Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico

Rosana Teresinha Vaccare Braga

Tese de Doutorado apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC/USP, para a obtenção do título de Doutor na Área de Ciências de Computação e Matemática Computacional.

Orientador: Prof. Dr. Paulo Cesar Masiero

Dedico este trabalho
Ao Cirilo, ao Thales e ao Vitor

Agradecimentos

Agradeço a Deus, pela vida!

Ao Cirilo por estar sempre presente, mesmo nas horas mais difíceis, pelo incentivo constante para que eu atingisse meus objetivos e por muitas vezes suportar minha ausência.

Aos meus filhos Thales e Vitor, por compreenderem a importância deste doutorado para mim e me apoiarem, mesmo sabendo que para consegui-lo eu teria que diminuir minha dedicação em parte de suas vidas.

Aos meus familiares, em especial aos meus pais Miguel e Maria Teresa, por sempre me incentivarem a continuar os estudos e por me ampararem sempre que deles preciso.

Ao meu orientador, Prof. Dr. Paulo Cesar Masiero, pela exímia condução da pesquisa, pela confiança em me aceitar como aluna e pelos ensinamentos passados ao longo deste trabalho.

Ao Prof. Dr. Fernão Stella R. Germano, pelo incentivo que fez com que eu retornasse à Universidade após dez anos de formada e pelas muitas contribuições oferecidas a este trabalho.

À Prof.ª Dr.ª Rosângela Penteado pela amizade, pela participação ativa durante as discussões e pela disponibilidade na execução de experimentos conjuntos.

Ao Prof. Dr. Francisco Louzada-Neto, do departamento de Estatística Universidade Federal de São Carlos e à Prof.ª Gleici Castro Perdoná, da Faculdade de Medicina de Ribeirão Preto, Universidade de São Paulo, pela apuração dos resultados estatísticos dos experimentos.

À aluna de doutorado Maria Istela Cagnin, que realizou usos da GRN, do GREN e do GREN-Wizard em diversos estudos de caso, pelas inúmeras sugestões de melhorias e pela dedicação na implementação de novas funcionalidades ao GREN e ao GREN-Wizard.

Ao aluno de mestrado Reginaldo Ré, que utilizou a abordagem proposta num outro domínio (leilões virtuais), pelas melhorias propostas ao processo.

Aos amigos do Labes: Adenilso, Aline, André, Andrea, Auri, Débora, Elisa, Ellen, Erika, Gelza, Prof. José Carlos, Lisandra, Luciana, Maria Istela, Mateus, Mayb, Reginaldo, Profa. Renata, Rejane, Profa. Rosely, Sandra, Simone, Tania, Tatiana, Thaise, Valéria e Willie, pelos bons momentos de convivência e descontração. Em especial ao Adenilso, Reginaldo e André pelo apoio no uso do LaTeX.

Aos participantes dos experimentos: alunos do Bacharelado em Ciências da Computação do ICMC-USP (cursando o oitavo período no segundo semestre de 2001); alunos de pós-graduação da UFSCAR da disciplina "Tópicos em Engenharia de Software" (segundo semestre de 2001); alunos do curso de especialização em "Tecnologia da Informação" do SENAC/UNESP-Presidente Prudente (2001); e alunos de pós-graduação do ICMC da disciplina "Seminários Avançados em Engenharia de Software I" (segundo semestre de 2002), pela dedicação e seriedade com a qual conduziram os experimentos.

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

À FAPESP, pelo apoio financeiro.

Sumário

1 Introdução **1**
1.1 Contextualização 1
1.2 Motivação 2
1.3 Objetivos 4
1.4 Organização 5

2 Situação do Processo Proposto no Panorama da Literatura Técnica Especializada **7**
2.1 Considerações Iniciais 7
2.2 Conceitos básicos relacionados à tese 8
2.2.1 Padrões e Linguagens de Padrões 8
2.2.2 Frameworks de Software Orientados a Objetos 12
2.2.3 Outras formas de reuso 16
2.2.4 Comparações entre formas de reuso 18
2.3 Padrões de Análise e Linguagens de Padrões Relevantes 20
2.4 Frameworks Relevantes na área de Negócios 23
2.4.1 IBM SanFrancisco 23
2.4.2 OmniBuilder 24
2.4.3 Accounts 25
2.5 Trabalhos relevantes que relacionam padrões a frameworks 26
2.5.1 *HotDraw* 26
2.5.2 G++ 27
2.6 Abordagens para Construção de frameworks 29
2.7 Abordagens para Instanciação de Frameworks 31
2.8 Considerações Finais 32

3 Um processo Geral para Desenvolvimento e Instanciação de Frameworks a partir de uma Linguagem de Padrões **34**
3.1 Considerações Iniciais 34
3.2 Visão Geral do Processo 35
3.3 O Processo de Construção da Linguagem de Padrões 38
3.3.1 Análise de Domínio ou Engenharia Reversa 39
3.3.2 Determinação dos Padrões 39

3.3.3 Criação de um grafo de fluxo de aplicação dos padrões 41
3.3.4 Escrita dos Padrões 42
3.3.5 Validação da Linguagem de Padrões 43
3.4 Processo de Uso de uma Linguagem de Padrões 43
3.5 A Linguagem de Padrões para Gestão de Recursos de Negócios 45
3.5.1 Visão Geral 45
3.5.2 Exemplo de um Padrão da GRN 48
3.5.3 O processo de construção da GRN 51
3.5.4 O Processo de Uso da GRN 52
3.5.5 Processo detalhado para Aplicação da GRN 53
3.5.6 Exemplo de Aplicação da GRN 55
3.6 Considerações Finais 58

4 O Processo de Construção e Instanciação de um Framework Caixa Branca 60
4.1 Considerações Iniciais 60
4.2 O Processo de Construção 61
4.2.1 Identificação dos Pontos Variáveis 61
4.2.2 Projeto do framework 64
4.2.3 Implementação do framework 65
4.2.4 Validação do framework 67
4.3 Exemplo – GREN: Um Framework para Gestão de Recursos de Negócios 68
4.4 O Processo de Instanciação 74
4.4.1 Análise do Sistema 74
4.4.2 Mapeamento entre o modelo de análise e o framework 75
4.4.3 Implementação do sistema específico 76
4.4.4 Teste do sistema resultante 76
4.5 Exemplo – Instanciação do GREN para Sistema de Acompanhamento e Reparo de Buracos 77
4.5.1 Análise do SARB 77
4.5.2 Mapeamento do SARB para o GREN 77
4.5.3 Implementação das classes do SARB 79
4.5.4 Teste do SARB 81
4.5.5 Comentários 82
4.6 Considerações Finais 83

5 O Processo de Construção e Utilização de um *Wizard* baseado na Linguagem de Padrões 84
5.1 Considerações Iniciais 84
5.2 Alternativas para Construção de um *Wizard* 85
5.3 Arquitetura do *Wizard* 86
5.4 Construção de um *Wizard* específico 88
5.5 Exemplo – GREN-Wizard: Uma ferramenta de suporte ao GREN e à GRN 91
5.5.1 Visão Geral do GREN-Wizard 91
5.5.2 Interface Gráfica do GREN-Wizard 91
5.5.3 Arquitetura do GREN-Wizard 94
5.5.4 O processo de Construção do GREN-Wizard 104
5.6 O Processo de Utilização do *Wizard* 105

5.7 Exemplo – Utilização do GREN-Wizard para geração de um Sistema de Controle de Reparo de Buracos . . . 106
5.8 Desenvolvimento de um *Wizard* Genérico . . . 107
5.8.1 Construção de um *Wizard* genérico . . . 109
5.8.2 Adaptação de um *Wizard* genérico . . . 117
5.9 Considerações Finais . . . 120

6 Avaliação da Proposta 122
6.1 Considerações Iniciais . . . 122
6.2 Avaliação do processo de construção de linguagens de padrões . . . 123
6.3 Avaliação da utilidade de linguagens de padrões na modelagem de sistemas . . . 124
6.3.1 Visão Geral . . . 124
6.3.2 E-GRN-1: Avaliação da GRN na modelagem de sistemas . . . 124
6.3.3 E-GRN-2: Avaliação da GRN na modelagem de sistemas . . . 131
6.3.4 E-GRN-3: Avaliação da GRN na modelagem de sistemas . . . 136
6.3.5 Outras avaliações . . . 140
6.4 Avaliação da utilidade de uma linguagem de padrões na construção de um framework correspondente . . . 141
6.4.1 Visão Geral . . . 141
6.4.2 A GRN na construção do GREN . . . 142
6.4.3 A LV na construção do Qd+ . . . 142
6.5 Avaliação da utilidade de uma linguagem de padrões na instanciação do framework associado . . . 143
6.5.1 Visão Geral . . . 143
6.5.2 E-GRN-GREN: Avaliação da GRN na instanciação do GREN . . . 143
6.5.3 A LV na instanciação do Qd+ . . . 149
6.5.4 Outros Casos de Uso da GRN na instanciação do GREN . . . 149
6.6 Avaliação da utilidade de uma linguagem de padrões na construção de um *Wizard* . 150
6.7 Avaliação da utilidade de uma linguagem de padrões na instanciação de um framework usando seu *Wizard* . . . 151
6.8 Avaliação da utilidade de um *Wizard* no teste do framework associado . . . 154
6.9 Avaliação do Framework GREN . . . 155
6.10 Avaliação do GREN-Wizard . . . 156
6.11 Considerações Finais . . . 158

7 Conclusões 159
7.1 Considerações Iniciais . . . 159
7.2 Resumo do Trabalho Efetuado . . . 159
7.3 Contribuições . . . 160
7.4 Limitações do Trabalho Efetuado . . . 161
7.5 Sugestões de trabalhos futuros . . . 162
7.6 Trabalhos decorrentes desta pesquisa . . . 163

Referências 165

Apêndices **172**

Apêndice A - Requisitos para o Sistema de Hotel
Apêndice B - Requisitos para o Sistema de Locadora de Carros
Apêndice C - Requisitos para o Sistema de Clínica Veterinária
Apêndice D - Formulários usados no E-GRN-1 a 3
Apêndice E - Formulários usados no E-GRN-GREN
Apêndice F - Documentação adicional do GREN e do GREN-Wizard
Apêndice G - Figuras Referentes aos Processos

Lista de Figuras

2.1 Padrão Composição (Gamma et al., 1995) 10
2.2 Relacionamento entre Linguagens de Padrões, Frameworks e Aplicações 20
2.3 Arquitetura do framework IBM SanFrancisco (Monday et al., 2000) 24
2.4 Estrutura do OmniBuilder (OMNISPHERE, 2002) 26
2.5 Linguagem de Padrões G++ (Aarsten et al., 2000) 28

3.1 Ilustração da Abordagem Proposta 36
3.2 Processo Proposto 36
3.3 Processo de Construção da Linguagem de Padrões 38
3.4 GRN: Uma Linguagem de Padrões para Gestão de Recursos de Negócios 47
3.5 Padrão 2 - Quantificar o Recurso 49
3.6 Exemplo de uso do Padrão 2 50
3.7 Modelo de Análise do SARB com padrões 57

4.1 Processo para construção de um framework baseado em uma linguagem de padrões 61
4.2 Identificação dos pontos variáveis do framework 62
4.3 Projeto do Framework 64
4.4 Criação da Hierarquia de Classes do Framework 65
4.5 Implementação do Framework 66
4.6 Validação do Framework 68
4.7 Arquitetura do GREN 70
4.8 Exemplo de algumas classes do GREN 71
4.9 Processo de Instanciação do Framework Caixa branca 74
4.10 Parte da Hierarquia de Classes do GREN e classes derivadas no SARB 80
4.11 Menus e janela principal do SARB 81
4.12 Exemplo de GUI para Ordem de Serviço (SARB) 82

5.1 Alternativas possíveis para construção de um *Wizard* 86
5.2 Arquitetura do *Wizard* para instanciação de um framework baseado em uma linguagem de padrões 87
5.3 Processo para construção de um *Wizard* específico para instanciação de um framework baseado em uma linguagem de padrões 89
5.4 Exemplo da GUI do GREN-Wizard 92

5.5 Arquitetura do GREN-Wizard 94
5.6 Diagrama de classes do módulo de especificação do domínio da GRN 95
5.7 Exemplo de Conteúdo de algumas tabelas de especificação do domínio da GRN . . 97
5.8 Diagrama de classes do módulo de especificação do domínio da GRN 98
5.9 Diagrama de classes do módulo GUI do GREN-Wizard 100
5.10 Parte do algoritmo para criar novas classes da aplicação no GREN 102
5.11 Parte do algoritmo para sobrepor métodos-gancho no GREN 102
5.12 Algoritmo para criar o corpo do método-gancho `resourceClass` no GREN . . . 103
5.13 Algoritmo para gerar o script de criação de tabelas MySQL no GREN-Wizard . . 104
5.14 Processo para utilização de um *Wizard* na geração de uma aplicação específica . . . 106
5.15 GUI do GREN-Wizard após alimentação com informações sobre o SARB (padrão 1) 107
5.16 GUI do GREN-Wizard após alimentação com informações sobre o SARB (atributos incluídos) 108
5.17 GUI do GREN-Wizard após alimentação com informações sobre o SARB (padrão 9) 109
5.18 GUI do GREN-Wizard após alimentação com informações sobre o SARB (seleção de relatórios) 110
5.19 Processo para desenvolvimento de *Wizards* para instanciação de frameworks baseados em linguagens de padrões 110
5.20 Processo para construção de um *Wizard* genérico para instanciação de frameworks baseados em linguagens de padrões 111
5.21 Meta-modelo para uma Linguagem de Padrões 112
5.22 Meta-modelo para aplicações geradas usando a Linguagem de Padrões 113
5.23 Ilustração da correspondência entre uma padrão e sua implementação no framework 113
5.24 Processo para adaptação do *Wizard* para um framework e linguagem de padrões específicos 118

6.1 Representação gráfica dos resultados do E-GRN-1 nas duas abordagens - Sistema Hotel 130
6.2 Representação gráfica dos resultados do E-GRN-1 nas duas abordagens - Sistema Locadora de Carros 130
6.3 Representação gráfica dos resultados do E-GRN-2 nas duas abordagens 135

