, no caso do formulário 1.1, as variantes seriam 1.2, 1.3, 1.4 e assim sucessivamente.

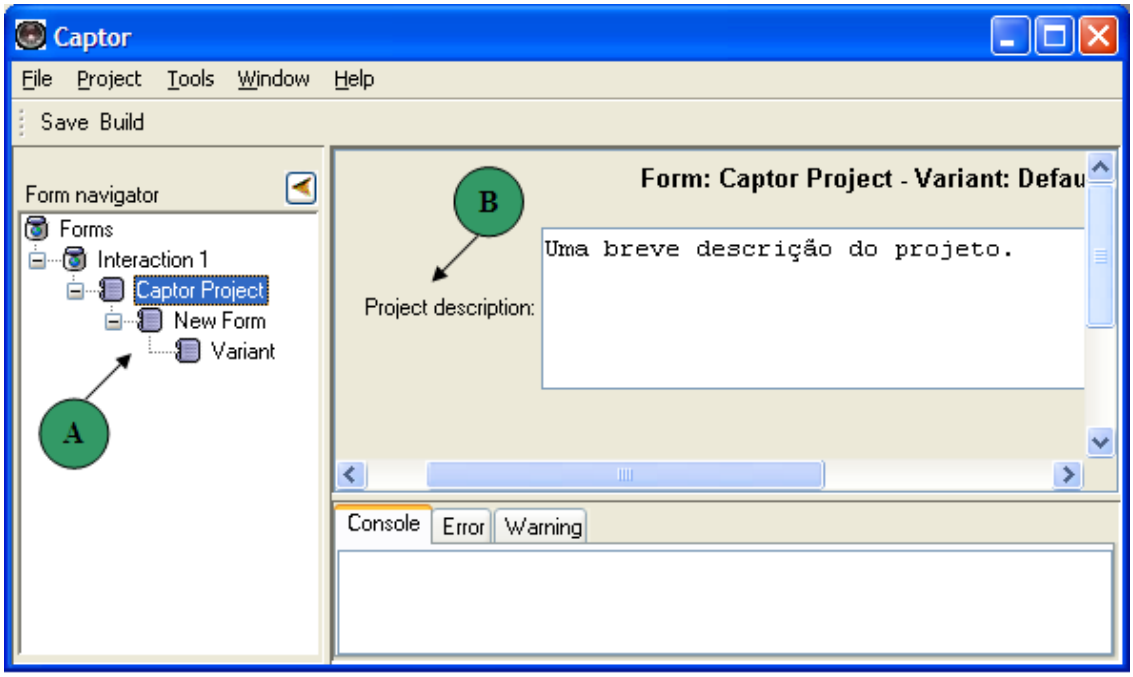

Figura 4.17: Edição da especificação

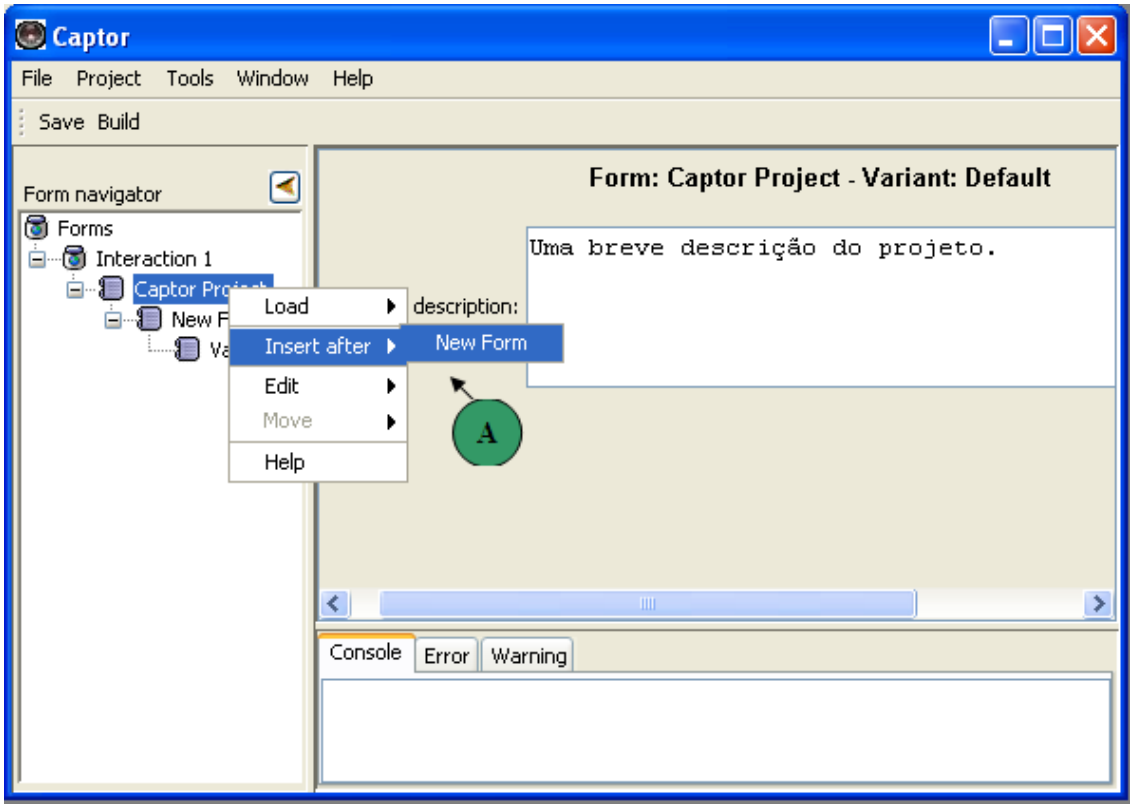

Figura 4.18: Inserção de novos formulários

Após inserir todos os formulários, o engenheiro de domínio deve editar os dados de cada um deles individualmente. O processo de edição dos formulários de especificação de formulários consiste na definição do nome do formulário (letra A da Figura 4.20) e na criação dos relacio-

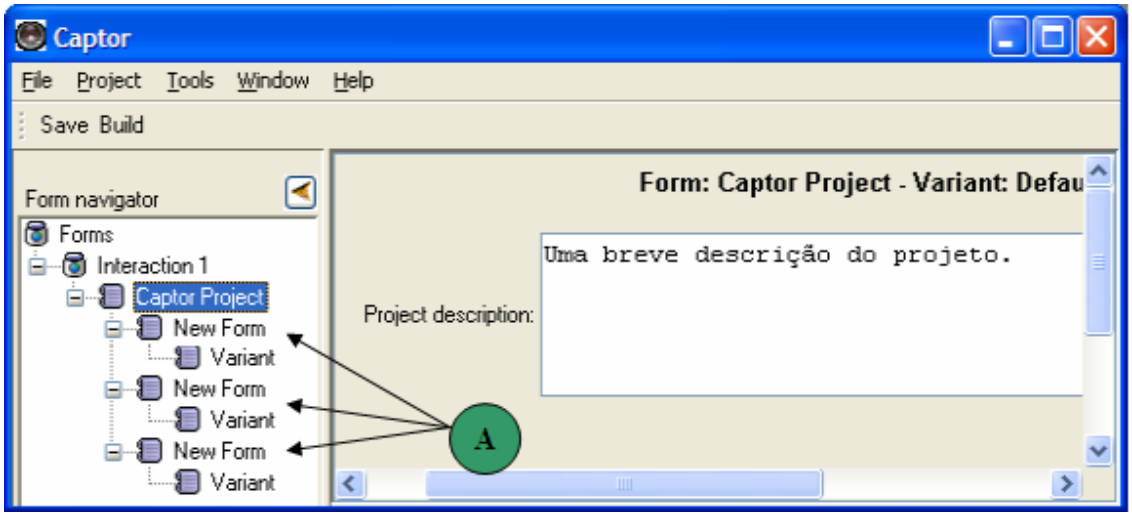

Figura 4.19: Novos formulários

namentos entre os formulários (definir quais são os formulários filhos de cada formulário, letra B da Figura 4.20). Na definição das variantes, deve-se especificar um nome para a variante, os elementos de formulários da variante e o texto de ajuda para essa variante.

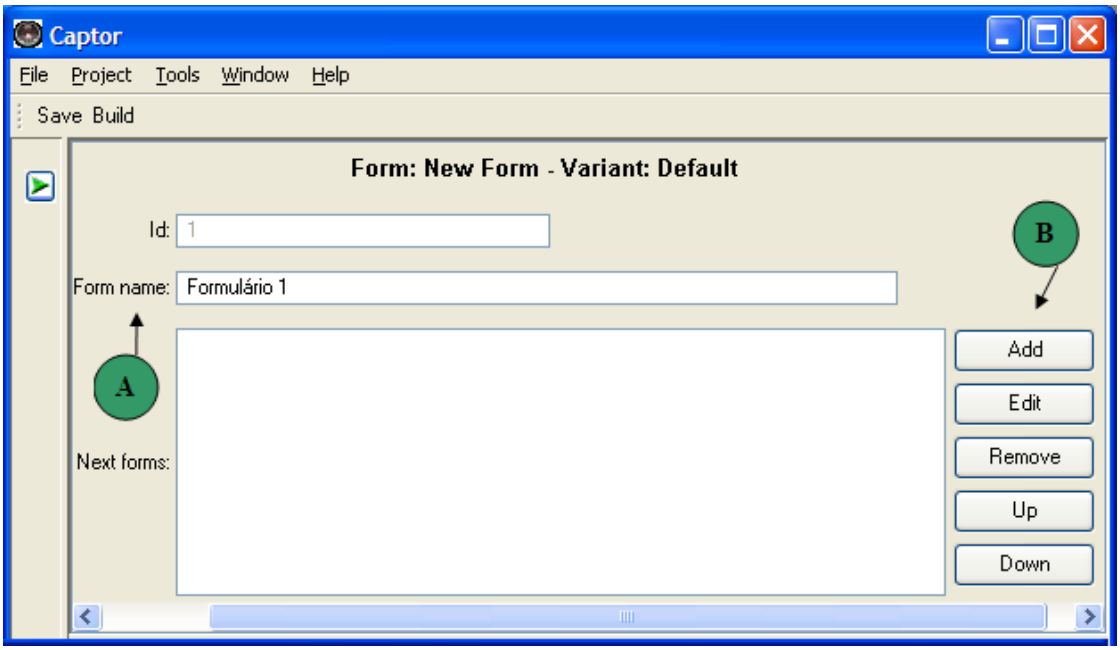

Figura 4.20: Edição de um novo formulário

O botão `Add` do elemento `Next Forms` abre a janela apresentada na Figura 4.21. Nessa janela, o usuário deve selecionar os formulários que são filhos do formulário que está sendo editado na réplica da árvore de especificação indicada pela letra “A”. Quando o usuário clica nessas réplicas, os detalhes do formulário são exibidos no painel do lado direito indicado pela letra “B” e as cardinalidades mínimas e máximas podem ser especificadas nas caixas de seleção indicadas pelas letras “C” e “D”, respectivamente.

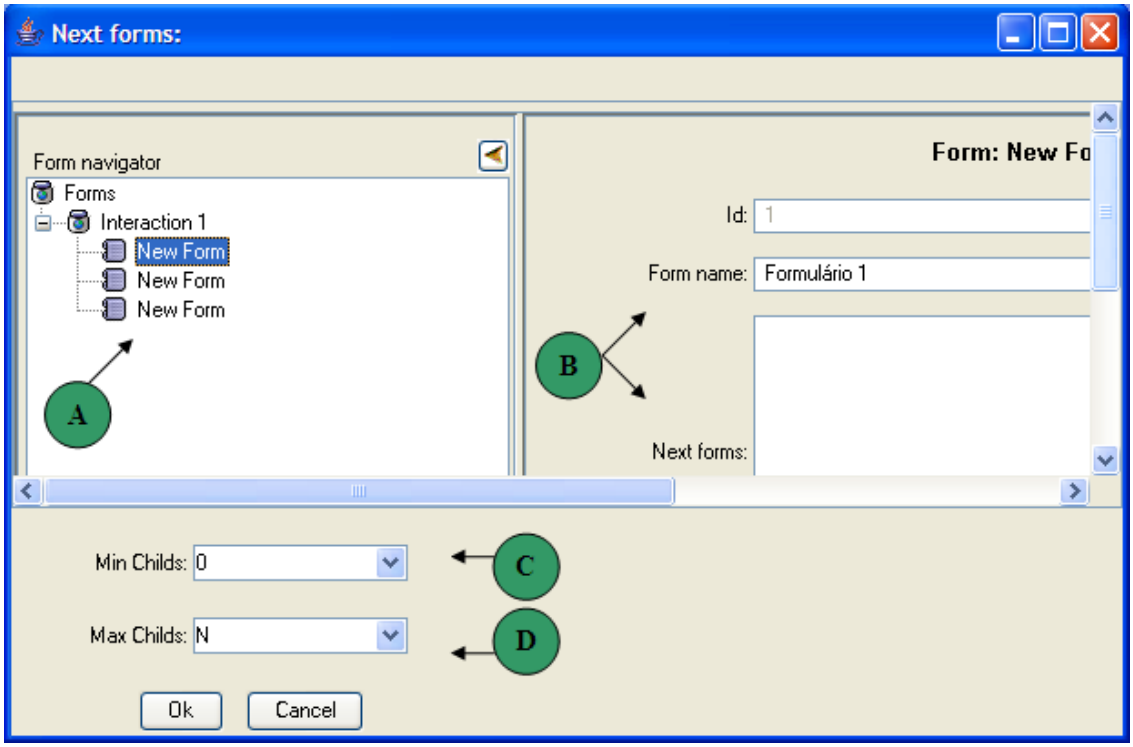

Figura 4.21: Edição dos formulário filhos

Após a edição dos relacionamentos entre os formulários, o engenheiro de domínio pode realizar a edição de cada variante. O formulário de edição das variantes é apresentado integralmente na Figura 4.22 e os principais elementos desse formulário são: elemento de definição de nome da variante, elementos de definição de elementos da variante e texto de ajuda, indicados pelas letras “A”, “B” e “C” respectivamente. O elemento que representa os elementos de formulário deve conter a definição de todos os elementos de formulário que a variante deve apresentar e o campo de texto de ajuda deve conter um texto para ser mostrado para o usuário caso ele solicite o detalhamento desse formulário durante a utilização da Captor.

Para adicionar um elemento em um formulário, o engenheiro de domínio deve clicar no botão `Add` (indicado pela letra “B” da Figura 4.22) do elemento de definição de elementos de formulário. Esse botão abre a janela apresentada na Figura 4.23 e as letras “A”, “B” e “C” mostradas nessa Figura indicam respectivamente: o seletor de elementos de formulário, o painel de parâmetros desse elemento e o botão que testa o elemento. Nessa janela, o engenheiro de domínio pode adicionar os elementos de formulário, tais como: caixas de texto, caixas de seleção e tabelas.

Como ilustrado na Figura 4.24, após a edição de todos os formulários, o engenheiro de domínio pode validar e gerar os artefatos desse projeto. O processo de validação realiza uma sequência de verificações que garantem que a especificação está correta. Dentre as verificações, as mais importantes são: verificação de preechimento de todos os campos obrigatórios, verificação da sintaxe do nome dos formulários e variantes e verificação dos ponteiros para os filhos