8.1 Processo Proposto 173
8.2 Processo de Construção da Linguagem de Padrões 173
8.3 Processo para construção de um framework baseado em uma linguagem de padrões 174
8.4 Identificação dos pontos variáveis do framework 174
8.5 Projeto do Framework 175
8.6 Criação da Hierarquia de Classes do Framework 175
8.7 Implementação do Framework 175
8.8 Validação do Framework 176
8.9 Processo de Instanciação do Framework Caixa branca 176
8.10 Processo para construção de um *Wizard* específico para instanciação de um framework baseado em uma linguagem de padrões 177
8.11 Processo para desenvolvimento de *Wizards* para instanciação de frameworks baseados em linguagens de padrões 177

8.12 Processo para construção de um *Wizard* genérico para instanciação de frameworks baseados em linguagens de padrões 178
8.13 Processo para adaptação do *Wizard* para um framework e linguagem de padrões específicos 179
8.14 Processo para utilização de um *Wizard* na geração de uma aplicação específica . . . 179

Lista de Tabelas

3.1 Histórico de padrões usados na instanciação do SARB 58

4.1 Lista parcial dos Pontos Variáveis do Framework GREN 69
4.2 Exemplo da documentação do GREN - Tabela usada para identificar novas classes da aplicação 72
4.3 Exemplo da documentação do GREN - Tabela usada para identificar novas classes da GUI 73
4.4 Exemplos de métodos a serem sobrepostos no GREN 73
4.5 Algumas classes a serem criadas no SARB (TC) 79

6.1 Tipos de erros considerados no Estudo de Caso com a GRN 126
6.2 Projeto do E-GRN-1 126
6.3 Área de interesse dos alunos do E-GRN-1 127
6.4 Experiência dos alunos do E-GRN-1 128
6.5 Resultados do E-GRN-1 – Abordagem *Ad hoc* 129
6.6 Resultados do E-GRN-1 – Abordagem GRN 129
6.7 Resultados Estatísticos para E-GRN-1 131
6.8 Projeto do E-GRN-2 133
6.9 Área de interesse dos alunos do E-GRN-2 133
6.10 Experiência dos alunos do E-GRN-2 133
6.11 Resultados do E-GRN-2 – Abordagem *Ad hoc* 134
6.12 Resultados do E-GRN-2 – Abordagem GRN 134
6.13 Projeto do E-GRN-3 137
6.14 Área de interesse dos alunos do E-GRN-3 137
6.15 Experiência dos alunos do E-GRN-3 138
6.16 Cargos desempenhados pelos alunos do E-GRN-3 138
6.17 Resultados do E-GRN-3 – Abordagem *Ad hoc* 139
6.18 Resultados do E-GRN-3 – Abordagem GRN 139
6.19 Projeto do E-GRN-4 145
6.20 Resultados do E-GRN-GREN 146
6.21 Manutenção realizada no GREN e GREN-Wizard 154
6.22 Medidas de uso do GREN-Wizard 156

Lista de Abreviaturas

E-GRN-1 Experimento com a linguagem de padrões GRN - número 1

E-GRN-2 Experimento com a linguagem de padrões GRN - número 2

E-GRN-3 Experimento com a linguagem de padrões GRN - número 3

E-GRN-GREN Experimento com a linguagem de padrões GRN e o framework GREN

GUI Interface Gráfica com o Usuário (do inglês *Graphical User Interface*)

GREN Framework para Gestão de Recursos de Negócios

GRN Linguagem de Padrões para Gestão de Recursos de Negócios

LOC Linhas de Código (do inglês *Lines Of Code*)

LV Linguagem de Padrões para Leilões Virtuais

PLoP Conferência sobre Linguagens de Padrões em Programação (do inglês *Pattern Language of Programs*)

Qd+ Framework para Gestão de Leilões Virtuais

SARB Sistema de Acompanhamento e Reparo de Buracos

SGBD Sistema Gerenciador de Banco de Dados

UML Linguagem de Modelagem Unificada (do inglês *Unified Modeling Language*)

Resumo

UM PROCESSO PARA CONSTRUÇÃO E INSTANCIAÇÃO DE FRAMEWORKS BASEADOS EM UMA LINGUAGEM DE PADRÕES PARA UM DOMÍNIO ESPECÍFICO

Padrões, linguagens de padrões e frameworks são formas de reuso de software. A complexidade de frameworks, causada pela dificuldade tanto em construí-los quanto em utilizá-los, é um dos inibidores do uso dessa tecnologia. Um processo para facilitar a construção e instanciação de frameworks é apresentado. Uma linguagem de padrões é utilizada para apoiar todo o processo, desde a identificação da funcionalidade do framework, seu projeto, implementação, validação, até sua instanciação para sistemas específicos do domínio. O framework obtido é do tipo caixa-branca e sua instanciação é feita especializando-se suas classes para sistemas específicos. O processo inclui, também, a construção de uma ferramenta para automatizar a instanciação do framework, por meio da qual é possível obter um sistema específico fornecendo apenas informações sobre os padrões da linguagem utilizados na sua modelagem. O processo é ilustrado com a Linguagem de Padrões para Gestão de Recursos de Negócios (GRN), que serviu de base para construção do framework GREN. Apresenta-se também o GREN-Wizard, uma ferramenta para instanciação automática do GREN para sistemas no domínio da GRN. A avaliação do processo é feita por meio de alguns experimentos e vários relatos de uso dos diversos sub-processos que compõem o processo geral, usando o GREN, a GRN e o GREN-Wizard.

PALAVRAS-CHAVE: Reuso de Software, Frameworks Orientados a Objetos, Padrões, Linguagens de padrões, Padrões de Análise

Abstract

A PROCESS FOR CONSTRUCTION AND INSTANTIATION OF FRAMEWORKS BASED ON A DOMAIN-SPECIFIC PATTERN LANGUAGE

Patterns, pattern languages, and frameworks are ways of enhancing software reuse. The complexity of software frameworks, caused both by the difficulty to build and to use them, is one of the inhibitors of this technology. A process to ease the construction and instantiation of frameworks is presented, in which a pattern language is used to support the whole process, starting from the identification of the framework functionality, its design, implementation, validation, and its instantiation to domain-specific systems. The framework that results from applying the proposed process is white-box and its instantiation is done by specializing its abstract classes according to a specific system. The process includes the construction of a tool to automate the framework instantiation, through which it is possible to obtain a specific system only by supplying information about the language patterns used to model it. The proposed process is illustrated with the Pattern Language for Business Resource Management (GRN), which was used as the basis for the GREN framework construction. The GREN-Wizard is also presented, which is a tool to automatically instantiate the GREN framework to specific applications in the GRN domain. The proposed process is evaluated by a few experiments and several uses of the sub-processes that compose the general process, using GRN, GREN and the GREN-Wizard.

KEY-WORDS: Software Reuse, Object-oriented Frameworks, Patterns, Pattern Languages, Analysis Patterns

CAPÍTULO

1

Introdução

1.1 Contextualização

A reutilização de software é um propósito enunciado quase que simultaneamente com o surgimento da própria Engenharia de Software (Bosch et al., 1999). Já nos anos 70, com a definição da programação estruturada, os módulos podiam ser vistos como componentes reusáveis em sistemas novos. Até então o reuso era conseguido copiando-se o código e editando-o para adaptá-lo ao novo sistema.

Nos anos 80, com a popularização da programação orientada a objetos, o reuso por meio de herança mostrou-se mais poderoso na adaptação do código a novos sistemas. O reuso obtido era ainda apenas em pequena escala, consistindo de componentes de código a serem utilizados na construção de sistemas maiores (Bosch et al., 1999). O paradigma da orientação a objetos não resolvia o problema, bem mais complexo, de reutilizar componentes grandes que cuidassem de parte significativa de um sistema.

A análise de domínios ganhou forças, também na década de 80, com o objetivo de maximizar a reutilização no desenvolvimento de software, por meio da identificação e organização do conhecimento a respeito de uma classe de problemas – um domínio de aplicações (Prieto-Diaz e Arango, 1991). Ela é um passo fundamental na criação de artefatos de software reutilizáveis, já que o modelo nela produzido capta a funcionalidade essencial requerida por um domínio e pode ser reutilizado durante a análise de sistemas de tais domínios.

Nesse mesmo contexto surgiram os frameworks de software orientados a objetos, que permitem o reuso de grandes estruturas em um domínio particular, as quais são personalizadas para atender aos requisitos de aplicações específicas desse domínio (Johnson e Foote, 1988; Johnson e Russo, 1991). Assim, famílias de aplicações similares, mas não idênticas, podem ser derivadas a partir de um único framework.

Ainda com o objetivo de reuso em diferentes níveis de abstração, surgiram também na década de 90, os padrões de software (Buschmann et al., 1996; Coad, 1992; Coplien, 1992; Gamma et al., 1995), que tentam captar a experiência adquirida no desenvolvimento de software e sintetizá-la em forma de problema e solução. Além de prover o reuso das soluções, os padrões ajudam a melhorar a comunicação entre desenvolvedores, que podem conduzir suas discussões com base nos nomes dos padrões (Gamma et al., 1995). Para desenvolvedores que conhecem os padrões, a simples citação de seu nome traz consigo um conteúdo semântico bastante significativo, o que dispensa a explicação dos detalhes envolvidos na solução.

Embora os primeiros frameworks tenham surgido para domínios mais amplos, como por exemplo interface com o usuário, existe uma preocupação cada vez maior em desenvolver frameworks para domínios de aplicação específicos, como negócios, engenharia, medicina, seguros, etc. Em particular, o ambiente de negócios está se tornando mais dinâmico com o aumento da competição mundial e mudanças no mercado (Carey et al., 2000), demandando novas tecnologias a fim de diminuir o esforço necessário para o desenvolvimento de aplicações com maior rapidez e eficiência.

Entretanto, frameworks são, na maioria das vezes, muito complexos para construir, entender e utilizar. Poucos são os métodos para desenvolvimento de frameworks e muitas das técnicas para sua documentação enfatizam o projeto do framework, não evidenciando como utilizá-lo na construção de aplicações. Assim, embora a descoberta de novas técnicas para reuso seja uma questão significativa na Engenharia de Software (tendo inclusive o reuso se firmado como uma das sub-áreas da Engenharia de Software), não existe uma técnica que tire proveito das vantagens de linguagens de padrões, que podem atuar com um guia no desenvolvimento de aplicações. O trabalho aqui proposto encaixa-se nesse contexto, propondo uma abordagem para o desenvolvimento de um framework com base em uma linguagem de padrões num particular domínio, de forma a facilitar a posterior utilização do framework, por meio de sua vinculação à linguagem de padrões correspondente.

1.2 Motivação

A pesquisa sobre linguagens de padrões é considerada de muita importância. Schmid et al. (1996) discutem aspectos de padrões que deverão receber atenção especial da comunidade de software, mencionando como um desses aspectos, especificamente, a integração de padrões de projeto para

formar linguagens de padrões. Apesar de trabalhosa e desafiadora, essa integração promete grandes recompensas ao esforço investido. Kerth e Cunningham (1997) discutem o uso de padrões no contexto da arquitetura de software e, em especial, destacam que linguagens de padrões agem como ponte entre a arquitetura e o projeto, assegurando que a filosofia dessa arquitetura possa ser difundida, ensinada e seguida.

A experiência no desenvolvimento de sistemas permite o estabelecimento de padrões de software a serem utilizados em futuros empreendimentos. Dentro de sua prática profissional de mais de dez anos no desenvolvimento de sistemas de informação para pequenas e médias empresas, a autora desta tese, sempre que possível, reutilizou software. Porém, apenas os mecanismos mais simples de reuso, como a cópia e adaptação de código, foram empregados até então. Sua experiência em desenvolvimento no domínio de gestão de recursos de negócios permitiu-lhe propor uma família de padrões para gestão de recursos de negócios (Braga et al., 1998). Muitos foram os incentivos para que tal família de padrões se convertesse em uma linguagem de padrões composta por padrões menores, que apoiassem a modelagem de sistemas desse domínio. A construção de um framework para apoiar a linguagem de padrões constitui, portanto, um passo adiante em direção ao reuso também de projeto e código.

Se o projeto de aplicações de negócios é difícil, o projeto de componentes reusáveis é mais difícil e o projeto de frameworks, com base nos quais múltiplas aplicações de negócios podem ser construídas, é ainda mais difícil (Carey et al., 2000). Assim, o investimento na construção de frameworks é grande, mas pode ser compensado à medida que se utilize o framework para derivar muitas aplicações. O domínio de sistemas de informação para gestão de recursos de negócios possui mercado bastante abrangente. Muitos sistemas legados ainda são largamente utilizados, principalmente no Brasil, mesmo tendo sido desenvolvidos com tecnologia considerada ultrapassada nos dias de hoje. A existência de frameworks nesse domínio pode motivar e agilizar a reengenharia de tais sistemas. Além disso, outros domínios relacionados ao domínio de gestão de recursos de negócios podem se beneficiar de tecnologias desenvolvidas nessa área, como por exemplo os domínios de seguradoras, hospitais, escolas, órgãos governamentais, etc.

Pesquisas sobre frameworks têm sido amplamente conduzidas (Pree, 1999; Roberts e Johnson, 1998; Schmid, 1997) com o intuito de descobrir técnicas mais eficientes não somente para construí-los, mas também para torná-los mais fáceis de utilizar e manter, por desenvolvedores não familiarizados com sua estrutura. A vinculação de uma linguagem de padrões a um framework pode solucionar esse problema, à medida que permite a utilização do framework por meio de uma linguagem situada em um nível de abstração mais alto do que classes, e facilita o entendimento do framework por meio dos padrões de análise por ele implementados.

Além disso, outra necessidade atual, identificada por Bosch et al. (1999) (página 78), é a avaliação da possibilidade de uso de um framework para desenvolver uma aplicação específica.

Se o framework tiver sido construído com base em uma linguagem de padrões, esta pode fornecer meios para verificar sua adequação para construção de aplicações específicas. Ademais, a aplicação da linguagem de padrões pode indicar exatamente os pontos não cobertos e que precisam ser desenvolvidos ou adaptados. Isso também gera novas oportunidades de extensão do framework para atender a novos aspectos do domínio, ou até mesmo a novos domínios.

O forte relacionamento existente entre uma linguagem de padrões e seu framework associado desperta o interesse de construir uma ferramenta para automatizar o desenvolvimento de aplicações derivadas a partir do framework. Essa ferramenta poderia, de acordo com informações sobre os padrões utilizados na modelagem de um sistema específico, usar o mapeamento da linguagem de padrões para o framework na identificação de quais classes do framework devem ser especializadas para atender aos requisitos particulares do sistema.

1.3 Objetivos

Esta tese tem como objetivo principal propor um processo geral para desenvolvimento e instanciação de um framework a partir de uma linguagem de padrões de análise para um domínio específico. Esse processo deverá ser detalhado suficientemente para permitir sua utilização por outros desenvolvedores de frameworks, originando, portanto, um novo método na área de Engenharia de Software.

O trabalho tem também os seguintes objetivos: propor diretrizes para construção de uma linguagem de padrões para um domínio específico; fornecer um processo para uso da linguagem de padrões na modelagem de sistemas do domínio; fornecer um processo bem definido para construção de um framework com base na linguagem de padrões desenvolvida; fornecer um processo que facilite a instanciação do framework para aplicações específicas do domínio; fornecer um processo que ajude na criação de uma ferramenta de instanciação automática do framework com base na linguagem de padrões correspondente; e fornecer um processo para guiar a instanciação de aplicações usando a ferramenta criada.

O processo geral visa atender às expectativas de diversos profissionais da Engenharia de Software: o construtor de frameworks poderá beneficiar-se do processo de construção de frameworks com base em linguagens de padrões; o construtor de aplicações experiente poderá usufruir do framework para obter suas aplicações concretas num tempo muito menor do que se as implementasse a partir do zero (em inglês, *from scratch*); e, finalmente, o construtor de aplicações inexperiente poderá utilizar a linguagem de padrões para modelar sua aplicação concreta e, por meio da ferramenta, obter a implementação de tais aplicações apenas informando os padrões utilizados durante a modelagem.

1.4 Organização

Este trabalho está organizado em sete capítulos e sete apêndices. No Capítulo 2, são fornecidos os conceitos que proporcionam o embasamento teórico necessário à compreensão do trabalho de doutorado aqui descrito e dos pontos em aberto que puderam ser alvo do estudo realizado. O destaque é para os trabalhos relacionados ao domínio aqui tratado.

No Capítulo 3, dá-se uma visão geral do processo proposto para desenvolvimento de um framework com base em uma linguagem de padrões, composto de quatro passos: desenvolvimento da linguagem de padrões, construção do framework caixa branca, construção do *Wizard* e instanciação do framework. Ainda no Capítulo 3 mostra-se em detalhes o primeiro passo do processo proposto, exemplificando-o com uma linguagem de padrões para gestão de recursos de negócios (GRN), desenvolvida durante esta pesquisa de doutorado.

No Capítulo 4 é enfocado, em detalhes, o segundo passo do processo, que refere-se ao sub-processo de construção de um framework caixa branca com base em uma linguagem de padrões. Esse passo é exemplificado pelo framework GREN, construído a partir da linguagem de padrões GRN. Mostra-se também, nesse Capítulo, o sub-processo de instanciação manual do framework, que constitui parte do quarto passo da abordagem, usando como exemplo a instanciação do framework GREN para desenvolvimento de um sistema de controle de reparos de buracos em uma cidade.

No Capítulo 5, define-se o terceiro passo do processo geral, que trata da construção de uma ferramenta (*Wizard*, em inglês) para automatizar o processo de instanciação do framework. Esse passo é exemplificado com a ferramenta GREN-Wizard, implementada para dar suporte à linguagem de padrões GRN e ao framework GREN. Também é mostrado, nesse Capítulo, o sub-processo de instanciação usando o *Wizard*, exemplificado usando o GREN-Wizard para desenvolvimento do mesmo sistema citado no Capítulo 4. Esse sub-processo faz parte do quarto passo do processo geral proposto.

No Capítulo 6, avaliam-se os processos propostos e os produtos obtidos, por meio de relatos de uso dos mesmos e de experimentos realizados com a linguagem de padrões GRN e o framework GREN. O objetivo é avaliar as vantagens, desvantagens e lições aprendidas durante esses usos e experimentos.

No Capítulo 7, apresentam-se as conclusões desta tese, com um resumo do trabalho efetuado, as contribuições ao estado da arte, as limitações da abordagem proposta e sugestões de trabalhos futuros.

Nos Apêndices A, B e C estão os documentos de requisitos para os sistemas de hotel, locadora de carros e veterinária, respectivamente. Nos Apêndices D e E encontram-se os formulários utilizados para coleta de dados dos experimentos E-GRN-1 a E-GRN-3 e E-GRN-GREN, respec-

tivamente. No Apêndice F apresenta-se documentação adicional do GREN e do GREN-Wizard, constituída da hierarquia de classes e de diversos diagramas de classes. No Apêndice G são reproduzidas as figuras referentes aos processos e sub-processos propostos nesta tese.

CAPÍTULO

2

Situação do Processo Proposto no Panorama da Literatura Técnica Especializada

2.1 Considerações Iniciais

Neste Capítulo é relatada a situação do processo proposto nesta tese em relação ao panorama da literatura técnica especializada. Para tal, na seção 2.2 são definidos os conceitos sobre frameworks, padrões e linguagens de padrões, que formam o embasamento teórico necessário para compreensão da abordagem proposta nesta tese. Nas seções 2.3 e 2.4 são apresentados os padrões e frameworks, respectivamente, de relevância para o domínio tratado nesta tese. Na seção 2.5 são discutidos os trabalhos relevantes que relacionam padrões ou linguagens de padrões a frameworks. Nas seções 2.6 e 2.7 apresentam-se os trabalhos relacionados à construção e instanciação de frameworks, respectivamente, identificando os problemas dessas abordagens que dão origem a oportunidades de pesquisa. Na seção 2.8 são apresentadas as considerações finais sobre este capítulo.

2.2 Conceitos básicos relacionados à tese

2.2.1 Padrões e Linguagens de Padrões

O conceito de padrões de software foi inspirado na área de arquitetura, mais especificamente no trabalho de Alexander (Alexander, 1979; Alexander et al., 1977), que diz: "Cada padrão descreve um problema que ocorre repetidas vezes em nosso ambiente, e então descreve o núcleo da solução para esse problema, de forma que você possa utilizar essa solução milhões de vezes sem usá-la do mesmo modo duas vezes". Em 1987, Beck e Cunningham utilizaram essas idéias no desenvolvimento de uma pequena linguagem de padrões para guiar programadores inexperientes em Smalltalk (Beck e Cunningham, 1987). Logo em seguida, Coplien catalogou vários estilos (*idioms*) em C++, que consistem em um tipo de padrão, tendo-os publicado em um livro alguns anos depois (Coplien, 1992). De 1990 a 1992, vários autores iniciaram a criação de um catálogo de padrões de projeto, que vieram posteriormente a ser publicados em um livro (Gamma et al., 1995). Foi esse livro, posteriormente denominado "GoF book", que iniciou a disseminação de padrões de software para a comunidade científica. As conferências *Pattern Language of Programs* (PLoP)[1], são outra fonte de produção e disseminação de padrões. Nelas têm sido apresentados padrões dos mais diferentes níveis de abstração, desde padrões de processo, arquiteturais, organizacionais, de análise, de projeto, de programação, etc.

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

De fato, quando especialistas trabalham em um problema particular, é raro que inventem uma nova solução para resolvê-lo que seja diferente das já existentes. Diversas soluções são conhecidas por tais especialistas, de acordo com experiência própria ou de outros profissionais. Assim, ao confrontarem-se com novos problemas, é comum lembrarem-se de outros similares e reutilizarem a solução antiga, pensando em pares "problema/solução". Esses pares podem ser agrupados em famílias de problemas e soluções similares, sendo que cada família exibe um padrão tanto de problema quanto de solução (Buschmann et al., 1996). Projetistas familiarizados com certos padrões podem aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los (Gamma et al., 1995).

[1]Essas Conferências ocorrem anualmente em diversos locais no mundo todo: a mais tradicional é a PLoP realizada nos Estados Unidos anualmente a partir de 1994, mas existem outras como a EuroPLoP (a partir de 1995), a ChiliPLoP (a partir de 1998) e a KoalaPLoP (a partir de 2000). No Brasil, já foram realizadas duas edições da SugarLoafPLoP – "Conferência Latino-Americana em Linguagens de Padrões para Programação".

Além de fornecerem exemplos a serem seguidos e artifícios a serem copiados e posteriormente refinados ou estendidos, os padrões garantem uniformidade na estrutura do software, aumentando a produtividade no seu desenvolvimento e manutenção (Gall et al., 1996). Também, proporcionam o reuso do conhecimento obtido por projetistas experientes, ajudando um novato a agir como um especialista (Gamma et al., 1993). O processo de desenvolvimento de software orientado a objetos pode ser facilitado pelo uso de padrões de projeto. Eles proporcionam um vocabulário comum para a comunicação entre projetistas, criando abstrações num nível superior ao de classes e instâncias (Gamma et al., 1993).

Com o intuito de facilitar o reuso dos padrões de software, pode-se classificá-los em diversas categorias. Porém, não deve haver ortogonalidade nessa classificação, pois pode haver padrões que se encaixam em mais do que uma categoria. Padrões de processo definem soluções para os problemas encontrados nos processos envolvidos na Engenharia de Software: desenvolvimento, controle de configuração, testes, etc. Padrões arquiteturais expressam o esquema ou organização estrutural fundamental de sistemas de software ou hardware. Padrões de análise descrevem soluções para problemas de análise de sistemas, embutindo conhecimento sobre um domínio de aplicação específico. Padrões de projeto definem soluções para problemas de projeto de software, enquanto padrões de programação descrevem soluções de programação particulares de uma determinada linguagem ou regras gerais de estilo de programação. Este trabalho trata mais especificamente de padrões de análise e projeto, em particular de padrões de análise para gestão de recursos de negócios e padrões de projeto úteis no desenvolvimento de frameworks de software.

A Figura 2.1 mostra, de forma resumida, o padrão "Composição" (do inglês *Composite*) proposto por Gamma et al. (1995). Este padrão tem sido mencionado em muitos artigos sobre padrões e frameworks, por ser aplicável a projetos em diversos domínios e por ser útil no projeto de pontos variáveis de um framework. Ele pode ser aplicado na representação de hierarquias "todo-parte", quando deseja-se ignorar a diferença entre composições de objetos e objetos individuais.

Além do agrupamento de padrões segundo seu nível de abstração, o qual proporciona a recuperação de um padrão de acordo com a fase de desenvolvimento em que se encontra, pode-se reuni-los em coleções, catálogos, sistemas e linguagens. Em particular, neste trabalho será de grande importância compreender o significado de uma linguagem de padrões, que é uma coleção estruturada de padrões que se apóiam uns nos outros para transformar requisitos e restrições numa arquitetura (Coplien, 1998). Seus padrões constituintes cobrem todos os aspectos importantes de um dado domínio e pelo menos um padrão deve estar disponível para cada aspecto da construção e implementação de um sistema de software: não podem haver "vazios" ou "brancos". Uma linguagem de padrões representa a seqüência temporal de decisões que levam ao projeto completo de uma aplicação, de forma a tornar-se um método para guiar o processo de desenvolvimento (Brugali et al., 2000).

COMPOSIÇÃO

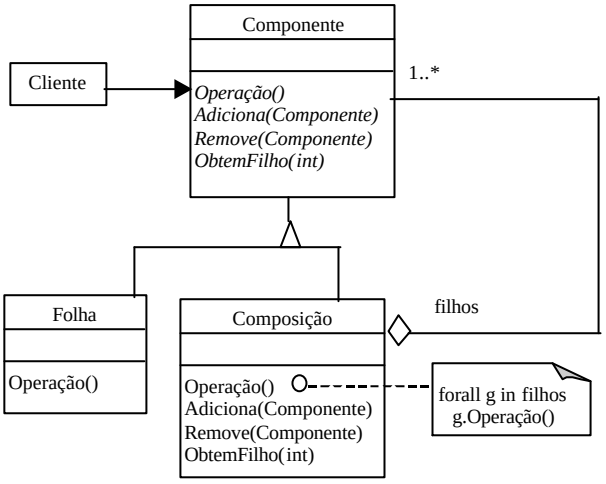
Intenção
Compor objetos em estruturas de árvore para representar hierarquias "todo-parte". Composição permite que clientes tratem de maneira uniforme objetos individuais e composições de objetos.

Motivação
Aplicações gráficas como editores gráficos permitem que diagramas complexos sejam construídos a partir de componentes simples.
. . .

Aplicabilidade
Utilize o padrão Composição quando:
- Você precisa representar hierarquias de objetos do tipo "todo-parte"
- Você quer permitir a seus clientes ignorar a diferença entre composições de objetos e objetos individuais

. . .

Estrutura



Participantes

Componente: declara a interface para os objetos da composição; implementa o comportamento *default* para a interface comum a todas as classes; declara uma interface para acessar e gerenciar os componentes-filho; . . .
Folha: representa os objetos-folha da composição. Uma folha não tem filhos; define o comportamento para os objetos primitivos da composição.
Composição: define o comportamento para os componentes que possuem filhos; armazena os componentes-filho; implementa operações relacionadas ao filhos na interface do Componente;
Cliente: manipula os objetos da composição por meio da interface do Componente.

Colaborações
O Cliente usa a interface do Componente para interagir com objetos da estrutura de composição. Se o recipiente for uma Folha, então é manipulado diretamente. Se for uma Composição, usualmente transmite a requisição a seus componentes-filho, possivelmente realizando operações adicionais antes ou depois dessa transmissão.

Conseqüências
. . .

Implementação
. . .

Exemplo de Código
. . .

Usos conhecidos
Exemplos do padrão Composição podem ser encontrados na maioria dos sistemas orientados a objetos. A classe original *View* do *Smalltalk Model/View/Controller* (MVC) é uma Composição. A maioria das ferramentas ou frameworks para interface com o usuário seguiu os mesmos passos do MVC, como por exemplo o ET++, *InterViews*, *Graphics* e *Glyphs*. . . .

Padrões Relacionados
. . .

Figura 2.1: Padrão Composição (Gamma et al., 1995)

Como uma linguagem de padrões possui vários padrões, grande parte da linguagem é formada pelos padrões em si. Outra parte consiste da visão geral do domínio coberto pela linguagem, de sua aplicabilidade e de uma visão geral da interação entre os padrões. Considerando esta outra parte da linguagem, Meszaros e Doble (1998) sugerem que toda linguagem de padrões ofereça um sumário

da linguagem, um sumário dos problemas e soluções de cada padrão, um exemplo de aplicação da linguagem e um glossário de termos. O sumário da linguagem deve explicar o porquê dos padrões estarem juntos, as linhas comuns encontradas em mais de um padrão e como os padrões podem trabalhar juntos na execução de algo útil. O sumário dos problemas e soluções consiste em uma tabela contendo um resumo de cada problema e a solução correspondente. Um exemplo deve ser escolhido para ilustrar, de preferência, todos os padrões da linguagem. Um glossário de termos deve explicar a terminologia que não seja familiar aos usuários.