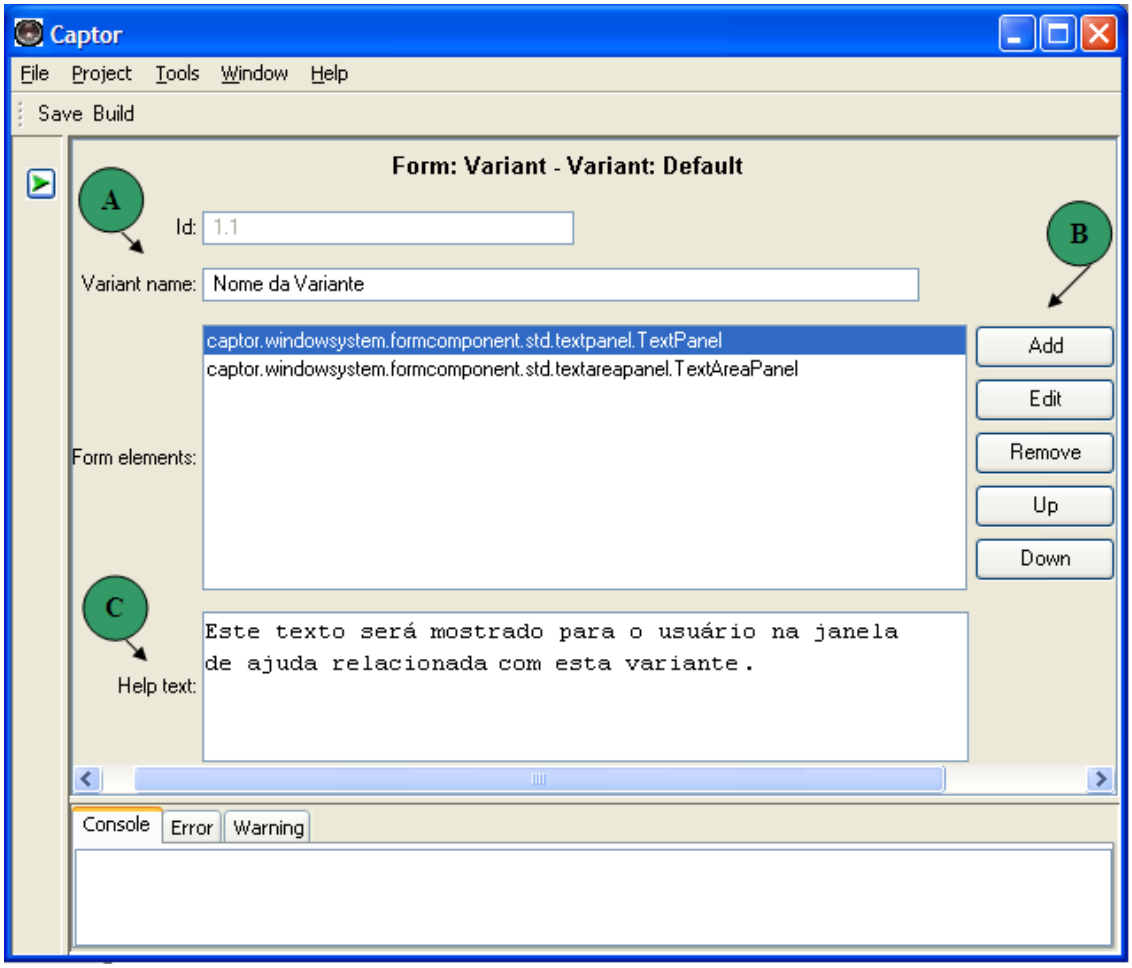

Figura 4.22: Edição de variantes

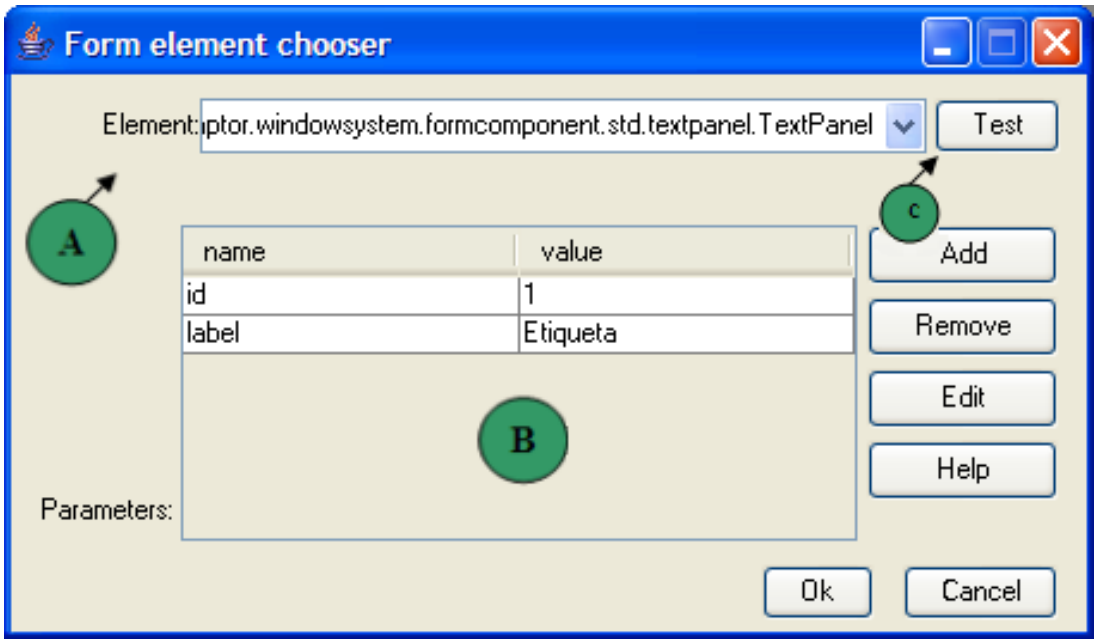

Figura 4.23: Adição dos elementos de formulário nas variantes

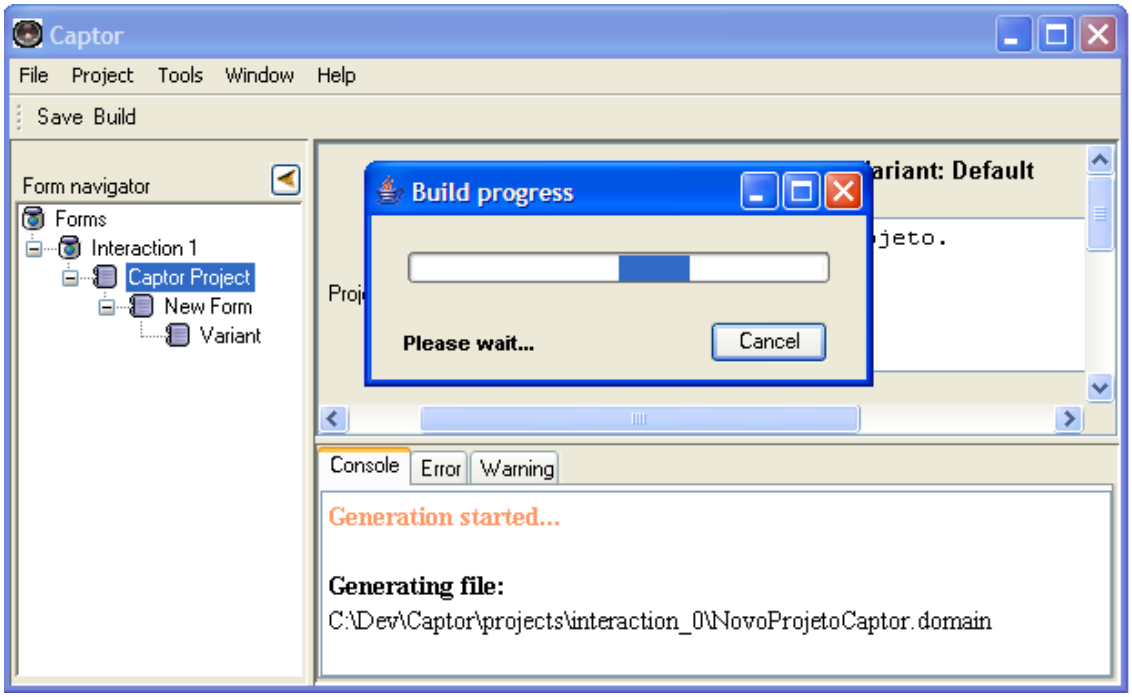

Figura 4.24: Geração do arquivo de configuração

dos formulários. Quando a Captor detecta um erro na especificação, ela emite um aviso para o usuário. Na Figura 4.25 é apresentado o comportamento que a Captor apresenta quando ela detecta um erro na especificação: a letra "A" indica qual formulário emitiu o erro, a letra "B" indica qual elemento do formulário está incorreto e a letra "C" mostra informações detalhadas sobre o erro que foi detectado[4].

Após o usuário pressionar o botão `Save` e `Build`, a Captor pode gerar automaticamente o arquivo de configuração de interface no formato XML. Após o processo de geração, o processo de pós-geração é acionado e o novo arquivo de configuração é implantado automaticamente na Captor. Para utilizar a Captor no novo domínio, basta selecionar o menu `File->New->Project` e selecionar o novo domínio na árvore de seleção de domínios. Os formulários serão apresentados pela Captor e o engenheiro de aplicação já poderá editar, validar e persistir a especificação para que ela possa ser recuperada na reabertura do projeto.

O processo gráfico da modelagem de uma LMA para a Captor com a própria Captor possui diversas vantagens em relação ao processo manual. Nesse modelo, o engenheiro de domínio tem à sua disposição um editor que permite realizar diversas modificações na especificação, utilizando para isso apenas os elementos gráficos da Captor, diversos mecanismos de validação da especificação e implantação automática do arquivo de configuração após a geração. Se o engenheiro de domínio desejar realizar o processo manual, ele deve criar um arquivo XML no formato adequado, validá-lo com a ferramenta de validação de configuração MMV (do inglês

[4]Note que essa funcionalidade está disponível em todos os domínios em que a Captor oferece apoio.

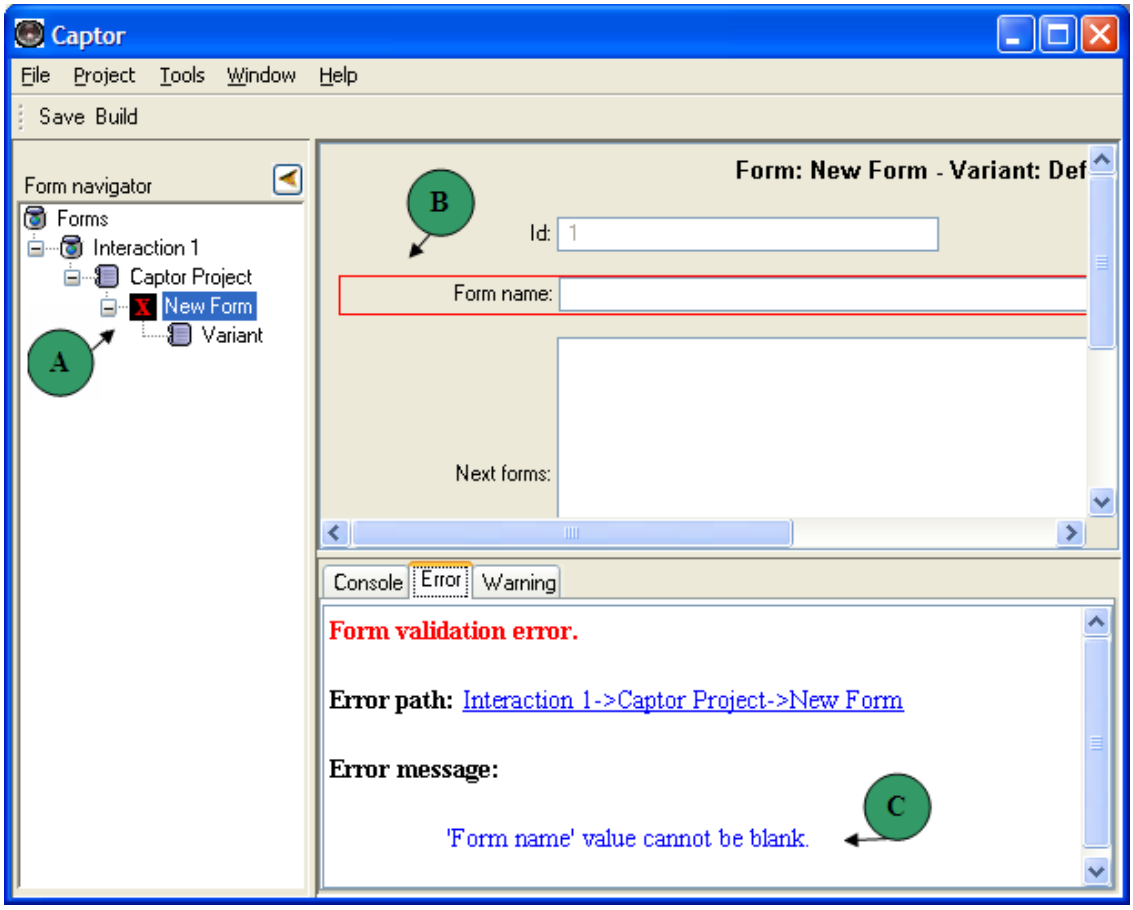

Figura 4.25: Detecção de erros na especificação

Meta-Model Validator), também implementado neste trabalho e disponibilizado no menu de opções da Captor, e colocá-lo no diretório de domínios de instalação da Captor.

Note que tanto o formulário de criação de formulários como o formulário de criação de variantes possuem um elemento de formulário especializado, ou seja, os elementos `Next Forms` e `Form Elements` respectivamente, projetados especialmente para a modelagem de LMAs.

O formulário de criação de formulários contém o elemento especial `Next Forms`, que abre a janela para o usuário inserir quais são os formulários filhos do formúlário que está sendo especificado. O formulário de criação de variantes contém o elemento especial `Form Elements`, que habilita o engenheiro de domínio a escolher, editar e testar graficamente os diversos elementos de formulários disponíveis na Captor.

A criação de elementos de formulário personalizados foi realizada em um dos três estudos de caso apresentados neste trabalho: na instanciação do framework GREN detalhada no Capítulo 5. Embora seja possível modelar LMAs para esse domínio com os elementos padrão (caixas de texto, seleção e tabelas), a introdução desses elementos personalizados melhorou a apresentação gráfica da Captor por disponibilizar telas com elementos gráficos específicos de domínio.

A criação de elementos de formulários personalizados é uma tarefa que envolve pouca codificação. A maior parte da funcionalidade requerida por um elemento de formulário está implementada em um framework interno da Captor. Quando o engenheiro de domínio deseja criar um novo elemento de formulário, ele deve estender uma classe abstrata, implementar alguns métodos gancho e criar a interface gráfica do elemento. Após a criação do elemento, ele pode usar a Captor sem necessidade de nenhuma codificação ou alteração no código da Captor. O manual de criação de elementos de formulário pode ser encontrado no pacote de instalação da Captor disponível no site `http://www.labes.icmc.usp.br/captor`.

4.3.4.2 Criação dos gabaritos

A Captor armazena as informações inseridas na sua interface gráfica em uma representação textual no formato XML. A estrutura do XML que a Captor utiliza para persistir a especificação possui uma parte fixa (independente de domínio) e uma parte variável (dependente de domínio). Na Listagem 4.1 é apresentada a parte fixa desse XML.

Listagem 4.1: Estrutura fixa da especificação armazenada em XML

```
<?xml version="1.0" encoding="UTF-8"?>

 <formsData>

  <project>
    <name>Nome do projeto</name>
  </project>

  <forms>

    <form id="1.1" variant="Default">

      <data>
        <!—
        Os dados armazenados pelos elementos de
        formulário 1.1 são armazenados nesta linha
        —>
      </data>

      <form id="2.1" variant="Default">

        <data>
          <!—
          Os dados armazenados pelos elementos de
          formulário 2.1 são armazenados nesta linha
          —>
        </data>

      </form>

    </form>

```

```
33   </forms>
34
35 </formsData>
```

Como ilustrado na Listagem 4.1, a estrutura fixa de XML que a Captor produz começa com a marcação raiz formsData. Dentro dessa marcação existem duas marcações filhas: `project` e `forms`. A marcação `project` (linha 5) contém o nome do projeto e a marcação `forms` (linha 9) contém os dados dos formulários da árvore dos formulários da especificação. A marcação `forms` contém uma marcação raiz `form` (linha 11). Essa marcação representa um formulário e contém uma marcação filho `data` (linha 13) e zero ou mais marcações filhas `forms` (nesse caso, uma marcação na linha 20). As marcações `data` representam os dados de cada formulário e as marcações `forms` representam os filhos de cada formulário.

A parte variável da estrutura representa os dados contidos nos elementos de cada formulário e são representados pelas marcações `data`, filhas da marcação `form` (linhas 11 e 20). Dependendo do número e do tipo de elementos que são utilizados para definir um formulário, o conteúdo dessas marcações pode variar. Na Listagem 4.2 é apresentada a estrutura em XML da parte variável de um formulário com dois elementos: uma caixa de texto e uma caixa de seleção.

Listagem 4.2: Estrutura variável do formulário 1.1

```
1 ...
2 <form id="1.1" variant="Default">
3  <data>
4
5   <textatt name="textId">
6    Valor inserido na caixa de texto
7   </textatt>
8   <combo name="comboId">
9    Valor selecionado da caixa de seleção
10  </combo>
11
12  </data>
13 </form>
14 ...
```

Normalmente deve haver um gabarito para cada artefato que deve ser gerado [5] e com base no diagrama de mapeamento de variabilidades é possível determinar quais definições do projeto vão afetar qual gabarito. Para realizar o desenvolvimento dos gabaritos são utilizados como entrada os diagramas de arquitetura, projeto, o modelo de implementação e os diagramas (ou tabelas) de mapeamento de variabilidades. Para realizar a implementação dos gabaritos, o engenheiro de domínio deve codificar os gabaritos de maneira similar ao modo como os programadores normalmente traduzem diagramas e especificações em código e essa implementação (descrita brevemente na Seção 3.4.4.1) está fora do escopo deste trabalho.

[5] Note que um gabarito pode ser composto por 1 ou mais gabaritos.

A linguagem de transformação de gabaritos que a Captor utiliza é chamada XSL (XSL, 2005). Essa linguagem de transformação de gabaritos é padronizada pela entidade W3C (W3C, 2006) e pode ser utilizada para geração de HTML em páginas Web com conteúdo dinâmico, transformação de documentos de um formato para outro e geração de código, entre outras aplicações.

A linguagem XSL permite asserções condicionais, interações, composição de gabaritos e disponibiliza um conjunto de funcionalidades, tais como: funções para manipulação de cadeias de caracteres e formatação de texto. A sintaxe e a semântica detalhada dessa linguagem está fora do escopo deste trabalho, mas instruções detalhadas sobre ela podem ser encontradas no site: `http://www.w3c.org/xslt`, em livros e manuais disponíveis na Internet.

Desenvolvimento de gabaritos com Zonas de Segurança: Quando os arquivos são gerados por uma ferramenta que não oferece nenhuma funcionalidade de proteção de código, os arquivos produzidos não podem ser modificados manualmente, caso contrário, se a ferramenta regerar o código, as modificações manuais serão perdidas durante o processo.

Para não perder as modificações manuais implementadas nos artefatos gerados, a Captor fornece apoio para a criação de zonas de segurança, tais como as apresentadas na Seção 2.4.2. As zonas de segurança implementadas nos gabaritos da Captor devem ser escritas conforme apresentado na Listagem 4.3: as linhas 1 e 3 indicam o início e o fim da zona de segurança e a linha 2 indica como o conteúdo da zona deve ser armazenado.

Listagem 4.3: Exemplo de criação de zonas de segurança nos arquivos de *templates*

```
1 // START–SAFE(ZoneId)
2   <xsl:value-of select="/data/safezone[@id=ZoneId]"/>
3 // END–SAFE
```

Quando o arquivo é gerado, as linhas 1 e 3 aparecem no artefato produzido e o desenvolvedor pode inserir código manualmente nesse espaço. Se a Captor for regerar o artefato, ela primeiro analisa se já existe um artefato gerado e em caso positivo, ela lê esse arquivo, busca por zonas de segurança e armazena em memória as linhas contidas entre o início e o fim da zona de segurança. Durante o processo de regeração, a Captor insere esses dados armazenados nos artefatos regerados, fazendo com que o desenvolvedor tenha a impressão que foram modificadas apenas as áreas fora da zona de segurança.

Como ilustração, na Listagem 4.4 é apresentado um exemplo da zona de segurança produzida em um arquivo com código Java gerado pela Captor. As modificações manuais devem ser inseridas pelo engenheiro de aplicação abaixo da linha 3 e acima da linha 5. Note que cada zona de segurança deve possuir um identificador único (o identificador da listagem é definido como `someSafeZoneId`).