Considerando os padrões em si, diversos formatos podem, em geral, ser adotados para descrevê-los. Dentre eles, pode-se citar o formato de Portland, o formato GoF e o formato de Appleton. Em todos eles o padrão é descrito por meio de diversos elementos, também chamados de seções por Gamma et al. (1995) ou componentes por Appleton (1997). O formato de Portland foi inspirado no formato de Alexander, o arquiteto e autor do livro "A Pattern Language" (Alexander et al., 1977). Consiste de apenas três elementos: após o nome do padrão, um parágrafo é escrito contendo a contextualização e descrição do problema. Coloca-se então a palavra "portanto" (*therefore*, em inglês), seguida de mais um parágrafo contendo a solução do problema, conseqüências, colaborações, etc. Este formato é bastante utilizado em linguagens de padrões.

No formato de GoF[2] cada padrão contém os seguintes elementos: nome do padrão; intenção; também conhecido como; motivação; aplicabilidade; estrutura; participantes; colaborações; conseqüências; implementação; código exemplo; usos conhecidos e padrões relacionados. O formato de Appleton foi elaborado com base em diversos formatos utilizados por autores de padrões, numa tentativa de uniformização dos formatos. No formato de Appleton, cada padrão possui os seguintes elementos: nome; problema; contexto; influências; solução (a qual pode ter subseções estrutura, participantes, dinâmica, implementação e variantes); exemplos; contexto resultante; justificativa; padrões relacionados e usos conhecidos. Vale ressaltar que esses formatos são apenas sugestões oferecidas aos autores de padrões, que devem escolher os elementos apropriados a cada padrão em particular. Segundo Vlissides (1998), é necessário refletir sobre o padrão sendo escrito e não se prender, a princípio, a nenhum formato de escrita. Não há, também, obrigatoriedade de uso de todos os elementos em todos os padrões.

Na linguagem de padrões apresentada no Capítulo 3, o formato de Appleton foi tomado como base para criação de um formato personalizado, composto dos seguintes elementos: nome; contexto; problema; influências; estrutura; participantes; exemplos; variantes e próximos padrões. O elemento "próximos padrões" foi adicionado a cada um dos padrões para fazer a ligação desse com os demais padrões da linguagem. Esse elemento substitui o elemento "Padrões Relacionados" e contém informações que ajudam na tomada de decisão que se faz necessária quando a linguagem é aplicada ao desenvolvimento de um sistema específico. Nesse elemento encontram-se as diretrizes

[2] A sigla GoF significa *Gang of Four*, apelido dado aos quatro autores do livro "Design Patterns" (Gamma et al., 1995).

sobre a ordem de aplicação dos padrões da linguagem, estabelecendo a ligação do padrão atual com os demais padrões da linguagem.

Além dos padrões propriamente ditos, a linguagem de padrões apresentada no Capítulo 3 possui uma introdução, um estudo de caso de aplicação da linguagem, um sumário dos problemas e soluções e uma seção contendo o relacionamento dos padrões da linguagem com padrões encontrados na literatura. Na introdução há uma visão geral do propósito da linguagem dentro do domínio tratado, uma figura ilustrando o relacionamento entre os padrões nela contidos e explicações sobre a notação utilizada.

2.2.2 Frameworks de Software Orientados a Objetos

São muitas as definições para framework de software orientado a objetos[3] encontradas na literatura. Contudo, todas têm em comum o enfoque na sua característica mais marcante: facilitar o reuso. Entende-se por reuso não só o aproveitamento de trechos de código de programas, mas também a reutilização de esforços de todas as fases do desenvolvimento de software, desde a análise dos requisitos, passando pelo projeto do software, implementação e testes.

A utilização de frameworks no desenvolvimento de aplicações em domínios específicos vem tendo sucesso por muitos anos (Fayad et al., 1999). Os primeiros frameworks surgiram no domínio de interface gráfica com o usuário, como é o caso dos frameworks MacApp, da Macintosh (Schmucker 86 *apud* (Fayad et al., 1999))[4]; Andrew Toolkit, da Universidade Carnegie Mellon (Pallay 88 *apud* (Fayad et al., 1999))[5]; ET++, da Universidade de Zurik (Weinand, 1989 *apud* (Fayad et al., 1999))[6]; e InterViews, de Stanford (Linton et al., 1989 *apud* (Fayad et al., 1999))[7].

Porém, logo começaram a surgir frameworks para outros domínios, como para algoritmos de roteamento VLSI (*very large scale integration*) (Gossain, 1990 *apud* (Fayad et al., 1999))[8], sistemas hipermídia (Meyrowitz, 1986 *apud* (Fayad et al., 1999))[9], sistemas operacionais (Russo, 1990 *apud* (Fayad et al., 1999))[10] e controle de manufatura (Schmid, 1995 *apud* (Fayad et al., 1999))[11].

[3]Para simplificação, o termo “framework” será utilizado nesta tese com o mesmo significado de “framework de software orientado a objetos”

[4]SCHMUCKER, K. J. Object-Oriented Programming for the Macintosh. Hayden Book Company, 1986.

[5]PALAY, A. et al. The Andrew Toolkit - an overview. Anais. Winter 1988 USENIX Conference, Dallas, TX, 1988.

[6]WEINAND, A. et al. Design and Implementation of ET++, a seamless object-oriented application framework. Structured Programming, 10(2), p. 63-87, 1989.

[7]LINTON, M. A.; VLISSIDES, J. M.; CALDER, P. R. Composing user interfaces with InterViews. Computer, V.22(2), p. 8-22, Fevereiro 1989.

[8]GOSSAIN, S. Object-oriented development and reuse. Tese de Doutorado, University of Esses, UK, 1990.

[9]MEYROWITZ, N. Intermedia: The architecture and construction of an object-oriented hypermedia system and application framework. Anais. OOPSLA 1986, p. 186-201, SIGPLAN Notices 21(11).

[10]RUSSO, V. An object-oriented operating system. Tese de Doutorado, University of Illinois at Champaign-Urbana, 1990.

[11]SCHMID, H. A. Creating the architecture of a manufacturing framework by design patterns. Anais. OOPSLA 1985, p. 370-384, Austin-TX, 1995.

Atualmente há frameworks largamente utilizados, como o OLE (*Object Linking and Embedding*), o OpenDoc e o DSOM (*Distributed System Object Model*). O surgimento da linguagem Java permitiu o desenvolvimento de novos frameworks como o AWT (*Abstract Windows Toolkit*), que faz parte do JFC (*Java Foundation Classes*) e Java Beans. A maioria dos frameworks disponíveis comercialmente oferece suporte para domínios técnicos (como interface com o usuário e distribuição, por exemplo) e a maioria dos frameworks de domínio específico é proprietária (Fayad et al., 1999).

Do ponto de vista da estrutura de um framework, pode-se defini-lo como um conjunto de classes que contém o projeto abstrato de soluções para uma família de problemas relacionados, propiciando o reuso com granularidade maior do que classes (Johnson e Foote, 1988). O reuso de código proporcionado pelos frameworks não é tão importante quanto o reuso das interfaces internas de um sistema e da maneira como as funções são divididas entre seus componentes (Johnson e Russo, 1991). Esse projeto de alto nível representa o conteúdo intelectual do software, que é muito mais difícil de criar ou recriar do que o código.

Ainda do ponto de vista da estrutura, um framework é composto por uma coleção de classes abstratas e concretas e a interface entre elas, representando o projeto de um subsistema (Pree, 1995). A base do framework é fornecida pelas classes abstratas, a partir das quais classes concretas ou outras classes abstratas podem ser implementadas. Quando essa coleção de classes abrange um sistema de software genérico para um domínio de aplicação, utiliza-se o termo "framework de aplicação".

Do ponto de vista do propósito do framework, pode-se defini-lo como o esqueleto de uma aplicação, que pode ser instanciado por um desenvolvedor de aplicações (Johnson, 1997b). Trata-se de uma aplicação semi-completa reutilizável que, quando especializada, produz aplicações personalizadas (Johnson e Foote, 1988). Pode ainda ser visto como um conjunto de blocos de software pré-fabricados que programadores podem usar, estender e adaptar para soluções computacionais específicas (Taligent, 1993). Frameworks fazem com que desenvolvedores não precisem começar do nada sempre que constróem uma nova aplicação. Assim, frameworks podem ser considerados uma técnica de reuso orientado a objetos (Johnson, 1997b), que compartilha muitas características de reuso de outras técnicas, como os *templates* (Volpano, 1989 *apud* (Johnson, 1997b))[12] e os *esquemas* (Lubars, 1987 *apud* (Johnson, 1997b))[13].

Uma das características mais importantes de um framework é a "inversão de controle" (Johnson, 1997a). No reuso tradicional de componentes ou bibliotecas, desenvolve-se a aplicação específica com chamadas aos componentes que se deseja reutilizar. Assim, a responsabilidade pelas

[12]VOLPANO, D.M.; KIEBURTZ, R.B. The templates approach to software reuse. In Biggerstaff, T. J.; Perlis, A.J. (eds), Software Reusability, vol 1, ACM Press, 1989.

[13]LUBARS, M.D.; HARANDI, M.T. Knowledge-based software design using design schemas. In Proceedings of the 9th International Conference on Software Engineering (março de 1987), p. 253-262

chamadas e pelo fluxo de controle do programa é do desenvolvedor de aplicações. Já em um framework o programa principal é reusado, ficando a cargo do desenvolvedor de aplicações a decisão de quais componentes chamar e até mesmo de criar novos componentes para serem chamados a partir do código do framework. Assim, o framework determina o fluxo de controle do programa e sua estrutura geral.

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

Uma outra denominação semelhante aos pontos variáveis é a de gancho (em inglês, *hook*), que consiste de um ponto do framework passível de ser adaptado de alguma forma, por exemplo, por meio do preenchimento de parâmetros ou da criação de subclasses (Froehlich et al., 1997). A descrição de cada gancho documenta um problema ou requisito do framework e oferece diretrizes de como utilizar o gancho para satisfazer tal requisito.

Os frameworks podem ser classificados de acordo com seu escopo, ou seja, a abrangência de sua aplicação em um dado domínio. Podem também ser classificados de acordo com a forma de reuso do framework para gerar novas aplicações. Considerando o escopo de aplicação, frameworks são classificados em três grupos (Fayad et al., 1999; Fayad e Schmidt, 1997): frameworks de infra-estrutura do sistema, frameworks de integração *middleware* e frameworks de aplicação empresarial.

Os frameworks de infra-estrutura do sistema simplificam o desenvolvimento da infra-estrutura de sistemas portáveis e eficientes, como por exemplo os sistemas operacionais, sistemas de comunicação, interfaces com o usuário e ferramentas de processamento de linguagem. Em geral são usados internamente em uma organização de software e não são vendidos a clientes diretamente.

Os frameworks de integração *middleware* são usados, em geral, para integrar aplicações e componentes distribuídos. Eles são projetados para melhorar a habilidade de desenvolvedores em modularizar, reutilizar e estender sua infra-estrutura de software para funcionar de forma integrada em um ambiente distribuído. Exemplos dessa classe de framework são o *Object Request Broker* (ORB), *middleware* orientado a mensagens e bases de dados transacionais.

Os frameworks de aplicação empresarial estão voltados a domínios de aplicação mais amplos e são muito úteis para atividades de negócios das empresas, como por exemplo sistemas de telecomunicações, aviação, manufatura e engenharia financeira. Frameworks dessa classe são mais

caros para desenvolver ou comprar, mas podem dar um retorno substancial do investimento, já que permitem o desenvolvimento direto de aplicações e produtos.

Existe um relacionamento entre essas três classes de frameworks (Yassin e Fayad, 2000): os frameworks de integração *middleware* incluem os frameworks de infra-estrutura e os frameworks de aplicação empresarial incluem os outros dois tipos. Neste trabalho é proposta a construção de um framework de aplicação empresarial.

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

Segundo um levantamento realizado por Yassin e Fayad (2000), a tendência atual é pelo desenvolvimento de frameworks caixa cinza. Apesar dos frameworks caixa branca serem mais fáceis de desenvolver, a exposição do código aos desenvolvedores pode causar problemas. Já os frameworks caixa preta são mais abstratos e menos flexíveis, tornando difícil a manutenção das aplicações dele derivadas. Frameworks caixa cinza bem projetados superam as barreiras impostas pelos outros dois tipos, pois possuem flexibilidade e facilidade de extensão, sem expor desnecessariamente informações que não interessam ao desenvolvedor de aplicações.

Em suma, um framework caixa branca é mais fácil de projetar, pois não há necessidade de prever todas as alternativas de implementação possíveis, o que ocorre no framework caixa preta. No entanto, o framework caixa preta é mais fácil de usar, pois basta escolher a implementação desejada, ao invés de fornecer a implementação completa, como é o caso do caixa branca. Frameworks caixa cinza tentam tirar proveito das vantagens dos outros dois tipos, sendo portanto um meio termo. Frameworks caixa branca podem evoluir para se tornar cada vez mais caixa preta (Roberts e Johnson, 1998). Isso pode ser feito de forma gradativa, implementando-se várias alternativas que depois são aproveitadas na instanciação do framework. Neste trabalho, o framework a ser construído é do tipo caixa branca, embora com tendência a evoluir para caixa cinza e depois caixa preta à medida que novas aplicações forem sendo desenvolvidas.

Os conceitos de “método-gancho” e “método-gabarito” (em inglês, *hook method* e *template method*, respectivamente), são importantes para entender como funciona o reuso de frameworks

caixa-branca. Um método-gancho é definido em uma classe abstrata do framework e sua implementação é fornecida pelas sub-classes concretas[14]. As classes abstratas de um framework caixa branca possuem métodos-gabarito com chamadas para um ou mais métodos-gancho. Os métodos gabarito não precisam ser sobrepostos pelas classes concretas da aplicação, visto que sua lógica não muda. Gamma et al. (1995) definiram o padrão *Template Method* justamente com o propósito de permitir essa flexibilidade em frameworks ou em software orientado a objetos em geral.

A presença de frameworks influencia o processo de desenvolvimento de software. Por exemplo, desenvolver um sistema financeiro tendo em mãos um framework para construção de aplicações financeiras é diferente de desenvolver o mesmo sistema partindo do nada. O desenvolvimento de software baseado em frameworks possui três fases (Bosch et al., 1999): a fase de desenvolvimento, na qual o framework é construído e testado; a fase de uso, na qual o framework é instanciado inúmeras vezes para aplicações específicas; e a fase de evolução e manutenção, na qual o framework evolui para incorporar novas funcionalidades do domínio. A fase de desenvolvimento de um framework é tratada na seção 2.6. A fase de uso do framework é apresentada na seção 2.7. A fase de evolução e manutenção do framework, que não está incluída no escopo desta tese, deve-se ao fato de que podem ser identificados novos requisitos funcionais no domínio de aplicação; pode haver evolução tanto de hardware quanto de software; podem ser relatados erros em aplicações derivadas do framework, etc. Porém, a manutenção de um framework não é tão simples quanto possa parecer a princípio, já que as aplicações derivadas do framework precisarão sofrer a mesma manutenção para continuaram compatíveis com o framework, ou então o framework deverá ser dividido em dois frameworks separados, o que pode acarretar futuros problemas de manutenção (Bosch et al., 1999). Deve-se notar que nem sempre é viável corrigir todas as aplicações derivadas de um framework, devido ao alto custo e, muitas vezes, ao grande (ou até mesmo ignorado) número de aplicações existentes. Existe, também, um alto risco de introduzir novos erros nessas aplicações.

2.2.3 Outras formas de reuso

Existem outras formas de reuso além de frameworks e padrões, como por exemplo componentes de software, geradores de aplicações e linhas de produto. Porém, elas não são aqui abordadas em detalhes por ser o foco específico desta tese relacionado a frameworks e padrões. Ainda assim, apresentam-se a seguir as definições desses termos e seu relacionamento (caso seja apropriado) com padrões e frameworks.

[14] é possível fornecer uma implementação *default* para o método-gancho, mas ela pode ser sobreposta pelas classes concretas

Segundo definição proposta por Werner e Braga (2000), feita com base em definições de Sametinger (1997 apud Werner & Braga, 2000)[15] e Krutchen (1998 apud Werner & Braga, 2000)[16], "componentes reutilizáveis são artefatos auto-contidos, claramente identificáveis, que descrevem e realizam uma função específica e têm interfaces claras, em conformidade com um dado modelo de arquitetura de software, documentação apropriada e um grau de reutilização definido". A distinção entre frameworks e componentes é feita principalmente considerando-se o fluxo de controle do programa, conforme visto na seção 2.2.2. Quando utiliza um componente, o desenvolvedor é responsável por sua chamada, ou seja, pelo fluxo de controle do programa, enquanto em um framework há a inversão de papéis: o programa principal é reutilizado e o desenvolvedor decide quais componentes são chamados e devem, portanto, ser derivados (Johnson e Foote, 1988). Em relação a padrões, é possível implementar um ou mais componentes com base em um padrão, já que o padrão possui um problema a ser resolvido sob determinadas influências, o que seria traduzido, em termos de componente, para a funcionalidade específica do componente e sua interface.

Geradores de aplicação são ferramentas de apoio por computador ao desenvolvimento de software. Além de reduzir os custos e aumentar a produtividade, eles ajudam a melhorar a qualidade dos sistemas produzidos. Essencialmente, um gerador de aplicação é um utilitário que, a partir de uma especificação de alto nível de um problema implementável, transforma automaticamente essa especificação na implementação do problema (Czarnecki e Eisenecker, 2000). Frameworks diferem de geradores de aplicação principalmente quanto ao código final da aplicação específica gerada. Em um framework, grande parte de seu código é incorporado à nova aplicação, enquanto em um gerador de aplicações é gerado todo o código da nova aplicação, com base em uma especificação do sistema em alto nível fornecida pelo usuário. Por outro lado, grande parte do código do gerador de aplicação existe apenas para fornecer mecanismos de geração do código do produto final. O *wizard* proposto nesta tese é um gerador de aplicações, pois ele gera classes e métodos a partir da especificação de um sistema particular, nesse caso documentada por meio dos padrões utilizados para modelar esse sistema.

O conceito de linhas de produto de software (em inglês *software product-lines*) (Gimenes e Travassos, 2002; Weiss e Lai, 1999) baseia-se em outros setores da indústria, como a aeroespacial, automobilística e de componentes eletrônicos, visando à criação de um conjunto de produtos de software com características similares, por meio da definição de uma infra-estrutura comum aos itens que compõem os produtos e da parametrização das diferenças entre eles. A abordagem de linha de produtos pode ser considerada complementar a frameworks e engenharia de domínio, fornecendo uma forma de organizar essas técnicas. Um processo bem definido é apresentado para

[15]SAMETINGER, J. Software Engineering with Reusable Components, Springer, 1997.

[16]KRUTCHEN, Modeling Component Systems with the Unified Modeling Language, International Workshop on Component-Based Software Engineering, 1998.

que a produção de software pertencente a uma família de produtos seja feita em larga escala, como numa linha de produção industrial. Assim, frameworks podem ser usados como uma das tecnologias que permitem a adaptação do produto final de acordo com as características específicas desejadas. Da mesma forma, o enfoque de linha de produtos encontra, nos padrões, indicações de como construir tanto a arquitetura quanto os componentes dessa linha de produtos (Gimenes e Travassos, 2002).

2.2.4 Comparações entre formas de reuso

Padrões X Frameworks

Padrões são mais abstratos do que frameworks, porque um framework inclui código executável, enquanto um padrão representa conhecimento e experiência sobre software (pode-se dizer que frameworks têm natureza física, enquanto padrões têm natureza lógica) (Johnson e Russo, 1991). Padrões são menores do que frameworks. Em geral, padrões são compostos por duas ou três classes, enquanto os frameworks envolvem um número bem maior de classes, podendo englobar diversos padrões (Gamma et al., 1995). Padrões são menos especializados do que frameworks, já que os frameworks geralmente são desenvolvidos para um domínio de aplicação específico e padrões (de projeto, principalmente) podem ser usados em diversos tipos de aplicação. Apesar de haver padrões mais especializados, eles não são capazes de ditar a arquitetura de uma aplicação, ao contrário dos frameworks.

Frameworks X Linguagens de Padrões

Padrões e linguagens de padrões podem ser utilizados para documentar um framework (Johnson, 1992), com o objetivo principal de facilitar seu reuso. Pode-se criar um conjunto de padrões relacionados que mostrem como utilizar o framework, sem mostrar como ele funciona ou como foi projetado. Porém, se ao invés de documentar um framework com padrões individuais for utilizada uma linguagem de padrões, é possível organizar a seqüência de decisões que geraram o projeto do framework (Brugali et al., 2000). Essa organização precisa ser entendida, dentro do contexto apropriado, para que o desenvolvedor possa tomar novas decisões na manutenção do framework. Em outras palavras, a linguagem de padrões torna mais fácil ao desenvolvedor alcançar o nível de entendimento necessário para estender o framework.

Um framework permite a implementação de uma aplicação projetada segundo a linguagem de padrões (Brugali, 1998; Brugali et al., 2000), ou seja, ele contém a implementação de todos os padrões que compõem a aplicação específica. Ao mesmo tempo, uma linguagem de padrões oferece as regras para uso dos elementos do framework e para sua extensão. Assim, olhando-

se para a linguagem de padrões pode-se saber o que encontrar no framework e o que facilita a manutenção do mesmo.

Segundo Brugali e Menga (1999), existe um inter-relacionamento estrito entre linguagens de padrões e frameworks. Primeiramente, ambos são concebidos para um domínio específico e resolvem a maioria dos problemas comuns a aplicações de tal domínio. Em segundo lugar, linguagens de padrões geram frameworks, pois elas contêm as principais abstrações encontradas no domínio de aplicação, as quais dão origem aos componentes de alto nível do framework. Em terceiro lugar, o framework dá suporte à linguagem de padrões, pois pode-se considerar a linguagem de padrões como um método que guia a transformação do framework em uma aplicação concreta. Segundo os autores, esse método não está em conflito com métodos conhecidos de desenvolvimento — como Booch et al. (1998), por exemplo — mas complementa-os, oferecendo diretrizes concretas e específicas de domínio. Assim, a disponibilidade de uma linguagem de padrões e do respectivo framework para um domínio específico, faz com que novas aplicações não precisem ser desenvolvidas a partir do zero, já que o framework fornece as implementações reutilizáveis de cada padrão da linguagem. O processo de desenvolvimento, portanto, pode seguir o grafo da linguagem, começando pela raiz em direção às folhas, decidindo pela aplicação de cada padrão específico e reusando a implementação correspondente fornecida pelo framework. Em quarto e último lugar, as linguagens de padrões são uma forma efetiva de documentação do framework, já que podem documentar diversos aspectos, tais como: o domínio de aplicação ao qual o framework se refere (essencial para que o usuário possa avaliar a adequação do framework às suas necessidades); o projeto do framework em termos de objetos e seus relacionamentos (útil na manutenção e extensão do framework); e a especificação e implementação das classes do framework (necessários para que aplicações concretas possam ser implementadas a partir do framework).

Linguagens de Padrões X Frameworks X Aplicações

A Figura 2.2 resume as idéias descritas anteriormente, ilustrando o relacionamento entre linguagens de padrões, frameworks e aplicações. Todos eles (frameworks, linguagens de padrões e aplicações) encontram-se em um domínio específico. A linguagem de padrões permite documentar e projetar o framework. Também ajuda na instanciação de aplicações, que pode ser feita usando o framework. O framework permite implementar aplicações e dá suporte a linguagens de padrões, implementando os padrões nela contidos. Embora muitos desses relacionamentos tenham sido notados pelos autores mencionados acima, nenhum estudo foi feito para mostrar como pode-se tirar proveito desses relacionamentos.

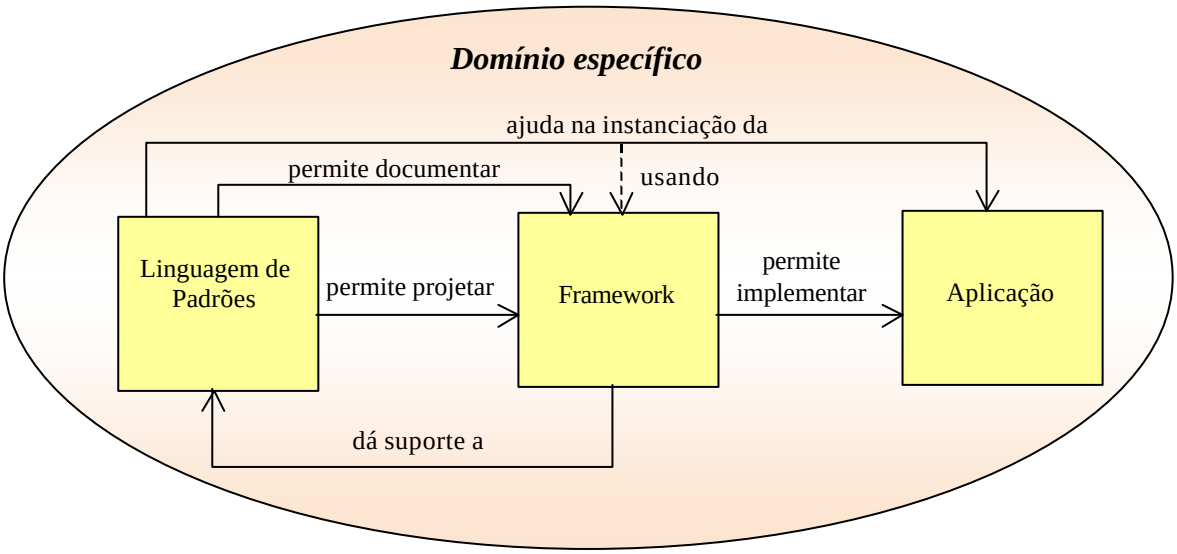


Figura 2.2: Relacionamento entre Linguagens de Padrões, Frameworks e Aplicações

2.3 Padrões de Análise e Linguagens de Padrões Relevantes

Dezenas de padrões de análise têm sido propostos nos últimos anos, sendo que os primeiros a surgir foram os de Coad (1992), úteis principalmente para o desenvolvimento de sistemas de informação. Alguns anos depois, Coad reuniu esses padrões a outros vinte e quatro, publicando-os em um livro que contém, além dos padrões, inúmeras estratégias a serem utilizadas no desenvolvimento de sistemas desse domínio (Coad et al., 1997). Cada estratégia é um plano de ação objetivando o cumprimento de um objetivo específico. Como exemplo de estratégia, pode-se citar a estratégia "Serviços Básicos". Ela sugere que se evite mostrar, no modelo de objetos, os serviços básicos da classe, como por exemplo: atribuir, recuperar, adicionar, remover, criar, inicializar e remover o objeto. Cada padrão de Coad é um gabarito (do inglês *template*) a ser seguido durante a construção do modelo de análise do sistema. Os trinta e um padrões são agrupados em cinco famílias de padrões: padrões fundamentais, padrões de transação, padrões agregados, padrões de planejamento e padrões de interação. Coad define ainda padrão de modelo de objetos como um gabarito de objetos com responsabilidades e interações típicas, de modo que esse gabarito pode ser aplicado inúmeras vezes, por analogia. Embora em seu primeiro artigo, Coad utilize um formato pré-definido para descrever seus padrões, no livro ele apresenta os padrões apenas consistindo de um nome, um diagrama com a solução proposta e exemplos em forma textual, não havendo explicação sobre os componentes do padrão. A forma de utilização dos padrões e das estratégias é ilustrada por

meio de exemplos. No total são cinco exemplos de sistemas de negócio, nos quais, a partir da coleta da especificação, deriva-se o modelo de análise utilizando estratégias e padrões. Como exemplo de um padrão de Coad, pode-se citar o "Estado através de uma coleção" (do inglês *State across a collection*) que resolve um problema comum em sistemas orientados a objetos, o de agregações, no qual uma super-classe possui atributos que são compartilhados pelas subclasses. Outro padrão proposto por Coad e posteriormente reescrito (Coad et al., 1997; Johnson e Woolf, 1998)[17] é o "Item-Descrição" (do inglês *Item-Description*). Contudo, a versão proposta por Johnson e Woolf encontra-se especificada em maior nível de detalhe e utilizando o formato GoF (ver seção 2.2.1), o que é importante para facilitar a aplicação do padrão. O problema resolvido por esse padrão é o de uma classe com diversos tipos, que podem variar em tempo de execução. A solução é agrupar os atributos comuns em uma classe chamada "Classe" e criar uma outra classe denominada "Classe Tipo" para conter os tipos. Os objetos da classe "Classe" possuem uma referência a algum dos objetos da classe "Classe Tipo".