Listagem 4.4: Exemplo de zonas de segurança em arquivos Java

```
1 public void algumMetodo() {
2
3 // START–SAFE(someSafeZoneId)
4   System.out.println("Olá zona de segurança!");
5 // END–SAFE
6
7   return;
8 }
```

4.3.4.3 Criação do arquivo de mapeamento de transformação de gabaritos

A especificação de uma aplicação em um domínio particular pode apresentar diversas características opcionais e obrigatórias, e essas características são implementadas por um ou mais gabaritos. Por exemplo, como indicado pela letra A da Figura 4.26, o modelo da aplicação contém três formulários de definição de classes persistentes. Para cada definição de classe persistente, a Captor deve passar como parâmetro para o processador de gabaritos o trecho da especificação com a classe persistente e o gabarito das classes persistentes. Para a geração do *script* de criação de tabelas SQL, a Captor deve passar como parâmetros o modelo completo e o gabarito de *scripts*.

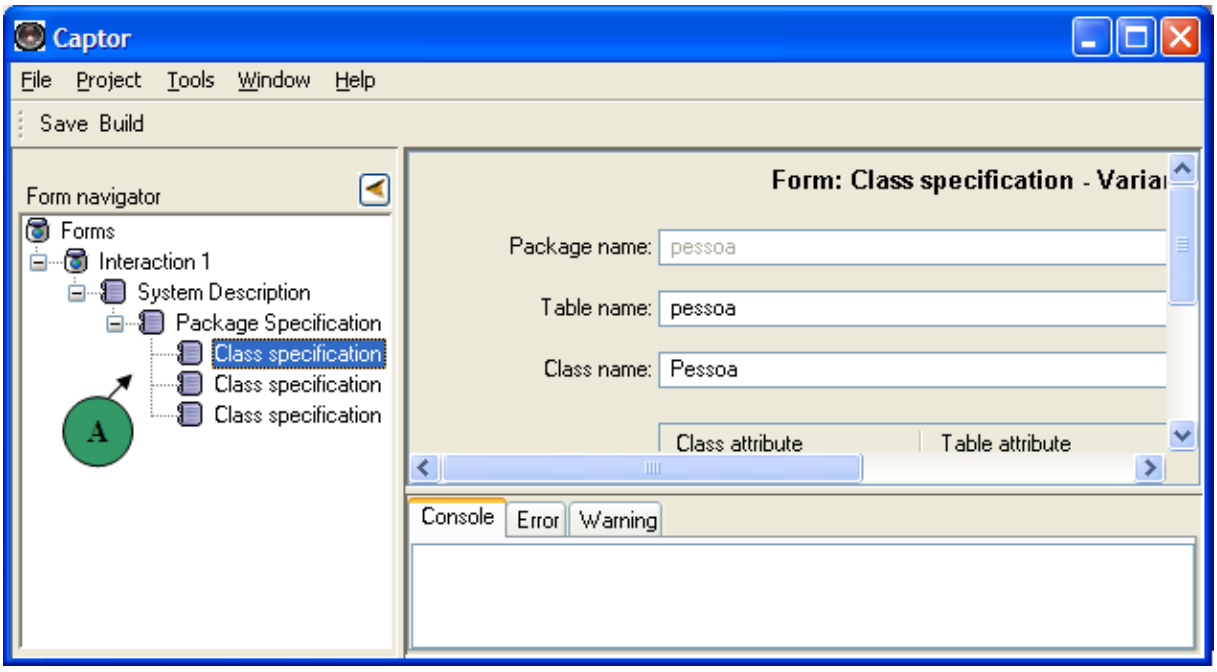

Figura 4.26: Formulários do domínio de persistência

O arquivo de mapeamento de transformação de gabaritos é utilizado pela Captor para indicar, com base no modelo da aplicação, quais gabaritos devem ser utilizados no processo de geração de artefatos. Esse arquivo controla o processo de geração de artefatos fazendo diver-

sas asserções no modelo da aplicação, com o objetivo de determinar quais gabaritos devem ser processados.

Para realizar o mapeamento das transformações, foi criada neste trabalho a linguagem MTL (do inglês *Mapping Transformation Language*). Essa linguagem é definida em XML e possui construções gramaticais especialmente projetadas para agendar as transformações da Captor. Na Listagem 4.5 é apresentado o arquivo MTL do estudo de caso de persistência.

Listagem 4.5: Arquivo de mapeamento de gabaritos

```
<composer name="Captor">

 <main>

  <callTask id="sql">

  <for-each select="/formsData/forms/form/form/form">
   <if test="equal(/formsData/forms/form/form/form/@id,'3.1')">
    <callTask id="class"/>
   </if>
  </for-each>

  <for-each select="/formsData/forms/form/form/form">
   <if test="equal(/formsData/forms/form/form/form/@id,'3.1')">
    <callTask id="interface"/>
   </if>
  </for-each>

 </main>

 <tasks>

  <task id="sql">
   <compose>
    <template>sql.xsl</template>
     <newFilename>scripts.sql</newFilename>
   </compose>
  </task>

  <task id="class">
   <compose>
    <template>class.xsl</template>
    <newFilename>
     ${formsData/current/form/data/textatt[@name='packageName']}/
     ${formsData/current/form/data/textatt[@name='className']}
     .java
    </newFilename>
   </compose>
  </task>

  <task id="interface">
   <compose>
    <template>interface.xsl</template>
    <newFilename>
     ${formsData/current/form/data/textatt[@name='packageName']}/
     I${formsData/current/form/data/textatt[@name='className']}
```

```
47        .java
48      </newFilename>
49    </compose>
50   </task>
51
52  </tasks>
53
54 </composer>
```

O arquivo da Listagem 4.5 possui duas marcações abaixo da marcação raiz `composer`: a marcação `main` (linha 3) e a marcação `tasks` (linha 21). A marcação `tasks` pode ter uma ou mais marcações `task` (linhas 23, 30 e 41). A marcação `task` é utilizada para definir a transformação de um gabarito em um arquivo de saída. Essa marcação deve possuir uma marcação `compose` (linhas 24, 31 e 42) e cada marcação `compose` é formada pelo nome do arquivo de gabarito (linhas 25, 32 e 43) e pelo nome do arquivo de saída que esse gabarito transforma (linhas 26, 33 e 44). O caminho de diretório dos arquivos de gabaritos deve ser indicado de forma relativa ao diretório de gabaritos do domínio (ex: /install_dir/domains/nome_do_dominio/templates) e o nome do arquivo que deve ser gerado (marcação `newFileName`) pode conter expressões XPATH (XSL, 2005) para gerar nomes de arquivos personalizados. Por exemplo, na linha 34, o nome do arquivo gerado será o nome contido na marcação "/formsData/current/form/data/textatt[@packagename]" da especificação, concatenado com a cadeia de caracteres "/", concatenado com a marcação "/formsData/current/form/data/textatt[@packagename]", concatenado com a cadeia de caracteres ".java". Como ilustração, o nome do arquivo de saída da especificação de classe persistente do formulário da Figura 4.26 seria "output_dir/pesssoa/Pessoa.java" em que a cadeia de caracteres "output_dir" deve ser substituída pelo diretório de saída indicado durante a criação do projeto na Captor[6].

A execução da linguagem é iniciada na marcação `main` e essa marcação pode conter três tipos de marcações: marcação `callTask`, `if` ou `for-each`. A marcação `callTask` indica para a Captor que uma determinada tarefa deve ser executada (chamadas para a marcação `task` nas linhas 5, 9 e 15).

As marcações `if` são utilizadas para realizar assertivas sobre o conteúdo da espeficação que está sendo processada. Por exemplo, na Listagem 4.6 são apresentados três cláusulas condicionais.

Listagem 4.6: Cláusulas condicionais no arquivo de mapeamento de *templates*

```
1 <if test="exist(/data/forms/form)">
2   <callTask id="processForm1"/>
3 </if>
4 <if test="equal(/data/forms/form@id='1.1')">
```

[6]Note que o arquivo de saída também pode ser configurado com nomes absolutos. Para isso, no Linux basta iniciar o nome do arquivo com "/", por exemplo "/home/user/nome_do_arquivo" e, no Windows, iniciar o nome do arquivo com "C:\", por exemplo: "C:\nome_do_arquivo".

```
5   <callTask id="processForm1.1"/>
6 </if>
7 <if test="not-equal(/data/forms/form/='1.1')">
8   <callTask id="processForm2"/>
9 </if>
```

A primeira cláusula da Listagem 4.6 (linha 1) só é executada se existir o caminho "/data/-forms/form" no arquivo-fonte (XML gerado a partir da especificação da aplicação), ou seja, a cláusula só é executada se o formulário inicial foi preenchido pelo engenheiro de aplicação. A segunda cláusula condicional (linha 4) só é executada se o atributo "id" da marcação "/data/-forms/form" for igual à cadeia de caracteres "1.1". A terceira cláusula condicional (linha 7) só é executada se o atributo "id" da marcação "/data/forms/form" for diferente da cadeia de caracteres "1.1".

O teste realizado pela função `exists` recebe como argumento uma expressão XPATH. O teste realizado pela função `equal` e `not-equal` recebe como parâmetros dois argumentos, que podem ser uma expressão XPATH ou uma cadeia de caracteres delimitada por aspas simples. Se a função da cláusula `if` que está sendo processada for avaliada como verdadeira, as instruções dentro do `if` são executadas, caso contrário essas instruções são ignoradas.

As marcações `for-each` são utilizadas para realizar iterações durante o processo de geração de artefatos. Na Listagem 4.7 são apresentados exemplos dessas cláusulas.

Listagem 4.7: Cláusulas "for-each" no arquivo de mapeamento de *templates*

```
1 <for-each select="/data/forms/form/form">
2   <callTask id="processSomeForm"/>
3 </for-each>
4
5 <for-each select="/data/forms/form/form">
6   <if test="equal(/data/current/form/@id,'2.1')">
7     <callTask id="processSomeForm"/>
8   </if>
9 </for-each>
```

A marcação na linha 1 da Listagem 4.7 indica que a Captor deve fazer um *loop* de transformação em que a chamada callTask da linha 2 é executada o mesmo número de vezes que o número de formulários /data/forms/form/form existente do arquivo-fonte (arquivo da especificação). Além da iteração, essa cláusula disponibiliza o nó corrente da árvore de XML do arquivo-fonte de maneira personalizada por meio do caminho "/data/current/". Por exemplo, em cada iteração da linha 1 da Listagem 4.7, a Captor realiza uma cópia do nó /data/forms/-form/form no caminho /data/current/form. Essa abordagem é utilizada para que o engenheiro de domínio tenha a informação, dentro dos gabaritos e dentro do arquivo MTL, sobre qual nó está sendo processado.

As marcações dentro da marcação `main` devem obedecer uma ordem pré-determinada. Por exemplo, as marcações `callTask` devem ser definidas nas primeiras posições. Após as mar-

cações `callTask`, todas as marcações `ifTask` devem ser definidas. Após as marcações `ifTask`, todas as marcações `for-each` devem ser definidas. Essa ordem é a mesma dentro das marcações `ifTask` ou `for-each`, ou seja, dentro de uma marcação `ifTask` ou `for-each`, as instruções devem apresentar a mesma ordem (`callTask`, `ifTask` e `for-each`, respectivamente).

Como foi apresentado nesta Seção, a linguagem MTL foi desenvolvida para selecionar os gabaritos que devem ser utilizados durante o processo de geração. Essa linguagem foi projetada para ser simples e realizar a transformação de um número limitado de gabaritos. Atualmente, a linguagem possui três tipos de marcações de controle: `if`, `for-each` e `callTask`, e as definições das transformações estão modularizadas na marcação task.

Essa estrutura da linguagem permitiu que os três estudos de caso deste trabalho fossem realizados com sucesso, mas para utilizar a Captor em projetos maiores ou mais complexos (por exemplo, com mais de 200 gabaritos diferentes) e manter a organização do código, pode ser necessário uma linguagem mais complexa, que pode disponibilizar apoio para a criação de funções e procedimentos, variáveis, e outras funcionalidades normalmente encontradas nas linguagens imperativas.

4.3.4.4 Criação dos arquivos de pré e pós-processamento

Como apresentado na Seção 3.4.2, o código de uma aplicação pode ser composto por artefatos gerados e por artefatos reutilizáveis, tais como: componentes e frameworks. Após a geração dos artefatos, o engenheiro de domínio pode configurar a Captor, junto com a ferramenta Ant, para copiar automaticamente os artefatos reutilizáveis, armazenados em um diretório de domínio (diretório_de_instalação/domains/nome_do_domínio/lib), para o diretório de saída dos artefatos gerados ou outros diretórios específicos. Essa funcionalidade permite que a Captor possa disponibilizar automaticamente as dependências de código para os artefatos gerados e o engenheiro de aplicação possa compilar esses artefatos sem ter o conhecimento de como essas dependências devem ser ligadas aos artefatos gerados.

Além da cópia de arquivos, o Ant pode realizar: alteração ou remoção dos arquivos gerados para diretórios específicos, compilação automática de arquivos-fonte Java, manutenção automática em repositórios de controle de versões, tais como o CVS (CVS, 2006), implantação automática de artefatos em servidores de páginas web e servidores de aplicação e *backup* automático de versões antigas.

Para realizar a configuração do pré ou pós-processamento dos artefatos, o engenheiro de domínio ou engenheiro de aplicação deve criar um arquivo com nome pre-build.xml ou pos-build.xml localizados no diretório do domínio da Captor (ex: /diretório_de_instalação/ domains/persistencia/pre-build.xml) ou no diretório escolhido pelo usuário para armazenar os

dados do projeto corrente e inserir as linhas com as instruções necessárias de acordo com o manual do Ant. A simples presença desses arquivos no diretório correto faz com que o Ant realize as instruções contidas neles.

Antes de iniciar o pré e o pós-processamento, a Captor abre os arquivos de configuração do Ant e insere dinamicamente linhas contendo a declaração de variáveis que podem ser utilizadas nas instruções do Ant, tais como: caminho do diretório de instalação da ferramenta, caminho do diretório de arquivos do projeto corrente, caminho do diretório de armazenamento dos arquivos gerados e nome do projeto corrente. Exemplos comentados de arquivos de configuração de pré e pós processamento podem ser encontrados no pacote de instalação da Captor.

4.3.5 Documentação de domínio

A atividade de documentação de domínio não depende da implementação do gerador configurável e deve ser realizada de acordo com a Seção 3.4.5.

4.4 Engenharia de aplicação

A etapa de engenharia de aplicação da Captor não apresenta nenhuma diferença da etapa de engenharia de aplicação do gerador configurável genérico apresentado no Capítulo 3, ou mesmo de um gerador de aplicação específico. Em suma, para realizar essa atividade, o engenheiro de aplicação deve criar o modelo da aplicação com base na necessidade dos clientes, modelos de casos de uso ou linguagens de padrões. A Captor deve gerar os artefatos e o engenheiro de aplicação deve implementar as funcionalidades não cobertas pelo gerador, testar e manter o sistema.

4.5 Discussão

O desenvolvimento da Captor foi uma atividade que levou seis meses para ser completada e foi dividida em duas grandes etapas: o desenvolvimento de um gerador configurável por linguagens de padrões (3.5 meses) e a generalização dessa ferramenta para o gerador configurável (2.5 meses).

A Captor foi desenvolvida com a linguagem de programação Java (Horstmann, 2001b,a) e um conjunto de bibliotecas de componentes reutilizáveis e frameworks externos fornecidos pelo projeto Jakarta (Jakarta, 2006) e pela Sun Microsystems (Java, 2006). A arquitetura escolhida para o desenvolvimento das telas gráficas foi a Model-View-Controller (Buschmann et al., 2000) e foram utilizados diversos padrões de projeto na sua construção, tais como: Facade, Template-

Method e Observer (Gamma et al., 1995). A aplicação completa, com exceção dos artefatos externos e dos casos de testes, possui 264 classes e um total de 45.245 linhas de código.

Cada uma dessas etapas foi desenvolvida iterativamente de forma que, em cada iteração, uma ou mais funcionalidades foram adicionadas, testadas e validadas pelo grupo de pesquisa. O estudo de caso de instanciação automática do framework GREN (apresentado no Capítulo 5), foi utilizado como base para a criação da primeira versão e após o término do desenvolvimento e da bateria de testes realizadas com o framework JUnit (Gamma e Beck, 2005), os estudos de caso de persistência (apresentado neste Capítulo e no Capítulo 3) e bóias náuticas (apresentado no Capítulo 5) foram realizados.

Os dois últimos estudos de caso demostraram que a Captor poderia ser facilmente configurada para um domínio específico, requerendo apenas que o engenheiro de domínio criasse manualmente o arquivo de configuração de interface, os gabaritos, o arquivo MTL e os arquivos de pré e pós-processamento. Para facilitar ainda mais o processo de configuração na parte técnica, os autores decidiram criar uma meta-LMA para a definição de LMAs na própria Captor. Esse trabalho resultou no aumento da facilidade de uso da Captor, pois após esse trabalho, tornou-se possível gerar o arquivo de configuração da interface gráfica rapidamente, restando apenas o trabalho de criar o arquivo MTL, os gabaritos e os arquivos de pré e pós-processamento.

4.6 Considerações finais

Neste Capítulo foi apresentado o gerador de aplicação configurável Captor, o seu processo de desenvolvimento, engenharia de domínio e engenharia de aplicação. Esses processos são similares aos processos do gerador genérico apresentados no Capítulo 3, com exceção de algumas atividades específicas que foram detalhadas neste Capítulo por serem dependentes de implementação.

A Captor foi configurada e utilizada para realizar a geração de artefatos para três domínios diferentes: persistência, bóias náuticas e gestão de recursos de negócios. O domínio de persistência, por ser o mais simples e conhecido, foi utilizado para exemplificar as principais atividades do processo proposto e os domínios de bóias náuticas e gestão de recursos de negócios são apresentados no Capítulo 5.

CAPÍTULO

5

Estudos de caso

5.1 Considerações iniciais

Como foi apresentado no Capítulo 4, o gerador de aplicação configurável Captor pode ser configurado para fornecer apoio em domínios específicos e, como ilustração, foi apresentada a configuração e a utilização da Captor para o domínio de persistência de objetos. Neste Capítulo são apresentadas configurações da Captor para dois domínios diferentes: o domínio das bóias FWS e o domínio de gestão de recursos de negócio. Esses estudos de caso foram baseados em estudos de caso apresentados por outros autores que utilizaram geradores de aplicação específicos para gerar aplicações, tendo sido aqui modificados com o objetivo de configurar e utilizar a Captor para a geração de código das aplicações nesses domínios.

Este Capítulo está organizado da seguinte forma: na Seção 5.2 são apresentadas a etapa de engenharia de domínio, na qual a Captor é configurada para fornecer apoio para a geração de artefatos para uma linha de produtos de bóias náuticas, e a etapa de engenharia de aplicação, na qual a Captor é utilizada para gerar esses artefatos; na Seção 5.3 são apresentadas as etapas de engenharia de domínio, na qual a Captor é configurada para apoiar o domínio de gestão de recursos de negócio, e a etapa de engenharia de aplicação, na qual a Captor é utilizada para instanciar o framework GREN; na Seção 5.4 é apresentada uma comparação dos resultados da configuração e utilização da Captor nos domínios da gestão de recursos de negócio, bóias náuticas e persistência; finalmente, na Seção 5.5 são apresentadas as considerações finais deste Capítulo.

5.2 Linha de produtos de bóias náuticas - FWS

Weiss e Lai (1999) apresentam no livro “Engenharia de Software de Linhas de Produtos” um estudo de caso de bóias náuticas implementado com o processo de linhas de produtos FAST (ver Seção 2.5.1.1). Nesta Seção, esse exemplo é modificado para ser apoiado pelo gerador de aplicação configurável Captor.

5.2.1 Engenharia de domínio

As atividades de engenharia de domínio realizadas no domínio de bóias náuticas são apresentadas nas Seções a seguir.

5.2.1.1 Análise de domínio

Descrição do domínio:

As bóias náuticas FWS (do inglês *Floating Weather Station*) são utilizadas em alto mar para realizar 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.

Definição da terminologia comum:

A terminologia comum para o domínio FWS é apresentada na Tabela 5.1.

Tabela 5.1: Terminologia comum do domínio FWS

Termo:	**Significado**
Período do sensor	O número de segundos entre as leituras de cada sensor.
Período de transmissão	O número de segundos entre as transmissões das mensagens.
Velocidade do vento	A velocidade do vento em nós: milhas náuticas por hora.
Média ponderada	Dado um conjunto de pares valor/peso: (v1,w1), (v2,w2),...,(vn,wn) onde wi >= 0 para todos os i dentro de [1...N], a média ponderada é: (v1*w1 + v2*w2 + ... + vn*wn) / (w1+w2+...+wn)

Análise dos aspectos similares e variáveis:

Os aspectos similares e variáveis do domínio das bóias náuticas são apresentados nas Tabelas 5.2 e 5.3, respectivamente.

Tabela 5.2: Aspectos similares

Aspecto similar	**Descrição**
Transmissão de mensagens	Em intervalos de tempo fixos, a bóia FWS transmite mensagens contendo uma aproximação da velocidade corrente do vento na sua localidade.
Média ponderada	O valor da velocidade do vento transmitido é calculado pela média ponderada das leituras realizadas pelos sensores instalados na superficie da bóia e calculados por meio de várias leituras de cada sensor.
Sensores	A bóia FWS deve ser equipada com um ou mais sensores que monitoram a velocidade do vento.
Transmissor	A bóia FWS é equipada com um transmissor de rádio que envia mensagens para uma central de recepção de mensagens.
Driver	Cada sensor vem equipado com um *driver* de software que possui um identificador único.

Tabela 5.3: Variabilidades identificadas

Variabilidade	**Descrição**
Identificador único	Cada bóia deve possuir um identificador único que é utilizado como um dos campos das mensagens enviadas para a torre de controle. A torre de controle identifica qual bóia emitiu a mensagem por meio deste campo.
Fórmula	A fórmula utilizada para computar a velocidade do vento de cada sensor pode variar. Em particular, os pesos aplicados para os sensores de alta resolução e baixa resolução podem variar, e o número de leituras de cada sensor utilizadas na fórmula também pode variar.
Tipo de mensagem	Os tipos de mensagens que a bóia envia podem variar em conteúdo e formato.
Período do sensor	O período de cada sensor pode variar.
Período de transmissão	O período de transmissão pode variar.
Número de sensores	O número de sensores de vento pode variar.
Resolução dos sensores	A resolução de cada sensor pode variar.
Hardware	O *hardware* do sensor que monitora o vento e o seu *driver* podem variar.
Equipamento de transmissão	O *hardware* do equipamento de transmissão de mensagens e o seu *driver* podem variar.

Criação da linguagem de modelagem de aplicação:

A linguagem de modelagem de aplicações da família FWS será uma linguagem de configuração. Essa linguagem deve apoiar as definições de diferentes sensores e diferentes resoluções para cada sensor, diferentes períodos de transmissão de mensagens e leitura dos sensores, entre outros valores indicados na tabela de variabilidades do sistema.

Após a análise dos aspectos similares e variáveis e reuniões com especialistas no domínio, foram elicitadas todas as variabilidades que podem estar associadas aos membros da família FWS. Na Tabela 5.4 é apresentada a representação Tabular dessas variabilidades.

Tabela 5.4: Representação tabular da linguagem de modelagem de aplicações para o domínio FWS

Identificador	**Significado**	**Espaço de valores**	**Valor padrão**
ApplicationID	Um identificador único para cada membro FWS	[a-zA-Z0-9]	1
ApplicationTitle	Um título para a aplicação (utilizado somente no simulador)	[a-zA-Z0-9]	1
MsgType	Tipo da mensagem que é transmitida	{LONG,SHORT}	SHORT
HighResWeight	Peso utilizado no cálculo da média ponderada para os sensores de alta resolução	[1...100]	50
LowResWeight	Peso utilizado no cálculo da média ponderada para os sensores de baixa resolução	[1...100]	50
History	Número de leituras dos sensores utilizadas para o cálculo da média ponderada da velocidade do vento	[1...10]	5
SensorPeriod	Período dos sensores	[1...MaxSensorPeriod]	5
SensorRes	Resolução de cada sensor	{LOW,HIGH}	LOW
TransmitPeriod	Período de transmissão	[1...MaxTransmitPeriod]	10
SensorCount	Número de sensores de vento	[4...20]	4
SensorPeriod	Período de cada sensor	[1...600]	5
TransmitPeriod	Período de cada transmissão	[1...600]	5

Conforme apresentado na Seção 4.3.1.4, o primeiro passo para a criação de uma LMA baseada em formulários é definir uma estrutura hierárquica em forma de árvore. Existem várias formas de realizar essa tarefa no domínio de bóias náuticas FWS e na Figura 5.1 apresenta-se uma delas.

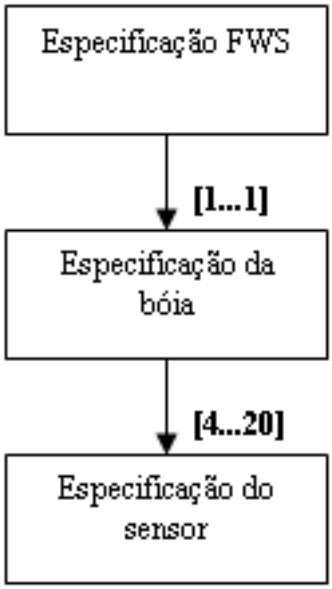

Figura 5.1: Árvore de formulários do domínio FWS

O formulário de especificação do sistema FWS é definido como o formulário raiz. Esse formulário deve ter um filho do tipo de especificação de bóia, com cardinalidade mínima de 1 e máxima de 1. O formulário de especificação de bóia deve ter no mínimo 4 e no máximo 20 formulários filhos de especificação de sensores.

O formulário de especificação FWS deve conter campos para a definição do identificador único e do título da bóia. O formulário de especificação da bóia deve conter os campos de especificação de tipo de mensagem, períodos de transmissão de mensagens e leitura dos sensores, tamanho do histórico e pesos aplicados na média ponderada dos sensores de alta e baixa resolução. O formulário de especificação de sensor deve ter um campo para a definição da resolução de um sensor. As tabelas 5.5, 5.6 e 5.7 apresentam os elementos de formulários e os seus respectivos parâmetros que devem ser utilizados para configurar os formulários da Captor no domínio FWS.

Tabela 5.5: Elementos do formulário de especificação FWS

Nome	**Elemento**	**Parâmetro**	**Valor**
ApplicationTitle	TextPanel	id	ApplicationTitle
		label	Application title
		use	required
		default_value	Floating Weather Station System
ApplicationId	TextPanel	id	ApplicationId
		label	Application id
		use	required
		default_value	1

Tabela 5.6: Elementos do formulário de especificação de bóia

Nome	**Elemento**	**Parâmetro**	**Valor**
Msg Type	ComboBoxPanel	id	MsgType
		label	Message type
		use	required
		default_value	SHORTMSG
		elements	SHORTMSG:LONGMSG
Transmit Period	ComboBoxPanel	id	TransmitPeriod
		label	Transmit Period
		use	required
		default_value	1
		elements	1...600
Sensor Period	ComboBoxPanel	id	SensorPeriod
		label	Sensor Period
		use	required
		default_value	5
		elements	1...600
History Length	ComboBoxPanel	id	HistoryLength
		label	History Length
		use	required
		default_value	5
		elements	1...10
High Resolution Weight	ComboBoxPanel	id	HighResolutionWeight
		label	High Resolution Weight
		use	required
		default_value	50
		elements	1...100
Low Resolution Weight	ComboBoxPanel	id	LowResolutionWeight
		label	Low Resolution Weight
		use	required
		default_value	50
		elements	1...100

Tabela 5.7: Elementos do formulário de especificação de sensor

Nome	**Elemento**	**Parâmetro**	**Valor**
Resolution	ComboBoxPanel	id	Resolution
		label	Resolution
		use	required
		default_value	Low
		elements	High:Low

5.2.1.2 Projeto de domínio

A realização da atividade de projeto de domínio tem o objetivo de projetar um sistema de software que apóie a criação de membros da família FWS. Para a realização deste estudo de caso foram criados dois simuladores do ambiente no qual as bóias são utilizadas. Esses simuladores são utilizados durante o processo de engenharia de aplicação para facilitar o desenvolvimento e testes do software das bóias FWS. O sistema completo (aplicação + simuladores) foi desenvolvido para ser executado de forma distribuída em uma arquitetura cliente/servidor. Um simulador representa uma torre de controle, outro simulador representa o vento e a aplicação FWS interage com esses dois simuladores durante a sua execução. Na bóia, cada sensor é implementado em uma linha de execução separada que coleta informações do vento de forma independente e armazena o valor lido no histórico de leituras. Na virada do ciclo do período de transmissão de mensagens, a bóia deve enviar a média dessas medições para a torre de controle. Na Figura 5.2 é mostrada esquematicamente a execução de um sistema FWS, o simulador de torre de controle e o simulador do vento.

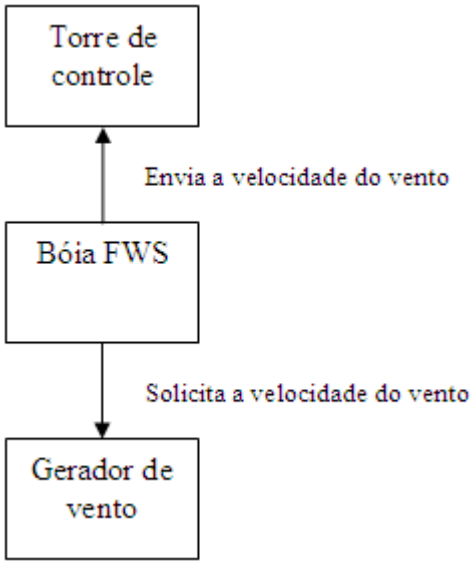

Figura 5.2: Ambiente de simulação das bóias FWS

O ambiente de simulação deve ser desenvolvido de forma distribuída, ou seja, os dois simuladores e a bóia devem ser executados em espaço de memória separados e a comunicação deve ser realizada por meio da troca de mensagens TCP/IP (Tanenbaum, 2002).

Durante a configuração do domínio FWS, o engenheiro de domínio decidiu modelar e implementar um conjunto de classes para realizar as funcionalidades invariáveis das bóias FWS, ou seja, realizar funcionalidades presentes em todas as bóias FWS (a parte fixa do domínio FWS). A criação dessas classes tem o objetivo de minimizar o número de artefatos de código que devem ser gerados e fazer com que as classes geradas reutilizem essas classes para realizar

parte das suas responsabilidades. Na Figura 5.3 é apresentado o diagrama de classes de uma aplicação no domíno FWS. As classes sombreadas são as classes da aplicação e as classes em branco são da parte fixa.

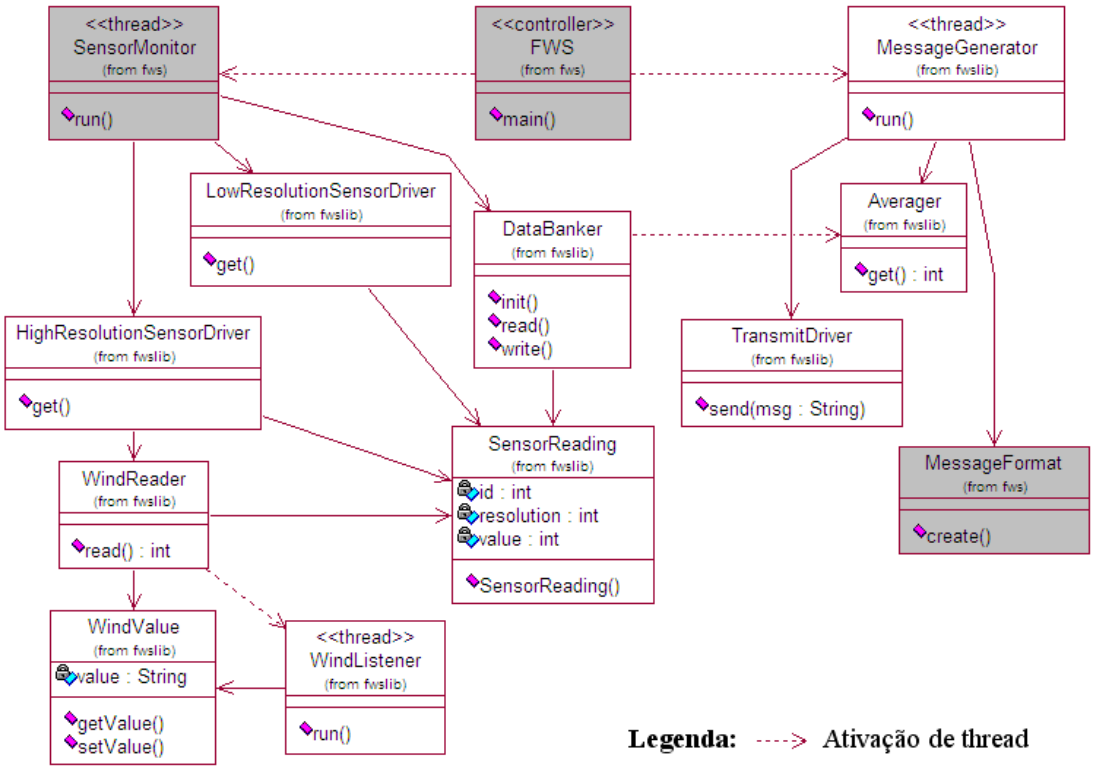

Figura 5.3: Diagrama de classes do domínio FWS

As classes da aplicação são as classes que são modificadas para a criação de um membro da família FWS. Por meio do desenvolvimento dessas classes, o engenheiro de aplicação deve adicionar as variabilidades necessárias durante a criação de um membro FWS. Por exemplo: a classe FWS é a classe controladora da aplicação. Essa classe contém algumas definições que representam as variabilidades de um membro FWS, tais como: a definição do intervalo de tempo entre as leituras dos sensores e o envio das mensagens, o tamanho do histórico de mensagens utilizado no cálculo da média e o identificador da aplicação. A classe `MessageFormat` cria uma mensagem no formato apropriado (mensagem curta ou longa). A classe `SensorMonitor` é responsável pelas instruções que utilizam os sensores para escrever a velocidade do vento no *buffer* que é utilizado para o cálculo da média. Dependendo do número e tipo dos sensores, essa classe deve executar as instruções apropriadas.

As classes da parte fixa realizam as tarefas rotineiras de uma aplicação FWS. Por exemplo: a classe `TransmitDriver` envia mensagens para a torre de controle, a classe `SensorReading` representa uma leitura de vento, a classe `Averager` calcula a média ponderada das medições da velocidade do vento detectada pelos diversos sensores, a classe `DataBanker` representa o histórico de leituras e possui um *buffer* para armazenar diversas leituras dos sensores de vento

e as classes `LowResolutionSensorDriver` e `HighResolutionSensorDriver` representam o *driver* para os dispositivos de *hardware* que realizam a leitura do vento (sensores de vento). Quando o software da bóia for retirado do ambiente de simulação e colocado no computador de bordo da bóia, algumas classes devem ser substituídas por classes com a mesma interface mas com implementações diferentes. Por exemplo: a classe `TransmitDriver` do simulador deve ser substituído por uma classe `TransmitDriver` que acompanha o transmissor de rádio utilizado na bóia e as classes do simulador `LowResolutionSensorDriver` e `HighResolutionSensorDriver` devem ser substituídas pelas classes que possuem as chamadas para os *drivers* específicos dos sensores das bóias.

Além do código-fonte das classes do sistema, o engenheiro de domínio decidiu que a Captor também deve produzir um diagrama de classes UML (FWS.xmi) com a representação de todas as classes do sistema e um caso de teste do framework JUnit (`MessageFormatTest.java`) para verificar a corretitude da geração das mensagens. O documento UML deve ser gerado com base no padrão de documentos XMI (OMG, 2005) e deve conter um diagrama de classes similar ao diagrama apresentado na Figura 5.3. O caso de testes deve verificar se a aplicação está gerando as mensagens no formato escolhido pelo engenheiro durante a modelagem da aplicação (SHORTMSG ou LONGMSG).

5.2.1.3 Implementação do domínio FWS

A realização da implementação do domínio FWS deve produzir o código do ambiente de simulação e as classes da parte fixa da aplicação utilizados na construção de aplicações do domínio FWS. Após a codificação, os artefatos devem ser testados e validados.