Outros dois trabalhos co-relacionados e de importância na área de padrões de análise são os de Hay (1996) e de Fowler (1997). Ambos foram escritos na mesma época e apresentam modelos lógicos de dados que são utilizados em inúmeros sistemas de negócios, tendo sido derivados a partir da experiência prática dos autores. Hay apresenta dezenas de modelos entidade-relacionamento para solucionar problemas na área de negócios e entidades governamentais. Entre eles pode-se enumerar diversos modelos que pertencem especificamente à gestão de recursos de negócios, como por exemplo: ordens de serviço, produtos, compra e venda. Fowler apresenta setenta e seis padrões que, segundo o autor, "refletem estruturas conceituais de processos de negócios, ao invés de implementações reais de software". Os padrões estão divididos em grupos, de acordo com os domínios de negócio. Dentre os grupos relacionados ao domínio de gestão de recursos de negócios estão o grupo "Responsabilidade" (do inglês *Accountability*), o grupo "Planejamento", para o planejamento e uso de recursos e o grupo "Comercialização", para lidar com a compra e venda de bens, com ênfase na mudança de valor desses bens de acordo com as condições de mercado.

Tanto Hay quanto Fowler utilizam formatos próprios para documentação dos padrões. Fowler afirma que um formato fixo traz desvantagens, pois um par "problema-solução" nem sempre é adequado a todo padrão. Ele usa um texto para descrever o contexto de aplicação do padrão, em geral mostrando exemplos práticos e soluções na forma de diagramas. Hay não chega a justificar o formato utilizado em termos dos formatos usualmente seguidos pela comunidade de padrões. Ele documenta os modelos por meio de uma descrição do contexto de utilização, mostra o diagrama entidade-relacionamento da solução, descreve os participantes, o raciocínio que levou à solução e indica os padrões relacionados.

[17] Johnson e Woolf (1998) o definiram sob o nome de "Objeto-Tipo" (do inglês *Type-Object*) e o próprio autor o re-definiu como "Item-Item Específico" (Coad et al., 1997)

Outros padrões de análise que podem ser citados são os de Boyd (1998) e os de Fernandez (Fernandez, 2000; Fernandez e Yuan, 1999; Fernandez et al., 2000). Boyd descreve padrões para representar a associação entre objetos em sistemas de negócios. São eles: o padrão "Objeto-Associação" (do inglês *Association-Object*), o padrão "Contato com o Cliente" (do inglês *Customer Contact*) e o padrão "Pedido em três níveis" (do inglês *3-Level Order*). Eles são comumente encontrados em sistemas de informação, já que representam algo que acontece em um ponto específico do tempo associando dois ou mais objetos.

Fernandez e outros autores propõem diversos padrões de análise para sistemas no domínio de negócios. O padrão de análise "Reserva e Uso de Entidades Reusáveis" (Fernandez e Yuan, 1999) descreve a realização de uma reserva para uma entidade reusável e sua subsequente utilização. Tal padrão possui diversas semelhanças com o padrão "Resource Location", apresentado pela autora desta tese anteriormente (Braga et al., 1998). Também é equivalente à aplicação dos padrões "Locar o Recurso" e "Reservar o Recurso" da linguagem de padrões para Gestão de Recursos de Negócios, apresentada no Capítulo 3. O termo "Entidade", utilizado por Fernandez e Yan, equivale ao termo "Recurso" utilizado pela autora em seus trabalhos (Braga et al., 1998, 1999).

Os padrões "Pedido" (do inglês *Order*) e "Expedição" (do inglês *Shipment*) (Fernandez et al., 2000) descrevem como o cliente efetua um pedido de um produto e como é feito seu subsequente despacho. Esses padrões são aplicações específicas do padrão "Resource Trade", também apresentado pela autora desta tese anteriormente (Braga et al., 1998) e equivalem à aplicação dos padrões "Comercializar o Recurso" e "Itemizar a Transação do Recurso" da linguagem de padrões para Gestão de Recursos de Negócios apresentada no Capítulo 3. Cabe observar que todos esses padrões já são instanciações particulares de padrões de negócios mais genéricos propostos por Coad (1992) e Boyd (1998). Essas instanciações visam especializar os padrões para um nível de abstração mais baixo, sem contudo chegar a um domínio específico. O padrão "Gerenciador de Estoque" (do inglês *Stock Manager*) (Fernandez, 2000) é um padrão de análise para sistemas de inventário, que controla a quantidade e localização de itens em estoque, atualizando essas quantidades de acordo com os diferentes estágios de manufatura ou produção, desde o pedido de componentes até o despacho do produto. Esse padrão engloba alguns outros padrões existentes, como o Objeto-Tipo (Johnson e Woolf, 1998) e o Recipiente-Contexto (Coad et al., 1997).

Embora inúmeras linguagens de padrões tenham surgido na última década, poucas são as que podem ser aplicadas a sistemas de negócios, entre as quais pode-se citar: a linguagem de padrões "Checagem de integridade da Informação" (Cunningham, 1995), que fornece diretrizes sobre como fazer checagem nas entradas de dados para separar dados inválidos de dados válidos e assegurar que o menor número possível de dados inválidos seja registrado; a linguagem de padrões "Atravessando Abismos" (Brown e Whitenack, 1996), que auxilia na construção de aplicações orientadas a objetos que utilizam persistência em base de dados relacional; e a linguagem de padrões

"Transações e Contas" (Johnson, 1996), que descreve o modo de funcionamento da maioria dos sistemas de processamento de transações de negócios e fornece diretrizes para melhorá-los.

2.4 Frameworks Relevantes na área de Negócios

Nesta seção, faz-se um resumo de alguns dos frameworks relevantes encontrados na literatura, cujo domínio é relacionado a negócios. Sem dúvida, o principal deles é o IBM SanFrancisco, internacionalmente conhecido e utilizado em diversas companhias no mundo todo (entre eles Japão, Canadá, Itália, Brasil, México, Polônia e Holanda). Outros frameworks aqui abordados são o *Accounts*, que é um framework de aplicação empresarial, e o *OmniBuilder*, que é um framework de integração de *middleware*.

2.4.1 IBM SanFrancisco

O framework de componentes SanFrancisco (em inglês *SanFrancisco Component Framework*), ou SFCF, foi desenvolvido por uma equipe da IBM utilizando a linguagem Java, para auxiliar seus parceiros de negócios na construção de aplicações para uso próprio ou para distribuição para seus clientes (Monday et al., 2000). Ele pode ser alugado por empresas para a criação de aplicações que, depois de encerrado o período de desenvolvimento, passam a utilizar o sistema gerado sem outros encargos (Yassin e Fayad, 2000). A Figura 2.3 mostra a arquitetura do SanFrancisco, que é composto por diversas camadas que permitem seu reuso em diferentes níveis (Monday et al., 2000).

A camada mais específica do SanFrancisco é denominada "Processos de Negócio Centrais" e trata dos principais processos e classes de um domínio particular. Por ser específica, essa camada aplica-se apenas aos domínios por ela cobertos, que são: Gestão de Armazéns, Livro Razão, Contas a Pagar, Contas a Receber e Gestão de Pedidos.

A camada intermediária do SanFrancisco, denominada "Objetos de Negócio Comuns", consiste de classes, processos e mecanismos comuns a vários domínios de aplicação, divididos em três grupos principais: objetos gerais de negócio (por exemplo, Companhia, Moeda e Cliente); objetos financeiros de negócio (por exemplo, Conta Bancária e Calendário Financeiro); e mecanismos generalizados ou padrões (por exemplo, Política dirigida à cadeia de responsabilidade e Controlador). Nessa camada intermediária do SanFrancisco são fornecidas partes independentes que podem ser escolhidas e utilizadas de maneira similar a uma biblioteca de componentes, desde que seja respeitado o relacionamento entre processos e classes do framework SanFrancisco.

A camada inferior do SanFrancisco, denominada "Fundação e Utilitários", fornece os serviços básicos necessários a uma aplicação de negócio, isolando-a, dessa forma, da tecnologia subjacente.

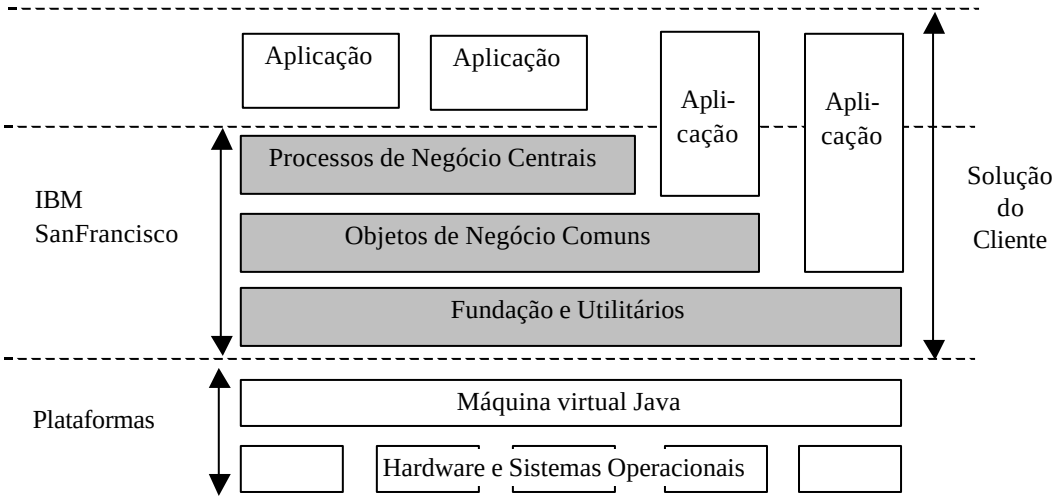


Figura 2.3: Arquitetura do framework IBM SanFrancisco (Monday et al., 2000)

Exemplos de serviços incluídos nessa camada são armazenamento e compartilhamento de objetos de negócio e utilitários para configuração e controle de serviços básicos.

Cada uma das camadas do SanFrancisco é construída com base nas camadas abaixo dela. Assim, a camada "Processos de Negócio Centrais" está acima da camada "Objetos de Negócio Comuns". Uma classe fornecida pela camada "Processos de Negócio Centrais" é construída usando uma ou mais classes da camada "Objetos de Negócio Comuns" ou da camada "Fundação e Utilitários". O usuário do SanFrancisco pode utilizar a camada mais apropriada às suas necessidades. Por exemplo, uma aplicação financeira pode ser construída sobre as partes Livro Razão, Contas a Pagar e Contas a Receber da camada "Processos de Negócio Centrais". Uma aplicação para seguradoras deveria ser construída sobre a camada "Objetos de Negócio Comuns", já que o SanFrancisco não possui uma parte específica para sistema de seguradoras em sua camada "Processos de Negócio Centrais". Uma aplicação que não seja da área de negócio deve ser construída sobre a camada "Fundação e Utilitários".

2.4.2 OmniBuilder

O OmniBuilder (OMNISPHERE, 2002) é um ambiente de desenvolvimento totalmente orientado a objetos, feito para desenvolver aplicações completas a partir de um modelo de negócios (*Business Model* em inglês). Segundo Yassin e Fayad (2000), ele é um framework de integração de *middleware*, pois consiste de um gerador de aplicações abertas capaz de construir aplicações de negócios totalmente funcionais e sofisticadas, a partir do modelo de negócio. O OmniBuilder gerencia o ciclo de vida completo da aplicação, captando o modelo de objetos, incluindo regras de

negócios, requisitos e casos de uso. Gera todos os componentes da aplicação, desde os objetos servidores, módulos cliente, protocolos de *middleware*, documentação e ajuda online. A arquitetura do OmniBuilder garante independência da linguagem alvo (por exemplo, Java, C++) e da estrutura de implementação (por exemplo, multi-camadas, Intranet). Pode ser personalizada e estendida pelo usuário para obedecer a requisitos específicos e foi projetada para acomodar novas tecnologias que venham a surgir.

A Figura 2.4 mostra a estrutura do OmniBuilder. Ele gera aplicações completas a partir do Modelo de Negócios, que pode ser fornecido manualmente durante a análise ou pode ser criado utilizando os Gabaritos de Negócios (*Business Templates*). Esse gabaritos podem conter um modelo completo ou parcial do negócio, sendo importados para o OmniBuilder e então melhorados ou modificados conforme desejado, para criar um modelo de negócios personalizado e completo. O Modelo da Aplicação é construído automaticamente a partir do modelo de negócios. Ele contém elementos para prototipação dos objetos e de seu comportamento. Padrões de Projeto são usados pelo OmniBuilder para captar comportamento complexo e para promover reuso de componentes. Muitos padrões de projeto já se encontram incorporados ao OmniBuilder, sendo oferecida a facilidade de adicionar padrões de projeto personalizados, que podem ser utilizados para gerar características particulares de maneira reusável. O Modelo de Implementação contém os objetos executáveis gerados. Podem ser produzidas aplicações com interface com o usuário baseada em Windows ou páginas dinâmicas baseadas em HTML para Internet/Intranet a serem executadas por um Browser Internet. O OmniBuilder pode gerar aplicações em um número de tecnologias alvo, incluindo Java e C++. Pode também gerar aplicações de três camadas compatíveis com CORBA/COM/OLE.

2.4.3 Accounts

O *Accounts* é um framework para criação de sistemas de processamento de transações de negócio, no qual definem-se contas e transações (Keefer, 1994). Foi desenvolvido na Universidade de Illinois-EUA usando a linguagem Smalltalk-80. As duas classes principais do *Accounts* são conta e transação. Uma conta controla as transações nela postadas e é capaz de calcular os valores de seus atributos em uma certa data. Exemplos de contas são: diários de inventário, contas bancárias e registros de empregados. Transação, um termo comumente usado em sistemas contábeis, consiste no registro de um evento significativo envolvendo dados relevantes ao sistema. Exemplos de transações são compra, venda, depósito e retirada.