5.2.1.4 Configuração da Captor

A realização da atividade de configuração da Captor no domínio FWS deve produzir quatro artefatos: arquivo de configuração de interface e validação da especificação, gabaritos, arquivo MTL e arquivo de pós-processamento.

Arquivo de configuração de interface e validação da especificação

O engenheiro de domínio deve utilizar o modelo da árvore de formulários apresentado na Figura 5.1 e as Tabelas 5.5, 5.6 e 5.7, que representam os elementos de formulários, para configurar a interface e as regras de validação, conforme o processo apresentado na Seção 4.3.1.4.

Gabaritos

Deve ser criado, conforme processo apresentado na Seção 3.4.4.1, um gabarito para cada artefato que deve ser gerado. Conforme apresentado na Seção 5.2.1.2, os artefatos que devem ser gerados no domínio FWS são as classes FWS, MessageFormat e SensorMonitor, o arquivo UML representando as classes do sistema (FWS.xmi) e um caso de testes do framework JUnit (MessageFormatTest.java).

Conforme apresentado na Tabela 5.4, o tipo da mensagem enviada por uma bóia FWS pode ser SHORTMSG ou LONGMSG. Após reunião com especialistas do domínio, foi definido que o tipo SHORTMSG deve estar no formato da fórmula 5.1 e o tipo LONGMSG deve estar no formato da fórmula 5.2. A cadeia de caracteres “SHORTMSG” e “LONGMSG” das fórmulas 5.1 e 5.2, respectivamente, representam o tipo da mensagem, a cadeia de caracteres “FWS_ID” (apresentada nas duas fórmulas) representa o identificador da bóia, a cadeia de caracteres “WIND_SPEED” (apresentada nas duas fórmulas) representa a velocidade do vento e a cadeia de caracteres “MSG_NUM” (apresentada somente na fórmula 5.2) representa o número da mensagem. A classe MessageFormat deve produzir diferentes tipos de mensagens, dependendo da escolha que o engenheiro de aplicação fizer. Como exemplo de gabarito, na Listagem 5.1 apresenta-se o gabarito da classe MessageFormat.

$$“SHORTMSG : FWS\_ID : WIND\_SPEED” \tag{5.1}$$

$$“LONGMSG : FWS\_ID : WIND\_SPEED : MSG\_NUM” \tag{5.2}$$

Listagem 5.1: Gabarito do arquivo MessageFormat.java

```
1
2 <?xml version="1.0"?>
3
4 <xsl:stylesheet version="1.0"
5  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
6 <xsl:output method="text"/>
7
8 <xsl:template match="/">
9
10 <xsl:variable name="MsgType">
11  <xsl:value-of
12    select="/formsData/forms/form/form[@id='3.1']/data/combo"/>
13 </xsl:variable>
14
15 package fws;
16
17 import java.util.Vector;
18
19 public class MessageFormat {
20
21  static int msgNum = 0;
```

```
22
23   public static String create(int i) {
24
25    msgNum++;
26
27    String s = new String();
28    s = s.concat(FWS.MsgType);
29    s = s.concat(":");
30    s = s.concat(FWS.ID);
31    s = s.concat(":");
32    s = s.concat(new Integer(i).toString());
33    <xsl:if test="$MsgType=''LONGMSG'"'>
34    try {
35     Integer mn = new Integer(msgNum);
36     s = s.concat(":");
37     s = s.concat(mn.toString());
38    } catch (RuntimeException e) {
39     System.out.println(e);
40    }
41    </xsl:if>
42
43    return s;
44   }
45  }
46 </xsl:template>
47 </xsl:stylesheet>
```

Da linha 9 até a linha 13 da Listagem 5.1 define-se a variável `MsgType`, que contém o valor da escolha que o engenheiro de aplicação realizou sobre o tipo da mensagem (SHORT ou LONG). Da linha 14 até a linha 31 define-se a parte fixa da classe `MessageFormat`. Nesse trecho, o engenheiro de domínio definiu o nome do pacote da classe, o atributo `msgnum`, que representa um contador para as mensagens enviadas e o método create(int i), que recebe como parâmetro uma leitura da velocidade do vento e é responsável por criar a mensagem que será enviada para a Torre de Controle. Na linha 32, faz-se um teste para verificar se o tipo da mensagem escolhida pelo engenheiro de aplicação foi LONGMSG. Em caso positivo, o processador de gabaritos deve adicionar as linhas 33 até a 39 no arquivo gerado. Essas linhas são responsáveis por concatenar o número da mensagem no corpo da mensagem enviada para a Torre de Controle.

Na Listagem 5.2 apresenta-se um trecho do gabarito do arquivo FWS.java. Esse gabarito deve gerar diversas definições contidas na especificação, dentre elas as definições do período dos sensores e o período de transmissão de mensagens da bóia.

Listagem 5.2: Gabarito do arquivo FWS.java

```
1 ...
2 public static final int SensorPeriod = <xsl:value-of select=\\
3  "/form[@id='3.1]/data/combo[@name='SensorPeriod']"/> * 100;
4 public static final int TransmitPeriod = <xsl:value-of select=\\
5  "/form[@id='3.1]/data/combo[@name='TransmitPeriod']"/> * 100;
6 ...
```

Os gabaritos XSL utilizados na Captor podem fazer acesso aos dados da especificação armazenada no formato XML por meio do comando "<xsl:value-of select='**XPATH**'/>". Esses dados podem ser utilizados, como no exemplo da Listagem 5.1, para gerar trechos opcionais de código, ou como no exemplo da Listagem 5.2, para completar trechos de código com partes variáveis. A linguagem XSL fornece apoio na formatação dos dados (substring, sum, decimal-format, text, etc), estruturas de controle (if, choose, for-each), mecanismos para a composição de gabaritos (import, apply-templates, apply-imports, call-template, etc), funções (copy, element, current, sort, etc) e uma eficaz forma de acesso aos dados da especificação definida por meio de expressões XPATH.

Arquivo MTL

Conforme apresentado na Seção 4.3.4.3, o arquivo MTL deve selecionar os gabaritos que devem ser gerados com base no modelo da aplicação. Como todos os membros da família FWS devem conter os mesmos arquivos gerados e os mesmos arquivos reutilizados, o arquivo MTL deve fazer a chamada para o processamento de todos os gabaritos. Na Listagem 5.3 apresenta-se o arquivo MTL para o domínio FWS.

Listagem 5.3: Arquivo MTL para o domínio FWS

```
1  <composer name="Captor">
2
3   <main>
4    <callTask id="FWS"/>
5    <callTask id="MessageFormat"/>
6    <callTask id="SensorMonitor"/>
7
8    <callTask id="MessageFormatTest"/>
9    <callTask id="UML"/>
10  </main>
11
12  <tasks>
13
14   <task id="FWS">
15    <compose>
16     <template>fws/FWS.java</template>
17     <newFilename>src/fws/FWS.java</newFilename>
18    </compose>
19   </task>
20
21   <task id="MessageFormat">
22    <compose>
23     <template>fws/MessageFormat.java</template>
24     <newFilename>src/fws/MessageFormat.java</newFilename>
25    </compose>
26   </task>
27
28   <task id="SensorMonitor">
```

```
29      <compose>
30        <template>fws/SensorMonitor.java</template>
31        <newFilename>src/fws/SensorMonitor.java</newFilename>
32      </compose>
33    </task>
34
35    <task id="MessageFormatTest">
36    <compose>
37    <template>testcase/MessageFormatTest.java</template>
38    <newFilename>src/testcase/MessageFormatTest.java</newFilename>
39    </compose>
40    </task>
41
42    <task id="UML">
43      <compose>
44        <template>doc/Model.xmi</template>
45        <newFilename>doc/${formsData/project/name}.xmi</newFilename>
46      </compose>
47    </task>
48
49  </tasks>
50
51 </composer>
```

Arquivo de pós-processamento

Uma aplicação FWS completa é composta dos arquivos gerados, biblioteca de classes, caso de testes, documento UML e manual de referência das classes. A estrutura de diretórios de uma aplicação FWS é apresentada na Figura 5.4.

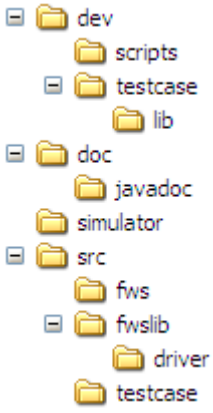

Figura 5.4: Estrutura de diretório dos arquivos de uma aplicação FWS

Durante a geração dos artefatos, a Captor produz os arquivos de código-fonte no diretório src\fws, o documento UML no diretório doc e o código-fonte do caso de testes no diretório src\testcase. Para completar os artefatos da aplicação, o mecanismo de pós-processamento da Captor copia diversos artefatos contidos no diretório do domínio FWS da Captor para o diretório de saída do projeto gerado. Por exemplo, são copiados: o diretório dev\scripts, que contém um

script utilizado para compilar e executar as aplicações geradas; o diretório dev\testcase\lib, que contém o framework de testes utilizado para executar o caso de testes gerado (JUnit); o diretório doc\javadoc, que contém o manual de referência das classes FWS; o diretório simulator, que contém os simuladores ; e o diretório src\fwslib e src\fwslib\driver que contém o código-fonte das classes. O pós-processamento permite a automação de diversas facilidades após a geração de artefatos. Além da cópia dos arquivos, conforme apresentado na Seção 4.3.4.4, dependendo das características do projeto que está sendo gerado, o engenheiro de aplicação pode configurar a compilação automática do código-fonte, realizar o *commit* automático em mecanismos de controle de versão e outras tarefas executadas pela ferramenta Ant.

5.2.1.5 Documentação de domínio

A realização da atividade de documentação do domínio FWS deve fornecer um manual de uso para o ambiente de simulação e uma descrição de todas as classes e suas interfaces. Esses documentos são criados manualmente e armazenados no diretório do domínio FWS da Captor. Na primeira geração de um projeto no domínio FWS, a Captor copia o manual e a descrição das classes e suas interfaces no formato Javadoc (Java, 2006) junto com o código gerado da aplicação FWS para o diretório de saída do projeto. O manual do simulador do domínio FWS é apresentado no Apêndice A. A descrição das classes não é apresentada e nada acrescenta no entendimento do texto.

5.2.2 Engenharia de aplicação

Como apresentado na Seção 3.5, a realização da etapa de engenharia de aplicação tem o objetivo de construir aplicações com o uso da Captor configurada para um domínio específico. Nas Seções seguintes, as atividades da engenharia da aplicação do domínio FWS são detalhadas.

5.2.2.1 Engenharia de requisitos

Nesta etapa são coletados os requisitos de uma aplicação particular da FWS explicitando todas as variabilidades da bóia que deve ser construída e todas as funcionalidades que não forem cobertas pela Captor.

5.2.2.2 Criação do modelo da aplicação

Para realizar a criação do modelo da aplicação, o engenheiro de aplicação deve executar a Captor e criar um novo projeto no domínio FWS. Todas as variabilidades identificadas no documento de requisitos devem ser inseridas no modelo da aplicação por meio dos formulários e a Captor deve validar esses dados. Como exemplo, ilustra-se na Figura 5.5, a Captor

sendo utilizada para armazenar o modelo de uma aplicação FWS, na Figura 5.6 o formulário de especificação de bóia e na Figura 5.7 o formulário de especificação de sensores.

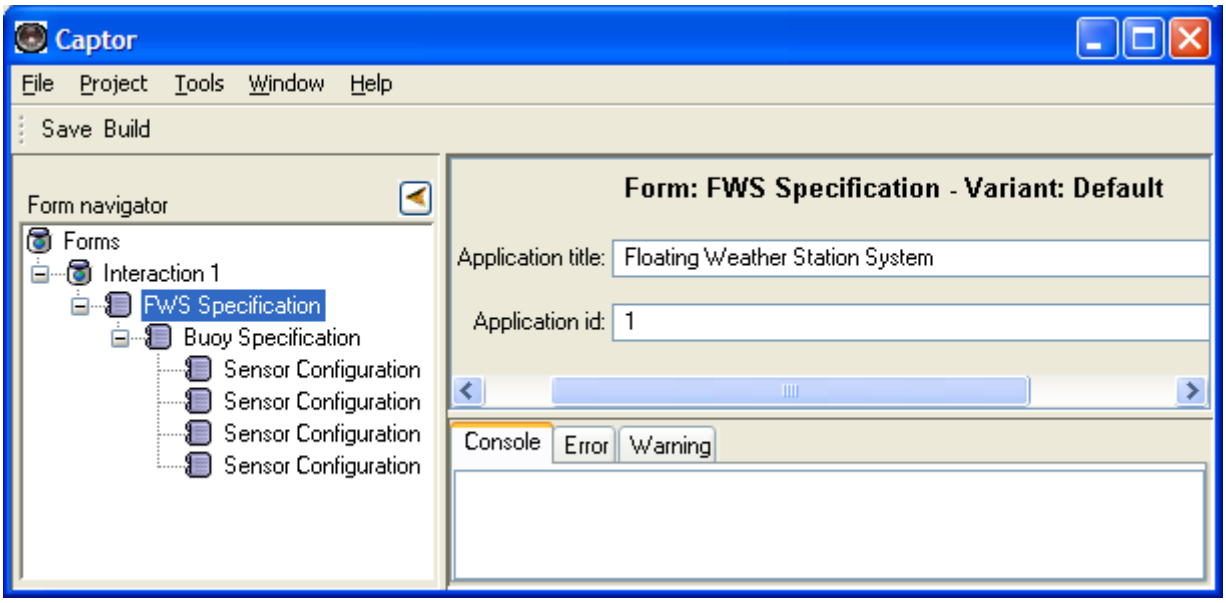

Figura 5.5: Captor sendo utilizada para armazenar o modelo de uma aplicação FWS

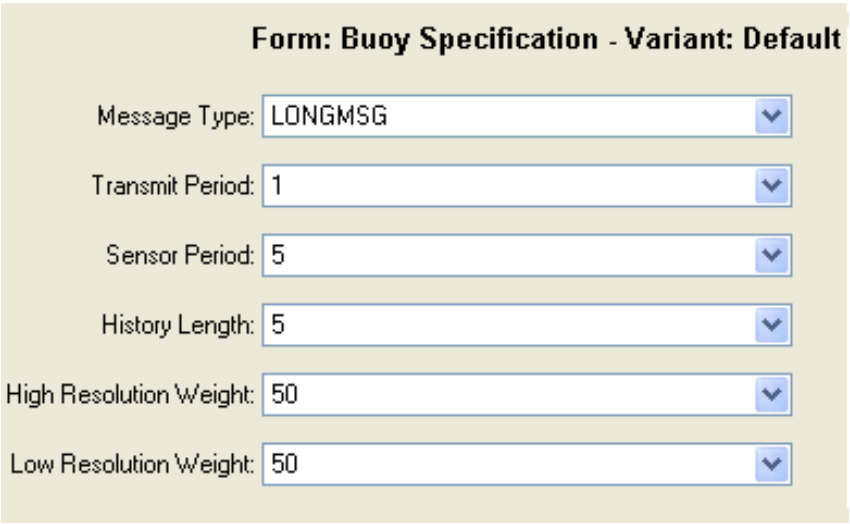

Figura 5.6: Formulário de especificação de bóias

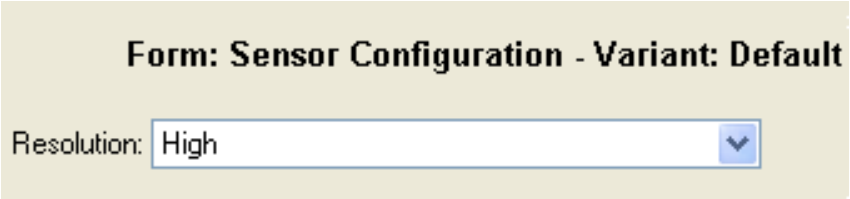

Figura 5.7: Formulário de especificação de sensores

5.2.2.3 Geração de artefatos

A geração de artefatos do domínio FWS deve produzir três arquivos de código-fonte (indicadas pelas classes cinza da Figura 5.3), um arquivo UML com a descrição das classes da aplicação e das classes da biblioteca de classes e um caso de testes do framework JUnit, que deve verificar se o sistema está gerando mensagens corretamente (LONGMSG ou SHORTMSG). Como ilustração apresenta-se, na Listagem 5.4, um trecho do código-fonte gerado da classe FWS.java.

Listagem 5.4: Código-fonte da classe gerada FWS.java

```
1  package fws;
2
3
4  import fwslib.DataBanker;
5  import fwslib.Runner;
6  import fwslib.FWSPrintStream;
7  import fwslib.MessageGenerator;
8  ...
9
10 public class FWS extends JFrame implements ActionListener,
11                                                WindowListener {
12
13  //Definições provenientes do formulário de configuração da bóia
14  public static final int SensorPeriod = 5 * 100;
15  public static final int TransmitPeriod = 1 * 100;
16  public static final int LowResWeight = 50;
17  public static final int HighResWeight = 50;
18  public static final String MsgType = "LONGMSG";
19  public static final int NumSensors = 4;
20  public static final int HistoryLenght = 5 * NumSensors;
21  public static final String ID = "1";
22
23  //Definições do ambiente de simulação
24  public static final int WIND_GENERATOR_PORT = 4017;
25  public static final int CONTROL_TOWER_PORT = 4018;
26  public static final String WIND_GENERATOR_HOST = "localhost";
27  public static final String CONTROL_TOWER_HOST = "localhost";
28  ...
29
30  //Método principal
31  public static void main(String[] args) {
32   FWS fws = new FWS();
33   fws.setVisible(true);
34  }
35
36  public void actionPerformed(ActionEvent e) {
37
38   if ( e.getActionCommand().equals("Start") ) {
39    startButton.setText("Stop");
40
41    runner.setRun(true);
42    DataBanker.init();
43    MessageGenerator m = new MessageGenerator(runner);
```

```
44      m.start();
45
46      SensorMonitor s = new SensorMonitor(runner);
47      s.start();
48     }
49     else if ( e.getActionCommand().equals("Stop") ) {
50      startButton.setText("Start");
51      runner.setRun(false);
52     }
53     else if ( e.getActionCommand().equals("Clean") ) {
54      msgTA.setText("");
55     }
56
57    }
58    ...
59
60    public void windowClosing(WindowEvent event) {
61     runner.setRun(false);
62
63     try {
64      Thread.sleep(1500);
65     } catch (InterruptedException e) {
66      System.out.println(e);
67     }
68
69     System.exit(0);
70    }
71
72    /*
73     * Inicio da zona de segurança utilizada para
74     * a criação de novos métodos.
75     */
76    // START-SAFE(newMethods)
77    // END-SAFE
78   }
```

5.2.2.4 Implementação de funcionalidades não cobertas pelo gerador

A implementação das funcionalidades não cobertas pelo gerador depende da aplicação que está sendo construída e pode ser realizada de acordo com a Seção 3.5.4. No exemplo apresentado, a Captor gerou todos os artefatos e não foi necessária a implementação de nenhuma funcionalidade adicional.

5.2.2.5 Testes e manutenção

Os testes das aplicações FWS podem ser realizados de forma automática, por meio do caso de teste gerado e por outros casos de testes criados manualmente, ou podem ser realizados com as técnicas de teste que o engenheiro de aplicação julgar mais conveniente. A manutenção do sistema pode ser realizada de acordo com o modelo apresentado na Seção 3.5.5. No caso do exemplo, foram realizados testes de maneira *ad-hoc*.

5.3 Gestão de recursos de negócios

Como foi apresentado na Seção 2.2.3.1, o domínio da linguagem de padrões GRN lida com aplicações nas quais seja necessário registrar transações de aluguel, comercialização ou manutenção dos bens ou serviços. Nesta Seção, a Captor é configurada para fornecer apoio na instanciação automática do framework GREN que, como apresentado na Seção 2.3.1, apoia a construção de aplicações com base na GRN.

5.3.1 Engenharia de domínio

Braga (2003) realizou a engenharia de domínio na qual foram realizadas as atividades de análise, projeto, implementação e documentação do domínio GRN. Na realização da engenharia de domínio deste estudo de caso, os artefatos criados por Braga (2003) foram reutilizados e os esforços de desenvolvimento foram concentrados na realização das atividades de criação da linguagem de modelagem de aplicações e configuração da Captor. Nas Seções seguintes são apresentadas resumidamente as atividades da engenharia de domínio realizadas por Braga (2003) no formato proposto neste trabalho, a sub-atividade da análise de domínio: "criação de uma linguagem de modelagem de aplicações" e atividade de configuração da Captor.

5.3.1.1 Análise de domínio

Descrição do domínio:

O domínio de gestão de recursos de negócio apresentado nesta Seção apoia aplicações nas quais seja necessário registrar transações de aluguel, comercialização ou manutenção dos bens ou serviços. O aluguel de recursos enfoca principalmente a utilização temporária de um bem ou serviço, como por exemplo uma fita de vídeo ou o tempo de um especialista. A comercialização de recursos enfoca a transferência de propriedade de um bem, como por exemplo uma venda ou leilão de produtos. A manutenção de recursos enfoca o reparo ou conservação de um determinado produto, utilizando mão-de-obra e peças para execução, como é o exemplo de uma oficina de reparo de eletrodomésticos.

Definição da terminologia comum:

Parte da terminologia comum do domínio de gestão de recursos de negócios é apresentada na Tabela 5.8.

Tabela 5.8: Terminologia comum do domínio de persistência

Termo:	**Significado**
Recurso	Bens ou possses.
Recurso de negócio	Bens ou possses que são negociados.
Negócio	Aluguel, venda ou manutenção.

Análise dos aspectos similares e variáveis:

A análise dos aspectos similares e variáveis do domínio GRN foram realizados por Braga (2003) e apresentados por meio da linguagem de padrões GRN. Essa linguagem foi brevemente apresentada na Seção 2.2.3.1 e por economia de espaço, ela não será apresentada nesta Seção. Para um detalhamento completo da linguagem, pode-se consultar Braga (2003) e Braga et al. (1999b)

Criação da linguagem de modelagem de aplicação:

A linguagem de padrões GRN contém quinze padrões e alguns deles podem conter até cinco variantes. Foram criados formulários, um para cada padrão/variante, e os dados variáveis de cada padrão devem ser especificados no seu formulário correspondente. A representação hierárquica da árvore de formulários da GRN tem a mesma forma da árvore de padrões apresentada na Figura 2.2. Para resumir a apresentação dos formulários, na Figura 5.8 é apresentado um trecho da árvore de formulários com os formulários 1, 2, 4 e 11, correspondentes aos padrões 1, 2, 4 e 11, respectivamente e, nas Tabelas 5.9, 5.10 e 5.11 são apresentados as representações Tabulares desses formulários[1].

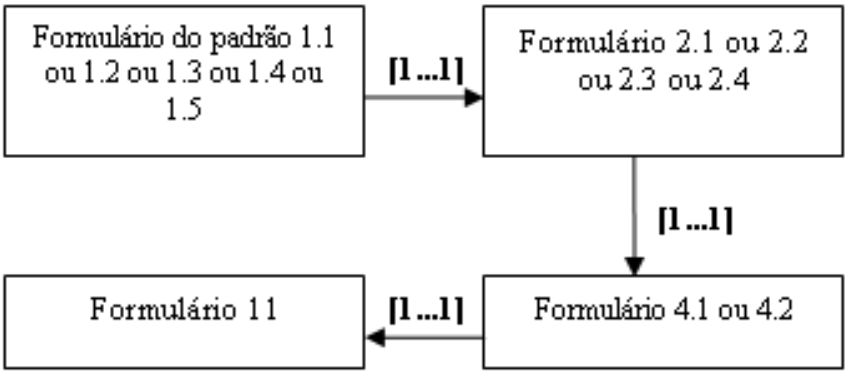

Figura 5.8: Árvore de formulários da LMA da GRN

[1]Os dados completos dos padrões/variantes da linguagem GRN podem ser encontrados no diretório do domínio GRN do pacote de instalação da Captor disponível no site `http://www.labes.icmc.usp.br/captor`.

Tabela 5.9: Representação tabular do formulário do padrão 1 variante 1

Elemento	Nome	Parâmetro	Valor
GRENClassPanel	Recurso	class	Recurso
		SYS_ATT1	IdCode:integer:0
		SYS_ATT2	description:char:10
GRENClassPanel	TipoDoRecurso	class	TipoDoRecurso
		SYS_ATT1	IdCode:integer:0
		SYS_ATT2	description:char:10

Tabela 5.10: Representação tabular do formulário do padrão 2 variante 2

Elemento	Nome	Parâmetro	Valor
GRENClassPanel	Recurso	class	Recurso
		read_from	parent(1.*)->Recurso
GRENClassPanel	InstanciaDoRecurso	class	InstanciaDoRecurso
		SYS_ATT1	resourceIdCode:integer:0
		SYS_ATT2	code:char:10
		SYS_ATT3	allocation:char:35
		SYS_ATT4	status:char:1

Em cada formulário do domínio, devem ser inseridos os dados variáveis de cada padrão, ou seja, o nome das classes que são utilizadas e os seus atributos adicionais. Por exemplo, no padrão 1 - variante 1 (apresentado na Seção 2.2.3.1), o engenheiro de aplicação deve inserir o nome da classe do recurso (por exemplo, Carro em uma loja de automóveis, Animal em uma clínica veterinária e Vídeo em uma videolocadora) e o tipo do recurso (por exemplo, Marca em uma loja de automóveis, Raça em uma clínica veterinária e Classificação em uma videolocadora). Cada classe já possui um conjunto de atributos padrão, especificados pelos parâmetros prefixados com SYS_ATTX (X é a variável que indica o número do atributo), e o engenheiro de aplicação deve adicionar no elemento de formulário `GRENClassPanel` os atributos adicionais de cada classe.

Embora seja possível especificar o domínio GRN com o conjunto de elementos de formulário padrão (caixas de texto, áreas de texto, listas de seleção e tabelas), o engenheiro de domínio GRN optou por desenvolver o elemento de formulário personalizado `GRENClassPanel` para aumentar a usabilidade da Captor no domínio GRN e disponibilizar um número de elementos de formulários maior para a configuração da Captor para outros domínios. Esse elemento possui dois campos, um para especificação do nome da classe e um para a especificação do nome da classe na forma plural (utilizado para a geração de uma etiqueta da GUI) e, uma janela associada em que é possível se especificar os atributos das classes que são específicos de aplicação.

Tabela 5.11: Representação tabular do formulário do padrão 4 variante 1

Elemento	**Nome**	**Parâmetro**	**Valor**
GRENClassPanel	Recurso	class	Recurso
		read_from	parent(1.*)->Recurso
GRENClassPanel	Aluguel	class	Aluguel
		SYS_ATT1	number:integer:0
		SYS_ATT2	date:date:0
		SYS_ATT3	observation:char:60
		SYS_ATT4	totalPrice:float:0
		SYS_ATT5	totalDiscount:float:0
		SYS_ATT6	destinationParty:integer:0
		SYS_ATT7	finishingDate:date:0
		SYS_ATT8	returnDate:date:0
		SYS_ATT9	fineValue:float:0
GRENClassPanel	Destino	class	Destino
		SYS_ATT1	id:integer:0
		SYS_ATT2	description:char:35
GRENClassPanel	TaxaDeMulta	class	TaxaDeMulta
		class_use	not_required
		attributes	false
		SYS_ATT1	percentageRate:float:0
		SYS_ATT2	lowNumber:integer:0
		SYS_ATT3	upperNumber:integer:0
GRENClassPanel	Origem	class	Origem
		class_use	not_required
		SYS_ATT1	id:integer:0
		SYS_ATT2	description:char:35

5.3.1.2 Projeto de domínio

Braga e Masiero (2001, 2002) criaram um processo para a construção de frameworks baseados em linguagens de padrões. O resultado da aplicação desse processo no domínio de gestão de recursos de negócio foi a criação da linguagem de padrões GRN e do seu framework caixa-branca associado GREN. A arquitetura e parte do projeto desse framework já foram parcialmente apresentados na Seção 2.3.1 e os seus detalhes podem ser encontrados em Braga e Masiero (2001), Braga e Masiero (2002) e Braga (2003).

Para instanciar manualmente esse framework, o engenheiro de aplicação deve criar classes de aplicação que estendem as classes abstratas do framework e sobre-escrevem os seus métodos-gancho. Para instanciar automaticamente esse framework, o gerador deve produzir essas classes de aplicação com base em uma instância da linguagem de modelagem de aplicações criada na atividade de análise de domínio.

5.3.1.3 Implementação de domínio

Nesta atividade, as classes abstratas definidas no projeto de domínio são implementadas utilizando uma linguagem de programação específica. O framework GREN foi implementado na linguagem SmallTalk (Goldberg, 1995) e o seu processo de implementação foi descrito em detalhes por Braga (2003).

5.3.1.4 Configuração da ferramenta

A realização da atividade de configuração da Captor no domínio GREN deve produzir três artefatos: o arquivo de configuração de interface e validação da especificação, os gabaritos e o arquivo MTL.

Arquivo de configuração de interface e validação da especificação

O engenheiro de domínio deve utilizar o modelo da árvore de formulários parcialmente apresentado na Figura 5.8 e as representações tabulares dos formulários, exemplificadas nas Tabelas 5.9, 5.10 e 5.11, que representam os elementos de formulários que devem ser utilizados para configurar a interface e as regras de validação, conforme o processo apresentado na Seção 4.3.1.4.

Gabaritos

O framework GREN foi implementado na linguagem SmallTalk com o ambiente de desenvolvimento Visual Works (Cincom, 2001). Nesse ambiente de desenvolvimento ou IDE (do inglês *Integrated Development Environment*), as classses são armazenadas em uma imagem binária impossibilitando o gerador de produzir o código das classes em arquivos-fonte, tais como os arquivos gerados nos estudos de caso de persistência e bóias náuticas. Para realizar a geração, a Captor produz um arquivo no formato XML com a definição das classes, atributos e métodos em uma estrutura padrão, esse arquivo é importado para a o Visual Works que insere essas classes na imagem binária na qual ele é capaz de executar. Dessa forma, a Captor precisa produzir dois artefatos: o arquivo de XML com a definição das classes e o *script* de criação de tabelas no banco de dados.

Esses gabaritos são desenvolvidos de forma similar à apresentada nas Seções 3.4.4.1, 4.3.4.2 e 5.2.1.4 e por economia de espaço não são apresentados nesta Seção. Foram construídos gabaritos para implementar as quatro variantes do padrão 1, as duas variantes do padrão 2, as duas variantes do padrão quatro e a única variante do padrão onze. Como a construção dos gabaritos é uma tarefa repetitiva e que não acrescenta valor de pesquisa para este trabalho, por motivo de

tempo, os outros gabaritos de instanciação do framework GREN não foram desenvolvidos.

Arquivo MTL

Conforme apresentado na Seção 4.3.4.3, o arquivo MTL deve selecionar os gabaritos que devem ser gerados com base no modelo da aplicação. Na Listagem 5.5 apresenta-se o arquivo MTL para o domínio GRN.

Listagem 5.5: Arquivo MTL para o domínio GRN

```
<composer name="Captor">

 <main>

  <callTask id="grn"/>
  <callTask id="sql"/>

 </main>

 <tasks>

  <task id="grn">

   <compose>
    <template>grn.st</template>
    <newFilename>grn.st</newFilename>
   </compose>

  </task>

  <task id="sql">

   <compose>
    <template>sql.sql</template>
    <newFilename>sql.sql</newFilename>
   </compose>

  </task>

 </tasks>

</composer>
```

5.3.1.5 Documentação de domínio

A documentação do domínio GRN deve fornecer um manual de uso do framework GREN, seus métodos-gancho e um manual de instanciação. Esses documentos foram criados por Braga (2003) e por motivos de espaço não são apresentados nesta Seção.

5.3.2 Engenharia de aplicação

Como apresentado na Seção 3.5, a realização da etapa de engenharia de aplicação tem o objetivo de construir aplicações com o uso da Captor configurada para um domínio específico. Nas Seções seguintes, as atividades da engenharia da aplicação do domínio GRN são resumidas, usando como exemplo uma aplicação de uma videolocadora.

5.3.2.1 Engenharia de requisitos

Nessa etapa os requisitos da videolocadora são mapeados para as funcionalidades disponibilizadas pela GRN e pelo framework GREN e as funcionalidades não cobertas pela Captor são elicitadas e documentadas. O resultado é que, para modelar a videolocadora, devem ser aplicados os padrões 1, 2, 4 e 11 da GRN.

5.3.2.2 Criação do modelo da aplicação

Para realizar a criação do modelo da aplicação, o engenheiro de aplicação deve executar a Captor e criar um novo projeto no domínio GRN. A utilização da linguagem de padrões nesse processo fornece as principais diretivas de como o sistema deve ser modelado e quais são as principais alternativas que o engenheiro tem na modelagem de aplicações no domínio GRN. Os dados coletados pelo engenheiro de aplicação em cada padrão devem ser inseridos nos formulários correspondentes do modelo da aplicação na Captor. Como exemplo, ilustra-se na Figura 5.9, a Captor sendo utilizada para armazenar o modelo da videolocadora, na Figura 5.10 o formulário do padrão 2-QUANTIFICAR O RECURSO e na Figura 5.11 o formulário do padrão 4-ALUGAR O RECURSO. Observe os botões `Attributes` nas Figuras 5.10 e 5.11, por meio dos quais é possível acrescentar novos atributos às classes, que são atributos específicos da aplicação.

5.3.2.3 Geração de artefatos

A geração de artefatos do domínio GRN deve produzir um arquivo XML com o código-fonte e um arquivo de script de criação de tabelas SQL de um banco de dados relacional. Como ilustração, apresenta-se para uma aplicação de VideoLocadoras a Listagem 5.6 com parte do script SQL de criação de tabelas e a Listagem 5.7 com parte do trecho do XML que representa o código de uma classe gerada. Note, a partir da linha 45 da Listagem 5.7, que foram acrescentados dois atributos novos à classe Video, gênero e restrição, que são tratados pelos métodos gerados.