Um sistema contábil representa a visão do mundo por parte de uma pessoa ou entidade de negócio e é tipicamente utilizado para representar o fluxo de caixa da organização. Sistemas contábeis podem controlar diferentes tipos de dados, inclusive não-financeiros, como suprimento em inventários, dados de censo ou dados meteorológicos. Em qualquer sistema contábil é possível

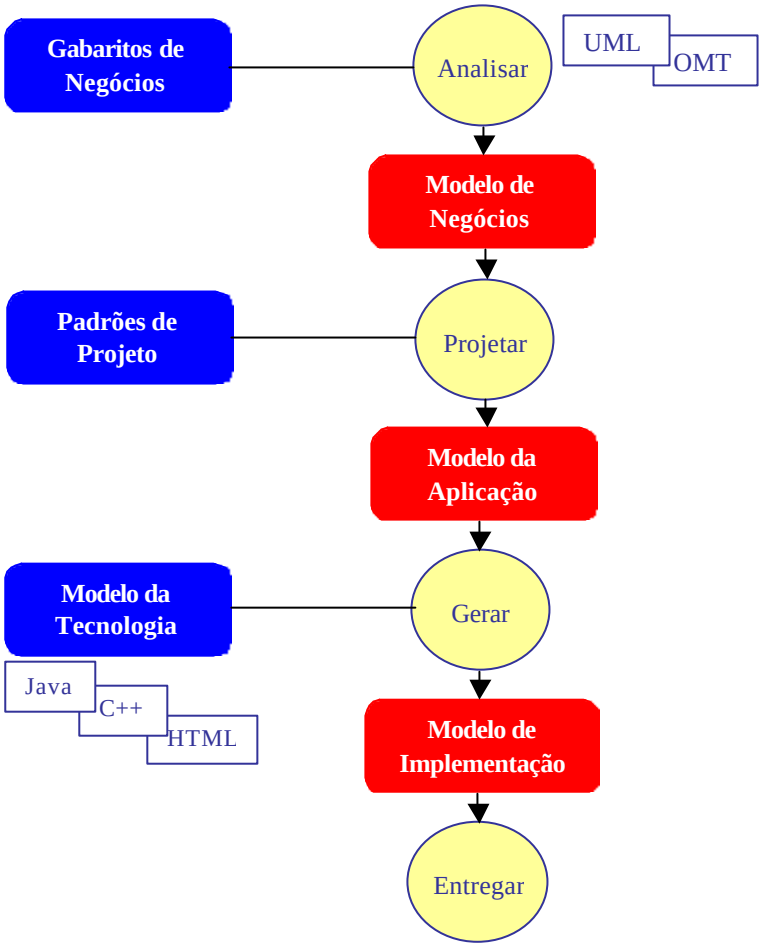


Figura 2.4: Estrutura do OmniBuilder (OMNISPHERE, 2002)

controlar mudanças causadas em categorias de objetos ao longo do tempo, o que pode ser representado utilizando contas e transações. Algumas características importantes em sistemas contábeis e providas pelo framework *Accounts* são: prover um meio de acesso às transações de acordo com sua data, criar grupos de contas de acordo com uma hierarquia e armazenar os dados em um banco de dados.

2.5 Trabalhos relevantes que relacionam padrões a frameworks

2.5.1 *HotDraw*

Os primeiros frameworks começaram a surgir a partir de 1986, praticamente na mesma época das linguagens de padrões. Quem primeiro notou o relacionamento entre linguagens de padrões e

frameworks foi Johnson (1992), em um artigo no qual sugere a utilização de padrões para documentar frameworks e, desta forma, facilitar seu reuso. Ele apresenta um conjunto de dez padrões para documentar o framework *HotDraw*, que é um framework para editores gráficos semânticos desenvolvido originalmente na empresa "Tektronix", por Kent Beck e Ward Cunningham, e reimplementado diversas vezes desde então[18].

O padrões que documentam o *HotDraw* foram criados após o framework, portanto não foram utilizados no seu desenvolvimento. Seu propósito é de ensinar a utilizar o framework e não de mostrar como ele funciona, embora muitas vezes os padrões contenham informações sobre a teoria envolvida no projeto do framework, útil para melhor entendê-lo.

Johnson sugere que o primeiro padrão do conjunto de padrões descreva o propósito do framework e que os padrões sejam tratados como um grafo dirigido, no qual as informações sobre o projeto do framework fiquem o mais longe possível do primeiro padrão. Cada padrão deve conter indicações dos próximos padrões a aplicar. Além disso, Johnson ressalta a necessidade de colocar exemplos ilustrativos em cada padrão, que tornam o framework mais concreto, do ponto de vista dos usuários, e fazem com que seu fluxo de controle seja melhor entendido.

Johnson optou pelo uso do termo "padrões" ao invés de "linguagens de padrões", devido à forte conotação matemática envolvida no termo "linguagem", que implica formalismos (como linguagens livres de contexto, por exemplo). Posteriormente, o termo "linguagem de padrões" foi melhor definido pela comunidade de padrões e outros trabalhos usando essa denominação foram publicados a partir da primeira conferência sobre padrões, em 1994, nos Estados Unidos.

Outro trabalho que trata do relacionamento entre o framework *HotDraw* e padrões é o artigo de Beck e Johnson (1994), no qual são mostrados os padrões de projeto contidos no *HotDraw* e que, portanto, servem para gerar sua arquitetura. O objetivo desses padrões é documentar o projeto do framework para facilitar sua compreensão, seja para fins de manutenção ou para permitir sua instanciação para aplicações específicas.

2.5.2 G++

O framework G++ (Brugali et al., 2000), construído com base na linguagem de padrões de mesmo nome (Aarsten et al., 2000), foi o único caso encontrado na literatura, até o momento da escrita desta tese, de um framework construído com base em uma linguagem de padrões.

O G++ é um framework para desenvolvimento de sistemas integrados de fabricação, fornecendo classes que implementam blocos construtivos a serem usados no desenvolvimento de novas aplicações. Foi construído com base na linguagem de padrões G++, que possui onze padrões estruturados por meio de uma árvore, conforme mostra a Figura 2.5.

[18] mais informações sobre o *HotDraw* podem ser obtidas na URL: `http://st-www.cs.uiuc.edu/users/brant/HotDraw/HotDraw.html`

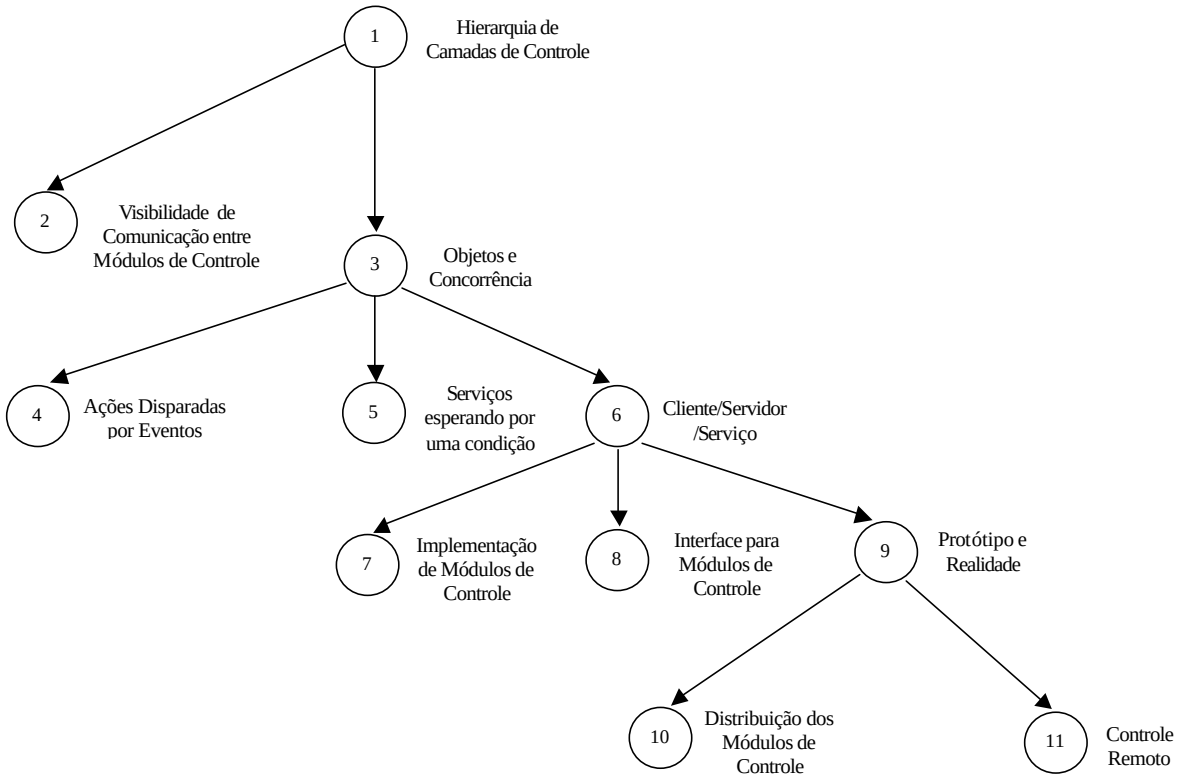


Figura 2.5: Linguagem de Padrões G++ (Aarsten et al., 2000)

Os padrões da G++ situam-se em diversos níveis de abstração: os dois primeiros padrões da Figura 2.5 referem-se à análise e projeto de alto nível da aplicação desenvolvida; os padrões de três a seis tratam do projeto lógico; os padrões de sete a nove cuidam da transição do protótipo da aplicação para sua implementação final; e os padrões dez e onze apresentam dois casos de distribuição física.

O framework G++ fornece mecanismos, classes abstratas e blocos construtivos para a maioria das soluções propostas pela linguagem de padrões G++ (Brugali et al., 1997). Os componentes do framework são classificados em três tipos: elementares, projeto básico e dependentes do domínio. Esses tipos refletem diferentes necessidades do domínio da aplicação e também os diferentes estágios, dentro do ciclo de desenvolvimento do framework, nos quais o componente foi introduzido. Os componentes elementares, tais como mecanismo para comunicação e concorrência lógica, determinam os elementos da arquitetura nos quais deve-se escrever toda a aplicação. Os elementos de projeto básico são classes intermediárias e praticamente independentes da aplicação, embora sejam concebidos com um domínio específico em mente. Os demais componentes são dependentes do domínio e, em geral, são especializações dos componentes básicos.

As publicações existentes sobre a linguagem de padrões G++ e o framework G++ , que consistem em diversos artigos (Aarsten et al., 2000; Brugali e Menga, 1999; Brugali et al., 1997, 2000) e em uma tese de doutorado(Brugali, 1998), não deixam claro como o framework foi implementado a partir da linguagem de padrões e como a linguagem de padrões auxilia na instanciação do framework. É dito que ela guiou o processo de construção, mas não foi documentado como isso foi feito.

2.6 Abordagens para Construção de frameworks

A construção de frameworks vem sendo alvo de pesquisa nos últimos anos, tendo sido propostas diversas abordagens, entre as quais destacam-se a de Pree (1999), a de Schmid (1997, 1999), a de Froehlich et al. (1997), a de Roberts e Johnson (1998) e a de Bosch et al. (1999). Pree (1999) propõe um processo para construção de frameworks no qual inicia-se pela definição de um modelo de objetos de uma aplicação específica, que é refinado por meio de um ciclo de construção repetido sucessivamente. Esse ciclo inclui diversos passos: inicialmente identificam-se os pontos variáveis do framework, que são documentados por meio de cartões de pontos variáveis (em inglês, *hot spot cards*). Os pontos variáveis são então projetados e implementados e o framework é testado para garantir que satisfaça aos requisitos do domínio. Novos pontos variáveis podem ser encontrados neste passo e o ciclo pode se repetir. No fim de cada iteração, o framework é avaliado para determinar se pode ser liberado para uso ou se deve passar por outro ciclo de refinamento. Esta abordagem sistematiza a construção do framework, mas não define o processo de instanciação por meio do qual as aplicações concretas são obtidas.

Outra técnica para construção de frameworks a partir de um modelo específico é a proposta por Schmid (1997, 1999). Ele sugere que o framework seja desenvolvido por generalização sistemática com base em um modelo de classes de uma aplicação fixa. Inicialmente faz-se uma análise de alto nível para determinar os pontos variáveis e estabelecer os principais aspectos do sistema que precisam ter flexibilidade. A seguir, cada ponto variável é analisado em detalhes, produzindo-se sua especificação, que passará ainda pelo projeto de alto nível, gerando vários sub-sistemas de pontos variáveis. No último passo, deve-se transformar o modelo de classes da aplicação fixa no modelo de classes do framework, por meio da substituição de grupos de classes do modelo original pelos sub-sistemas de pontos variáveis correspondentes. Esta abordagem também não define o processo de instanciação do framework resultante.

Bosch et al. (1999) apresenta um modelo para desenvolvimento de frameworks, composto de seis fases. A primeira trata da análise do domínio, que é executada para descrever o domínio coberto pelo framework e para captar seus principais conceitos e requisitos, resultando em um modelo de análise do domínio. Na segunda fase, cria-se o projeto arquitetural do framework. Na

terceira fase, refina-se esse projeto, que é implementado na quarta fase, utilizando uma linguagem de programação específica. Na quinta fase, o framework é testado para avaliar tanto sua funcionalidade quanto sua usabilidade. Finalmente, na sexta fase, o framework é documentado por meio de um guia do usuário e da documentação do seu projeto. Nota-se nessa abordagem uma preocupação maior (em relação às abordagens de Pree e de Schmid) com a usabilidade do framework, visto que a fase de teste explicita a avaliação da facilidade de uso do framework e que a fase de documentação produz um guia do usuário.

Uma outra abordagem para tentar resolver o problema de instanciação de frameworks é a de Froehlich e outros (Froehlich et al., 1997), que sugere a documentação dos "ganchos" (em inglês, *hooks*) do framework por meio da descrição detalhada de seus pontos flexíveis, para facilitar sua instanciação para aplicações particulares. Os ganchos são identificados pelo construtor do framework, que possui alto conhecimento sobre seu projeto. Esse conhecimento é armazenado na descrição dos ganchos e pode ser usado pelo desenvolvedor de aplicações, que terá condições de saber o que deve ser completado ou estendido no framework para adaptá-lo de acordo com a aplicação específica. Esta abordagem facilita a instanciação do framework, pois o desenvolvedor de aplicações pode se concentrar apenas nos pontos variáveis do domínio. No entanto, se o framework for muito grande, a quantidade de ganchos provavelmente também será grande, tornando complexa a tarefa de identificar quais deles são pertinentes ao sistema específico a instanciar.