Listagem 5.6: Script de criação de tabelas

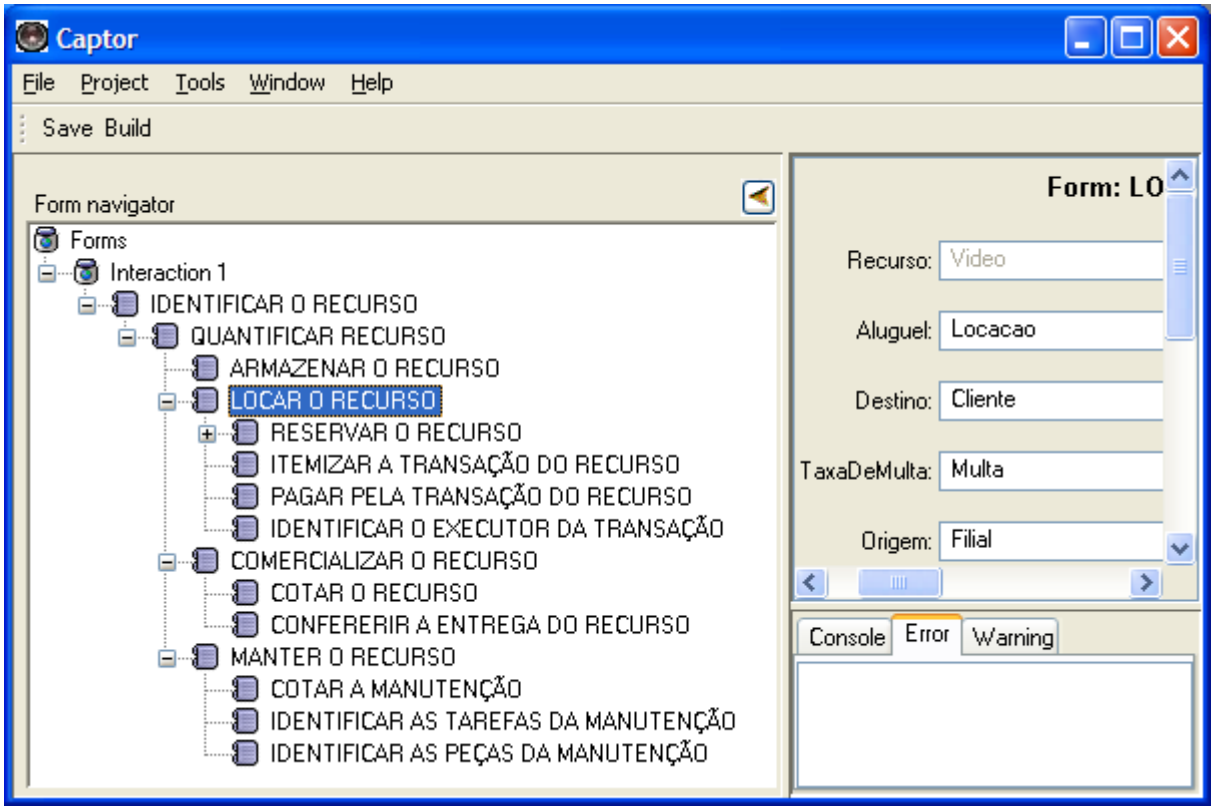

Figura 5.9: Captor sendo utilizada para armazenar o modelo de uma aplicação no domínio GRN

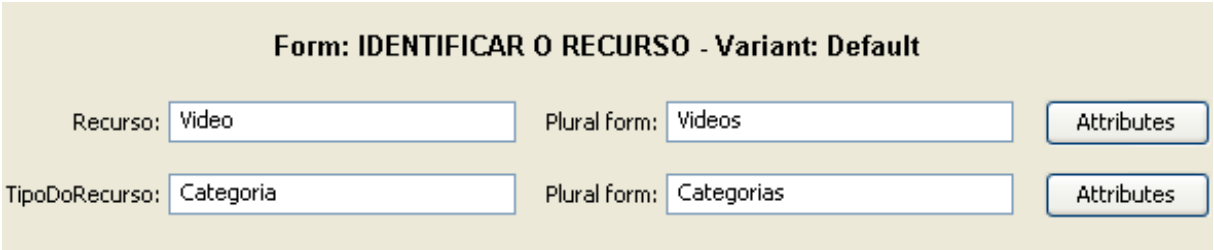

Figura 5.10: Formulário do padrão: Identificar o recurso

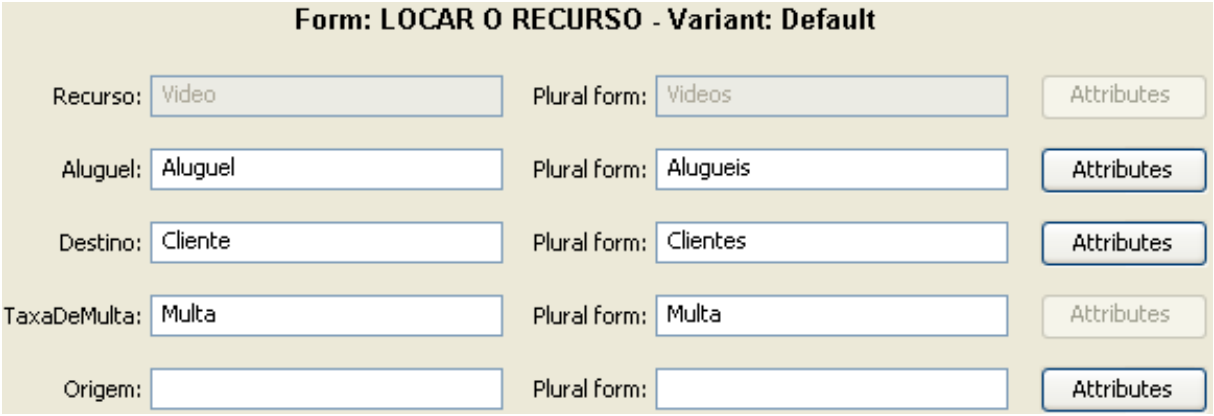

Figura 5.11: Formulário do padrão: Aluguel do recurso

```
#Apagar o banco de dados
DROP DATABASE IF EXISTS GRN;

#Criando o banco de dados
CREATE DATABASE IF NOT EXISTS GRN;

USE GRN;

#Tabela de idCodes
drop table IF EXISTS IdCode;
create table IF NOT EXISTS  IdCode (
          className char(35),
          lastIdCode integer);

#Tabela de Recurso
drop table IF EXISTS Video;
create table IF NOT EXISTS Video(
          idCode integer not null,
          description char(35),
          status char(1),
          genero char(100),
          restricao char(50));

#Tabela de aluguel de recurso
drop table IF EXISTS Locacao;
create table IF NOT EXISTS Locacao(
          number integer not null,
          date date,
          observation char(60),
          status char(1),
          totalPrice float,
          totalDiscount float,
          destinationParty integer,
          sourceParty integer,
          resource integer,
          finishingDate date,
          returnDate date,
          fineValue float);

...
```

Listagem 5.7: XML que representa a classe video

```
<class>

 <name>Video</name>
 <environment>GRN</environment>
 <super>GREN.Resource</super>
 <private>false</private>
 <indexed-type>none</indexed-type>

 <inst-vars>
  genero restricao
 </inst-vars>
 <class-inst-vars/>
 <imports/>
 <category>GRN</category>
```

```
15
16 </class>
17
18 <methods>
19  <class-id>GRN.Video class</class-id>
20  <category>classReferences</category>
21
22  <body>resourceRentalClass
23         ^GRN.Locacao</body>
24
25  <body>guiForm
26         ^GRN.VideoForm</body>
27
28  <body>quantificationStrategyClass
29         ^SingleResource</body>
30
31  <body>typeClasses
32         ^List new</body>
33
34  <body>typeForms
35         ^List new</body>
36
37 </methods>
38 ...
39
40 <methods>
41  <class-id>GRN.Video</class-id>
42  <category>SQLClauses</category>
43
44  <body>insertionFieldClause
45        ^super insertionFieldClause,',genero,restricao'.</body>
46
47  <body>insertionValueClause
48        ^super insertionValueClause , ',',
49        self genero printString, ',',
50        self restricao printString</body>
51
52  <body>updateSetClause
53        ^super updateSetClause,
54        ',genero=', self genero printString,
55        ',restricao=', self restricao printString.</body>
56
57 </methods>
58 ...
59
60 <methods>
61 <class-id>GRN1.Video</class-id>
62 <category>initialize-release</category>
63  <body>initialize
64   super initialize.
65   genero := ''.
66   restricao := ''.</body>
67
68 </methods>
69 ...
```

5.3.2.4 Implementação das funcionalidades não cobertas pelo gerador

A implementação das funcionalidades não cobertas pelo gerador dependem da aplicação que está sendo construída e pode ser realizada de acordo com a Seção 3.5.4. Para a realização dos estudos de caso deste trabalho, não foram implementadas funcionalidades adicionais.

5.3.2.5 Testes e manutenção

Os testes das aplicações no domínio GRN podem ser realizados de forma automática, por meio de casos de testes e um framework de testes específico, ou podem ser realizados com as técnicas de teste que o engenheiro de aplicação julgar mais conveniente. A manutenção do sistema pode ser realizada de acordo com o modelo apresentado na Seção 3.5.5. No caso da videolocadora, nenhuma técnica específica foi utilizada e os testes foram feitos de maneira *ad-hoc*.

5.4 Comparação de resultados

A realização dos três estudos de caso apresentados neste trabalho (persistência, bóias náuticas e GRN) permitiu medir o esforço necessário para se obter o apoio da Captor e comparar os processos de configuração e utilização nesses domínios. O esforço necessário para configurar a Captor para um domínio específico pode ser calculado de acordo com a fórmula abaixo, ou seja, o esforço de configuração total é igual à soma dos esforços da realização das atividades da etapa de engenharia de domínio (análise, projeto, implementação, configuração da Captor e documentação do domínio).

$$ECtotal = Eanálise + Eprojeto + Eimplementação + Econfiguração + Edocumentação$$

O principal objetivo deste trabalho foi reduzir o custo da atividade de configuração (*Econfiguração*) do gerador configurável Captor. Como ilustração dos resultados, pode-se citar que, com base nos artefatos da análise, projeto e implementação de domínio, a configuração da interface e regras de validação, criação do arquivo MTL e arquivo de pós-processamento para os domínios apresentados foram realizadas por um engenheiro de domínio especializado em menos de duas horas. Após a realização dessas atividades, para completar a configuração da Captor, deve-se construir os gabaritos. Essa atividade requer um esforço maior do que o esforço individual empregado na construção dos artefatos que são gerados a partir desses gabaritos manualmente. Por exemplo: no domínio de persistência, o gabarito da classe persistente, por causa do número limitado de variabilidades, pode ser construído em aproximadamente 1.5 vezes o tempo necessário para a construção manual de uma classe persistente; no domínio GRN,

o gabarito da classe de aplicação que estende a classe `Resource`, por causa do número de variabilidades para os quais o framework GREN oferece apoio, pode ser construído em aproximadamente 4 vezes o tempo necessário para construir manualmente a classe gerada por esse gabarito. Com base nos estudos de caso, conclui-se que, embora o esforço necessário para a construção de um gabarito seja maior do que a construção manual do artefato que esse gabarito produz, esse esforço pode ser recompensador porque existe tanto a possibilidade do engenheiro de aplicação utilizar esse gabarito diversas vezes na geração de uma mesma aplicação, como utilizar esse gabarito em diversas aplicações. Como ilustração dessas situações, no domínio de persistência, o gabarito da classe persistente pode, normalmente, ser utilizado para gerar dezenas de classes persistentes em uma mesma aplicação e, além disso, a configuração da persitência (e os seus gabaritos) pode ser utilizada na construção de diversas aplicações.

O esforço necessário para configurar a Captor para um domínio específico é representado pela fórmula abaixo. Com base nos estudos de caso, pode-se notar que, com a utilização dos artefatos criados nas atividades de análise e projeto de domínio, o esforço necessário para criar os arquivos de configuração da Captor é baixo e, conforme foi apresentado, a maior parte do esforço se concentra na construção dos gabaritos.

Econfiguração = Ecriação_arquivos_configuração + Egabaritos

É importante notar que, por terem sido reutilizados estudos de caso de outros autores, não foi possível calcular o esforço empregado nas atividades de análise, projeto, implementação e documentação de domínio, por falta de dados sobre o esforço empregado nos exemplos apresentados. De qualquer forma, com base nos resultados obtidos, pode-se concluir que para utilizar a Captor de forma que os esforços de desenvolvimento sejam menores que os esforços empregados nos métodos sem auxílio do gerador configurável, o retorno do investimento do esforço realizado na engenharia de domínio deve ser compensado durante a etapa de engenharia de aplicação. Com base nos estudos de caso, não foi possível chegar a uma fórmula genérica para o cálculo do número de aplicações que devem ser geradas para que o esforço da engenharia de domínio seja recompensado. Pelos dados coletados, o cálculo desse número depende, principalmente, dos seguintes fatores: dificuldade encontrada na definição das variabilidades e similaridades do domínio, da quantidade de variabilidades, da implementação dos artefatos reutilizáveis, do número de gabaritos do código da aplicação, da quantidade de artefatos que cada gabarito produz por aplicação e do apoio que o gerador fornece no domínio (geração de apenas parte dos artefatos, geração de grande parte dos artefatos, etc). A avaliação do custo/benefício da utilização dos geradores configuráveis deve ser analisada caso a caso e, mesmo assim, como acontece com outras técnicas de estimativa, a boa aproximação depende da experiência do engenheiro de domínio com o processo proposto e com o domínio avaliado.

Além da avaliação da economia do esforço que pode ser obtida com a utilização da Captor, foi realizada a geração de aplicações para sistemas desenvolvidos de duas maneiras diferentes: sistemas orientados a objetos e sistemas baseados em frameworks caixa-branca. Conforme apresentado na Figura 5.12, sistemas orientados a objetos tem seu código dividido basicamente em duas partes: classes fixas e classes variáveis. O código da aplicação parametriza e reutiliza as classes fixas para realizar funcionalidades específicas de aplicação. Conforme apresentado na Figura 5.13, nos sistemas baseados em frameworks caixa-branca, o código da aplicação também se divide em duas partes: uma estrutura fixa formada pelo framework e uma estrutura variável formada pelo código da aplicação. O código da aplicação é composto por classes concretas que estendem as classes abstratas do framework, preenchem os seus métodos-gancho e, opcionalmente, adicionam funcionalidades no sistema.

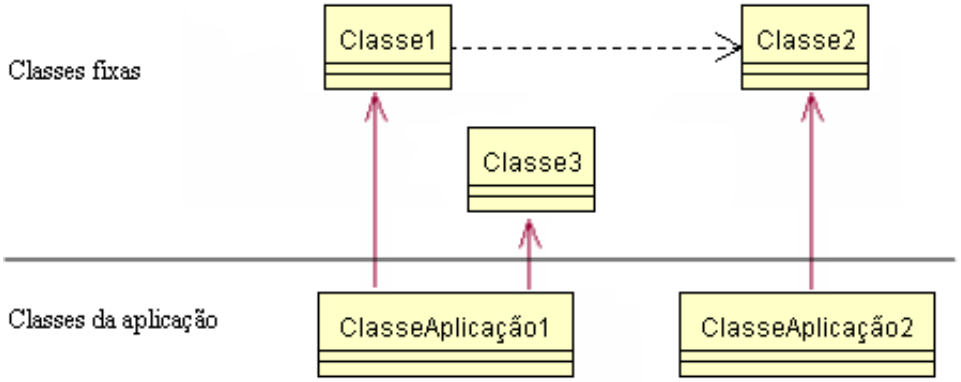

Figura 5.12: Aplicação orientada a objetos

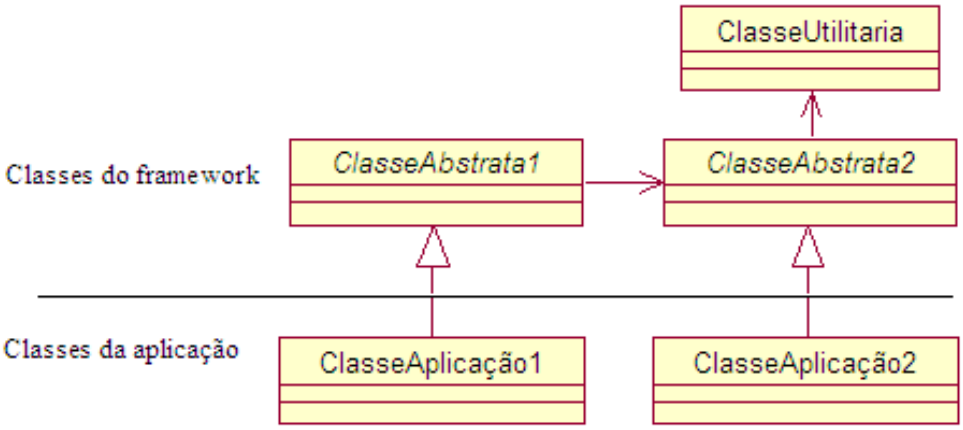

Figura 5.13: Instanciação de frameworks caixa-branca

Como a geração permite que sejam produzidos automaticamente grandes volumes de código, se não forem utilizadas técnicas adequadas para projetar os sistemas que devem ser gerados, pode-se obter grandes porções de código de baixa qualidade, ou seja, alto acoplamento, baixa coesão e baixa manutenibilidade. Em casos críticos, esse volume de código de baixa qualidade pode ser mais custoso para desenvolver e manter do que a criação de um código melhor

projetado, desenvolvido e mantido de forma manual. Conforme observado nos estudos de caso realizados neste trabalho, a geração do código da aplicação para sistemas orientados a objetos e frameworks caixa-branca pode apresentar boa qualidade, ou seja, baixo acoplamento, alta coesão e boa manutenibilidade.

Finalmente, no estudo de caso do domínio de bóias náuticas, foram gerados outros artefatos além de código-fonte: um caso de testes e um diagrama UML. Embora neste trabalho a geração de outros artefatos com base na especificação não tenha sido explorada profundamente, percebeu-se que a geração desses artefatos também pode maximizar os benefícios dos geradores de aplicação configuráveis e do processo proposto.

5.5 Considerações finais

Neste Capítulo foram apresentados dois dos três estudos de caso realizados com a Captor. Na Seção 5.2 foi apresentada a configuração e utilização da Captor no domínio de bóias náuticas e na Seção 5.3 foi apresentada a configuração e utilização da Captor no domínio GRN. A realização destes estudos de caso permitiu testar a Captor em domínios diferentes e estabelecer os principais critérios que devem ser analisados para estimar o custo da sua configuração para um domínio específico. Além disso, foram implementadas a geração de código e outros artefatos para sistemas orientados a objetos e sistemas baseados em frameworks caixa-branca.

CAPÍTULO

6

Conclusões

6.1 Considerações finais

Este trabalho apresentou os geradores de aplicação configuráveis de maneira genérica, delineou um processo de desenvolvimento apoiado por geradores configuráveis, o gerador de aplicação configurável Captor e três estudos de caso: persistência de objetos, gestão de recursos de negócios e bóias náuticas.

Com este trabalho o grupo de engenharia de software do ICMC tem agora ao seu dispor uma ferramenta que facilitará bastante a exploração de diferentes domínios e poderá apoiar a realização de experimentos e prototipação com diferentes aplicações. Os artefatos de software que podem ser gerados são os mais variados possíveis, como por exemplo código, documentação e testes. Em termos de arquiteturas de software, poderão ser definidos domínios em que a arquitetura é composta por um conjunto monolítico e integrado de classes, frameworks caixa branca que são instanciados por meio de especializações de classes, componentes caixa branca e código de ligação para componentes caixa-preta.

6.2 Contribuições

A principal contribuição deste trabalho foi a ferramenta Captor, que é um gerador de aplicação configurável desenvolvido em Java. A Captor baseia-se em arquivos XML e gabaritos XSL para realizar a configuração para domínios específicos. As informações sobre o domínio são

incluídas na Captor pelo engenheiro de domínio, por meio de uma linguagem de modelagem de aplicações (LMA), que é uma linguagem de alto nível capaz de representar as variabilidades do domínio. Uma instância de uma LMA é fornecida pelo engenheiro de aplicação por meio de formulários que ele preenche na interface gráfica da Captor.

Adicionalmente, foi delineado um processo de desenvolvimento baseado em geradores de aplicação configuráveis. Esse processo é composto por três etapas: desenvolvimento do gerador configurável, engenharia de domínio e engenharia de aplicação. Na etapa de desenvolvimento, o gerador configurável é analisado, projetado, implementado e testado. Na etapa de engenharia de domínio, o gerador configurável é instanciado para um domínio escolhido e, finalmente, na fase de engenharia de aplicação, o gerador é utilizado para gerar artefatos específicos do domínio para o qual foi configurado.

Também como parte deste trabalho, foi desenvolvida uma linguagem para seleção de gabaritos (MTL), que permite o mapeamento dos gabaritos para os arquivos XML contendo a especificação das aplicações, e uma ferramenta para validação das especificações (MMV) em XML.

6.3 Limitações do trabalho efetuado

Embora a realização dos estudos de caso deste trabalho tenha permitido que o processo proposto e a Captor fossem avaliados em domínios diferentes, para realizar uma avaliação mais precisa dos benefícios que podem ser alcançados com os geradores configuráveis, é necessária a realização de estudos de caso maiores, idealmente realizados em ambientes de desenvolvimento profissionais. Com esses dados seria possível fazer uma avaliação mais precisa do quociente da divisão entre o custo e o benefício da adoção dos geradores de aplicação configuráveis em domínios específicos.

6.4 Sugestões para trabalhos futuros

Além da realização de estudos de casos para situações mais realísticas de desenvolvimento de software, um trabalho a ser desenvolvido futuramente é a extensão da Captor para permitir que um projeto possa envolver mais de um domínio simultaneamente. Dessa forma, seria permitida a composição de aplicações por meio da fusão de diversos domínios, provavelmente em níveis diferentes. Por exemplo, um domínio de negócios, como a Gestão de Recursos de Negócios, poderia ser implementado com sua própria LMA, enquanto domínios transversais tais como, por exemplo, os domínios de persistência ou de controle de acesso, poderiam ser implementados com outras LMAs. Durante a etapa de utilização da Captor, o engenheiro de

aplicações entraria com uma instância de cada uma das LMAs, e a Captor faria a composição da aplicação final. Técnicas de orientação a aspectos (Clarke e Baniassad, 2005) poderiam ser usadas na composição.

Outro possível trabalho seria a integração automática de testes das aplicações geradas por meio da Captor. As atividades de geração de casos de testes poderiam ser integradas, de forma que ao gerar uma aplicação fossem gerados automaticamente os casos de testes associados, seguindo algum critério de testes previamente especificado.

A Captor também poderia ser estendida para facilitar o gerenciamento de configuração das aplicações geradas. Isso poderia ser feito por meio de um módulo adicional que armazenasse cada uma das versões dos arquivos utilizados na especificação e dos arquivos gerados.

Referências

ALEXANDER, C. *The timeless way of building*. Oxford University Press, 1979.

ALLEN, B. D. *Designing for Change: Minimizing the Impact of Changing Requirements in the Later Stages of a Spaceflight Software Project*. Relatório Técnico, NASA Scientific and Technical Information Program Office, 1998.

ALMEIDA, E. S.; LUCRÉDIO, D.; DE PAULA BIANCHINI, C.; DO PRADO, A. F.; TREVELIN, L. C. Mvcase tool - a technology integrating tool for the distributed component-based development. *XIV Simpósio Brasileiro de Engenharia de Software SBES2002*, p. 432–437, 2002.

APPLETON, B. Patterns and software: Essential concepts and terminology. *Object Magazine Online*, 2000.

BATORY, D.; CHEN, G.; ROBERTSON, E.; WANG, T. Web-advertised generators and design wizards. *5th International Conference on Software Reuse, Victoria, Canada*, 1998.

BATORY, D.; GERACI, B. Validating component compositions in software system generators. *International Conference on Software Reuse (ICSR) (Orlando)*, 1996.

BELL, T. E.; THAYER, T. A. Software requirements: Are they really a problem? *Proceedings of the 2nd international conference on Software engineering*, 1976.

BERRY, D. M. The safety requirements engineering dilemma. *Proceedings of the 9th International Workshop on Software Specification and Design*, 1998.

BICHLER, L. A flexible code generator for mof-based modeling languages. *Position Paper at the 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.

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

BOHLEN, M.; BRANDON, C.; ZOONS, W. Andromda. 2006. Disponível em `http://www.mysql.com` (Acessado em 16/02/2006)

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The unified software development process*. Addison-Wesley, 1999.

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

BRAGA, A.; RUBIRA, C.; DAHAB, R. *Tropyc: A pattern language for cryptgraphic - pattern languages of program design 4*, cáp. 16 Addison-Wesley, p. 337–371, 1999a.

BRAGA, R. T. V. *Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico*. Tese de Doutoramento, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2003.

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

BRAGA, R. T. V.; MASIERO, P. C. Identification of framework hot spots using pattern languages. In: *In 15th Brazilian Symposium on Software Engineering*, p. 145—160, 2001.

BRAGA, R. T. V.; MASIERO, P. C. The role of pattern languages in the instantiation of object-oriented frameworks. In: *8th International Conference on Object-Oriented Information Systems (OOIS 02) Montpellier, França, Lecture Notes On Computer Science*, p. 122–131, 2002.

BRAGA, R. T. V.; MASIERO, P. C. Building a wizard for framework instantiation based on a pattern language. *9th International Conference on Object-Oriented Information Systems (OOIS 03), Genebra, Suiça*, 2003.

BROWN, K.; WHITENACK, B. G. *Crossing chasms: A pattern language for object-rdbms - pattern languages of program design 2*, cáp. 14 Addison-Wesley, p. 227–238, 1995.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMNERLAD, P.; STAL, M. *Pattern-oriented software architecture, system of patterns*. British Library Cataloguing in Publication Data, 2000.

CINCOM Visualworks 5i.4 non-commercial. Disponível para instalação em 25/09/2001 na URL: `http://www.cincom.com`, 2001.

CLARKE, S.; BANIASSAD, E. *Aspect-oriented analysis and design*. Addison-Wesley, 2005.

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

COAD, P.; NORTH, D.; MAYFIELD, M. *Object models: Strategies, patterns and applications*. 3 ed. Yourdon Press, 1997.

CVS Concurrent version control. 2006.
Disponível em `http://www.cvshome.org/` (Acessado em 15/03/2006)

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

DANIELS, J.; CHEESMAN, J. *Uml components*. Addison-Wesley, 2000.

DÍAS-HERRERA, J. Issues and models in software product lines. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, v. 10, n. 4, p. 527–539, 2000.

DAVIDSON, J. D.; ATHERTON, B.; BAILLIEZ, S.; BENSON, M. The Apache Ant Project. Available at: http://ant.apache.org. 2005.
Disponível em `http://ant.apache.org/contributors.html` (Acessado em 26/09/2005)

DEURSEN, A.; KLINT, P. Little languages: little maintenance? *Journal of Software Maintenance*, p. 75–92, 1997.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, v. 35, n. 6, p. 26–36, 2000.

D'SOUZA, D. F.; WILLS, A. C. *Objects, components, and frameworks with UML - the catalysis approach*. Addison-Wesley, 339–381 p., 1999.

FAYAD, M.; SCHIMIDT, D. C.; JOHNSON, R. E. *Building application frameworks - application frameworks*. Wiley Computer Publishing, 3–27 p., 1999.

FRAKES, W.; ISODA, S. Success factors of systematic reuse. *IEEE Software*, v. 11, n. 9, p. 14–19, 1994.

FRAKES, W.; TERRY, C. *Software reuse and reusability metrics and models*. Relatório Técnico TR-95-07, 1995.
Disponível em `citeseer.ist.psu.edu/article/frakes96software.html`

GAMMA, E.; BECK, K. The JUnit framework. Available at: http://www.junit.org. 2005.
Disponível em `http://www.junit.org` (Acessado em 26/09/2005)

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

GOLDBERG, A. *The smalltalk developers guide to visualworks*. SIGS Books, 1995.

GOMAA, H. *Designing software product lines with uml*. Addison-Wesley, 2005.

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

HARRISON, N.; FOOTE, B.; ROHNERT, H. *Pattern languages of program design 4*. Addison-Wesley, 1999.

HARSU, M. *A survey on domain engineering*. Relatório Técnico, Institute of Software Systems, Tampere University of Technology, 2002.

HORSTMANN, C. *Core java 2, advanced features*. Prentice Hall, 2001a.

HORSTMANN, C. *Core java 2, fundamentals*. Prentice Hall, 2001b.

J. E. GAFFNEY, J.; CRUICKSHANK, R. D. A general economics model of software reuse. In: *ICSE '92: Proceedings of the 14th international conference on Software engineering*, New York, NY, USA: ACM Press, p. 327–337, 1992.

JACK HERRINGTON Extensible code generation with java, part 2. 2005. Disponível em `http://today.java.net/pub/a/today/2004/05/31/generation-pt2.html` (Acessado em 19/10/2005)

JAKARTA Apache software foundation. 2006. Disponível em `http://jakarta.apache.org` (Acessado em 15/03/2006)

JAVA Java 2 standard edition. 2006. Disponível em `http://java.sun.com` (Acessado em 15/03/2006)

JOHNSON, R.; FOOTE, B. Desiging reusable classes. *Journal of Object Oriented Programming*, p. 22–35, 1988.

JOHNSON, R.; WOOLF, B. *Pattern languages of program design 3*. Addison-Wesley, 47–65 p., 1998.

JOHNSON, R. E. Documenting frameworks using patterns. *OOPSLA*, 1992.

LARMAN, C. *Applying uml and patterns*. Prentice Hall, 2004.

LIM, W. Effects of reuse on quality productivity and economics. *Guest Editors Introduction, IEEE Expert*, p. 23–30, 1992.

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

MATTSSON, M.; BOSCH, J. Framework composition: Problems, causes and solutions. *Proceedings of TOOLS USA*, 1997.

MILLER, J.; MUKERJI, J. *Model driven architecture (MDA)*. Relatório Técnico, OMG - Object Management Group, 2001.

MILLER, J.; MUKERJI, J. *MDA guide version 1.0.1*. Relatório Técnico, OMG, 2003.

MONDAY, P.; CAREY, J.; DANGLER, M. *San francisco component framework: An introduction*. Addison-Wesley, 2000.

MYSQL MySQL 3.23 version. Disponível para instalação em 25 de Setembro de 2001 na URL: `http://www.mysql.com`, 2001.

OMG MOF 2.0 / XMI Mapping Specification, v2.1. Available at: http://www.omg.org. 2005. Disponível em `http://www.omg.org` (Acessado em 26/09/2005)

OMG Object management group. 2006.
Disponível em `http://www.omg.org` (Acessado em 07/03/2006)

PAZIN, A.; PENTEADO, R. A. D. GAwCRe: Um gerador de aplicações baseadas na web para o domínio de gestão de clínicas de reabilitação. 2004.

PREE, W. *Design patterns for object-oriented software development*. Addison-Wesley, 1995.

PRESSMAN, R. S. *Software engineering - a practitioner's approach*, v. 5. McGraw-Hill, 2001.

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

RAUSCH, A. A proposal for a code generator based on xml and code templates. *Proceedings of the Workshop of Generative Techniques for Product Lines, 23rd International Conference on Software Engineering*, 2001.

SAWYER, P. Maturing Requirements Engineering Process Maturity . *Requirements Engineering for Sociotechnical Systems, Idea Group Inc. (invited contribution)*, 2004.

SIEGEL, J. *Developing in OMG's model-driven architecture*. Relatório Técnico, OMG - Object Management Group, 2001.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. *Sistemas de banco de dados*. 3 ed. Makron Books, 1999.

SMARAGDAKIS, Y.; BATORY, D. Application generators. *Encyclopedia of Electrical and Electronics Engineering*, j.G. Webster (ed.), John Wiley and Sons, 2000.

SOMMERVILLE, I. *Engenharia de software*. Addison Wesley, 102-123 p., 2003.

STEYAERT, P.; CODENIE, W.; VERACHTERT, W. Effort estimation for changing requirements. *OOPSLA'96 Workshop on OO Process and Metrics for Effort Estimation*, 1996.

TAMAI, T.; ITOU, A. Requirements and design change in large-scale software development: analysis from the viewpoint of process backtracking. *Proceedings of the 15th international conference on Software Engineering*, 1993.

TANENBAUM, A. *Computer networks*. 4 ed. PRENTICE HALL, 2002.

TOLVANEN, J.-P.; GRAY, J.; ROSSI, M. 2nd workshop on domain-specific visual languages. In: *OOPSLA '02: Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ACM Press, p. 94–95, 2002.

VALERIO, A.; SUCCI, G.; FENAROLI, M. Domain analysis and framework-based software development. *SIGAPP Appl. Comput. Rev.*, v. 5, n. 2, p. 4–15, 1997.

W3C W3c. 2006.
Disponível em `http://www.w3c.org` (Acessado em 15/03/2006)

WEISS, D. M.; LAI, C. T. R. *Software product-line engineering*. Addison-Wesley, 1999.

XSL The extensible stylesheet language family (xsl). 2005.
Disponível em `http://www.w3.org/Style/XSL/` (Acessado em 09/03/2005)

YASSIN, A.; FAYAD, M. *Domain-specific application frameworks - application frameworks: A survey*, cáp. 29 Wiley Computer Inc, p. 615–632, 2000.

YODER, J. W.; JOHNSON, R. E.; WILSON, Q. D. Connecting business objects to relational databases. In: *5th Conference on the Pattern Languages of Programs*, Monticello-IL, EUA, disponível em 26/09/02 na WWW na URL: `http://www.joeyoder.com/papers/`, 1998.

APÊNDICE

A

Manual do simulador das bóias FWS

Existem dois simuladores do ambiente no qual as bóias são utilizadas. Esses simuladores são utilizados durante o processo de engenharia de aplicação para facilitar o desenvolvimento e testes do software das bóias FWS. O sistema completo (aplicação + simuladores) foi desenvolvido para ser executado de forma distribuída em uma arquitetura cliente/servidor. Um simulador representa uma torre de controle, outro simulador representa o vento e a aplicação FWS interage com esses dois simuladores durante a sua execução.

Na bóia, cada sensor é implementado em uma linha de execução separada que coleta informações do vento de forma independente e armazena o valor lido no histórico de leituras. Na virada do ciclo do período de transmissão de mensagens, a bóia deve enviar a média dessas medições para a torre de controle.

O ambiente de simulação deve ser desenvolvido de forma distribuída, ou seja, os dois simuladores e a bóia devem ser executados em espaço de memória separados e a comunicação deve ser realizada por meio da troca de mensagens TCP/IP. A porta de escuta do gerador de vento é 4017 e da torre de controle é 4018. Para iniciar a simulação, a aplicação deve abrir conexões TCP nessas portas com endereço IP das máquinas onde os simuladores estão sendo executados.

A execução da aplicação FWS com os simuladores é ilustrada na Figura A.1.

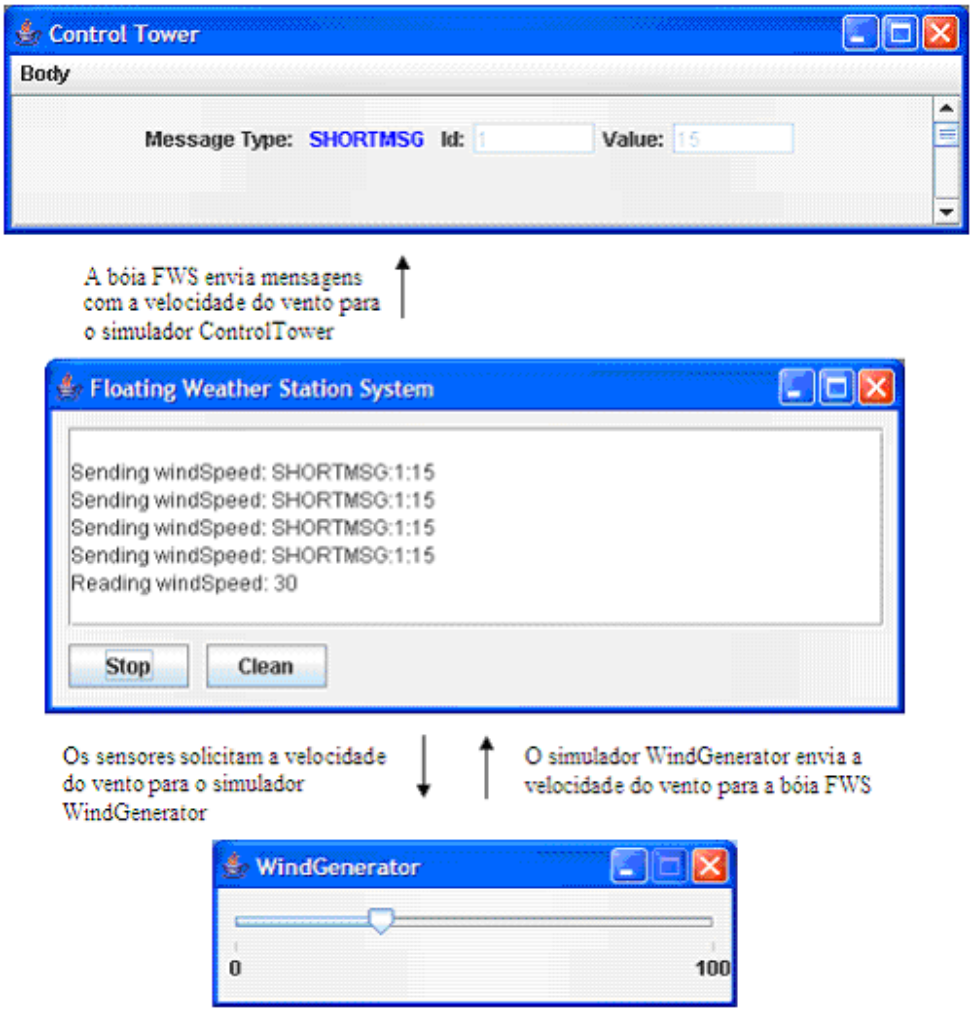

Figura A.1: Execução da aplicação FWS e dos simuladores.