Uma das abordagens preocupadas tanto com a construção quanto com a instanciação de framework é a de Roberts e Johnson (1998), que apresenta uma linguagem de padrões para o desenvolvimento de frameworks orientados a objetos, denominada "Evolução de Frameworks". Essa linguagem de padrões possui nove padrões que, quando aplicados, levam à construção de um framework caixa preta com todos os recursos necessários para facilitar sua instanciação. O primeiro padrão sugere que três aplicações concretas sejam inicialmente desenvolvidas e depois sejam generalizadas, de forma gradual, para produzir um framework caixa branca. Os padrões seguintes levam a diversas iterações sobre o framework, transformando-o de forma progressiva em um framework caixa preta, por meio do uso de bibliotecas de componentes, objetos plugáveis e objetos de granularidade menor. Os dois últimos padrões sugerem o desenvolvimento de um construtor visual e de ferramentas de linguagem para facilitar o uso do framework.

Em suma, as abordagens sugeridas por Pree, Schmid e Roberts e Johnson partem de modelos de aplicações particulares, incluindo a flexibilidade desejada posteriormente. Na abordagem de Froehlich, o conhecimento sobre os pontos variáveis é proveniente do desenvolvedor do framework, que o adquiriu por meio de experiência prática ou análise de domínio. Na abordagem de Bosch o modelo de análise do domínio é obtido no início do processo, o que torna os pontos variáveis do framework mais previsíveis. A abordagem proposta nesta tese segue um raciocínio mais próximo ao de Bosch, já que uma linguagem de padrões é utilizada para captar as informações sobre o

domínio da aplicação e para guiar o desenvolvimento do framework. Em particular, os pontos variáveis do framework são identificados no início para minimizar o número de ciclos de iteração e, ao mesmo tempo, facilitar o subseqüente aprendizado e uso do framework. O processo aqui proposto é similar aos processos mencionados acima no que diz respeito à fase de projeto, mas é guiado pela linguagem de padrões.

2.7 Abordagens para Instanciação de Frameworks

Um framework é uma técnica poderosa para aumentar o reuso de software, visto que inúmeras aplicações distintas podem ser obtidas por meio de sua instanciação. Entretanto, o processo de instanciação é, na maioria das vezes, muito complexo, exigindo conhecimento considerável a respeito do projeto e implementação do framework. Para alcançar a flexibilidade desejada, os frameworks contém construções especiais que podem dificultar seu entendimento. Conforme descrito na seção 2.2.2, os frameworks possuem uma parte fixa que reflete o comportamento comum do domínio e partes variáveis que devem permanecer flexíveis por representarem aspectos que mudam de uma aplicação para outra. Padrões de projeto (Gamma et al., 1995) ou meta-padrões (Pree, 1999) são algumas das maneiras de implementar as partes variáveis de um framework. O framework resultante possui algumas classes abstratas com um ou mais métodos que devem ser sobrepostos nas subclasses concretas da aplicação.

O principal problema ao instanciar um framework é saber exatamente quais pontos variáveis devem ser adaptados e como adaptá-los. Diversas abordagens foram propostas para resolver esse problema (Beck e Johnson, 1994; Brugali e Menga, 1999; Gangopadhyay e Mitra, 1995; Johnson, 1992; Ortigosa e Campo, 2000; Pree et al., 1995). Basicamente, essas abordagens contam com a documentação do framework para obter a informação necessária para instanciar aplicações específicas. Considerando-se um framework caixa branca, essa documentação é, em geral, constituída da hierarquia de classes do framework, das classes abstratas que precisam ser especializadas na nova aplicação, dos métodos a serem sobrepostos nessas classes e de exemplos de aplicações derivadas a partir do framework.

Existem pelo menos quatro tipos de abordagens para instanciação de frameworks: pela documentação do framework, por exploração de exemplos, por meio de padrões e por meio de *cookbooks*. A primeira abordagem consiste do estudo da documentação do framework, composta de sua hierarquia de classes, código fonte e outros documentos. Isso pode ser feito com treinamento convencional ou por meio de tutoriais especialmente elaborados com essa finalidade. As desvantagens desta abordagem são: o tempo necessário para aprender o framework e a dificuldade para determinar se o nível de compreensão é suficiente para iniciar o uso do framework.

A exploração de exemplos, como a proposta por Gangopadhyay e Mitra (1995), consiste na análise de aplicações existentes, construídas a partir do framework, para identificar as adaptações necessárias para obtenção do sistema específico. Algumas das dificuldades encontradas nessa abordagem são: a dificuldade de encontrar, entre os exemplos, uma aplicação similar à aplicação a ser desenvolvida; o problema de decidir o que fazer quando uma funcionalidade particular não é encontrada nos exemplos; o problema de decidir o que fazer quando uma funcionalidade particular é encontrada no exemplo, mas com características adicionais não exigidas na nova aplicação; e a dificuldade de saber quando considerar encerrada a instanciação, já que a análise dos exemplos é feita de maneira *ad hoc*.

Padrões de software podem ser empregados para documentar um framework e ajudar a instanciá-lo. Três trabalhos foram publicados nessa linha (Beck e Johnson, 1994; Brugali e Menga, 1999; Johnson, 1992) conforme visto na seção 2.5, mas eles não evoluíram rumo à definição de um processo a ser utilizado por outros desenvolvedores de frameworks. Esta tese explora a idéia do uso de padrões, em particular linguagens de padrões, no desenvolvimento e instanciação de frameworks.

Cookbooks (ou "livros de receitas") podem ser preparados como um conjunto de tarefas exigidas para adaptar um framework para uma aplicação específica, como em uma receita culinária. Diversos trabalhos baseiam-se nessa idéia (Krasner e Pope, 1988; Ortigosa e Campo, 2000; Pree et al., 1995), mas existem limitações tais como pequena flexibilidade e dificuldade em encontrar a receita adequada ao problema em mãos. Além disso, é pouco provável que as tarefas que compõem uma receita possam ser executadas passo a passo, pois uma escolha feita durante o processo de especialização pode causar mudanças no restante da receita. O trabalho proposto nesta tese também utiliza um processo bem definido para guiar a instanciação do framework.

Tanto a abordagem de *cookbooks* quanto a abordagem de padrões têm o intuito de fazer a instanciação do framework com o mínimo possível de conhecimento sobre o framework, ou seja, os usuários do framework concentram-se na adaptação dos pontos variáveis relativos ao problema específico que eles têm em mãos. Deve-se ressaltar que a maioria dos pontos variáveis são adaptados por meio de métodos-gancho (ver seção 2.2.2) e, portanto, detalhes de implementação estão envolvidos nessa tarefa.

2.8 Considerações Finais

Este Capítulo apresentou os conceitos básicos necessários para o entendimento do trabalho descrito nesta tese. Padrões de software e frameworks foram definidos e os trabalhos mais importantes relacionados à pesquisa sobre construção e instanciação de frameworks foram relatados. Com isso,

criou-se a motivação e foi exposta a relevância da pesquisa sobre reuso de software, particularmente sobre frameworks e padrões.

As pesquisas existentes sobre o relacionamento entre linguagens de padrões e frameworks foram apresentadas, tendo sido detectado que, na maioria delas, uma linguagem de padrões foi construída e utilizada somente após o desenvolvimento do framework. O único caso no qual a linguagem de padrões foi desenvolvida antes do framework é o da linguagem de padrões G++. Contudo, não há documentação do processo utilizado para conduzir a construção e instanciação do framework com base em tal linguagem de padrões. Assim, fica caracterizada a ausência de um método para desenvolvimento e instanciação de um framework com base em uma linguagem de padrões, propiciando a oportunidade de pesquisa desta tese.

Os próximos três capítulos apresentam o processo geral proposto para desenvolvimento e instanciação de um framework com base em uma linguagem de padrões para um domínio específico. Uma visão geral do processo proposto é mostrada no próximo capítulo, juntamente com o detalhamento do primeiro passo do processo. Os detalhes dos demais passos são fornecidos nos dois capítulos posteriores.

CAPÍTULO

3

Um processo Geral para Desenvolvimento e Instanciação de Frameworks a partir de uma Linguagem de Padrões

3.1 Considerações Iniciais

Este Capítulo apresenta o processo geral que é proposto nesta tese, cujos passos individuais são descritos detalhadamente ao longo da tese. O processo geral é composto de quatro passos: o primeiro trata da construção da linguagem de padrões, posteriormente descrita em detalhes na seção 3.3; o segundo refere-se à construção do framework caixa branca com base na linguagem de padrões, descrito na seção 4.2; o terceiro mostra como construir uma ferramenta (*Wizard*) para instanciação automática de aplicações com base no framework caixa branca, descrito na seção 5.2; e o quarto preocupa-se em definir o processo de instanciação do framework, seja de forma manual (descrito na seção 4.4) ou utilizando o *Wizard* (descrito na seção 5.6).

O Capítulo está organizado da seguinte forma: na seção 3.2 apresenta-se a visão geral do processo proposto. Na seção 3.3 detalha-se o primeiro passo desse processo, que é o sub-processo[1] de construção de uma linguagem de padrões para um domínio particular. Na seção 3.4 apresenta-se o sub-processo de uso de uma linguagem de padrões. Na seção 3.5 a linguagem de padrões para Gestão de Recursos de Negócios é apresentada, objetivando ilustrar os processos propostos nas seções 3.3 e 3.4. Na seção 3.6 são apresentadas as conclusões parciais do trabalho, enfocando o processo geral proposto e seu primeiro passo.

3.2 Visão Geral do Processo

O processo geral proposto (Braga e Masiero, 2002b) é ilustrado informalmente na Figura 3.1, que deve ser interpretada da seguinte forma: uma linguagem de padrões é construída para auxiliar a modelagem de sistemas em um domínio específico. Tal linguagem consiste basicamente de padrões de análise, ou seja, padrões a serem aplicados durante a fase de análise de sistemas de tal domínio, e serve de base para a construção de um framework que apóie a implementação de sistemas modelados com a linguagem de padrões. O framework implementa a funcionalidade comum ao domínio e também seus pontos variáveis. Para isso utiliza técnicas como padrões de projeto e meta-padrões para implementar a variabilidade desejada. Assim, as classes pertencentes aos padrões de análise possuem classes correspondentes no framework, o que, embora não sendo necessariamente na proporção de um para um, facilita a instanciação do framework a aplicações específicas. Para facilitar ainda mais a instanciação de aplicações, uma ferramenta pode ser construída (*Wizard*), cuja interface gráfica com o usuário espelhe os padrões da linguagem de padrões. Isso torna a instanciação mais fácil, pois sabendo-se utilizar a linguagem de padrões para modelar o sistema pode-se facilmente interagir com o *Wizard* para obter o sistema executável, sem ter que conhecer detalhes de implementação do framework caixa branca.

Utilizando uma notação mais formal (Ross, 1977) para ilustrar o processo proposto, a Figura 8.1 é obtida. Quatro passos (ou sub-processos) compõem o processo geral: o primeiro, descrito na seção 3.3, utiliza sistemas específicos e experiência prática em um dado domínio para construir uma linguagem de padrões, a qual é utilizada como entrada para todos os demais passos. Além disso, essa linguagem de padrões pode ser utilizada de forma independente para auxiliar desenvolvedores, principalmente aqueles inexperientes, a modelar sistemas no domínio por ela coberto.

No segundo passo, descrito na seção 4.2, um framework caixa branca é construído para apoiar a implementação de aplicações específicas modeladas pela linguagem de padrões. Esse framework

[1] O termo “processo” será utilizado no decorrer do texto com o mesmo significado de “sub-processo”, com exceção de alguns pontos nos quais fizer-se necessário enfatizar que um processo é composto de outros processos mais internos.

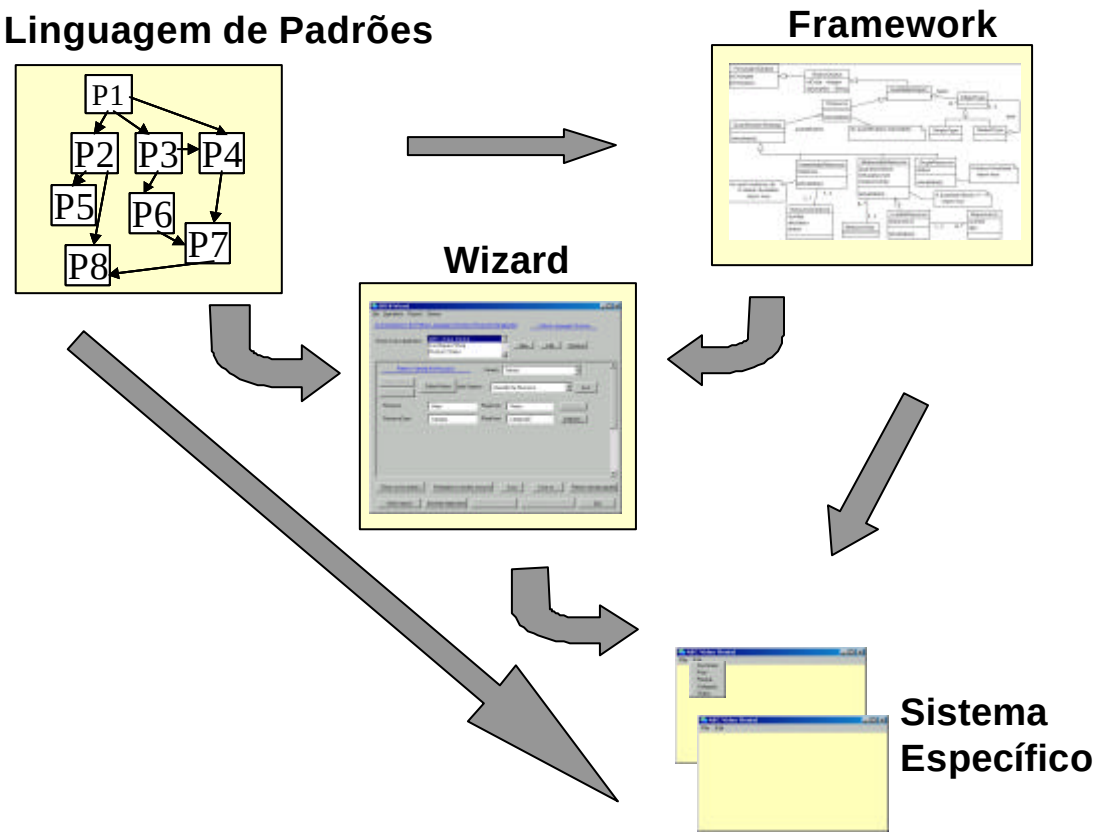


Figura 3.1: Ilustração da Abordagem Proposta

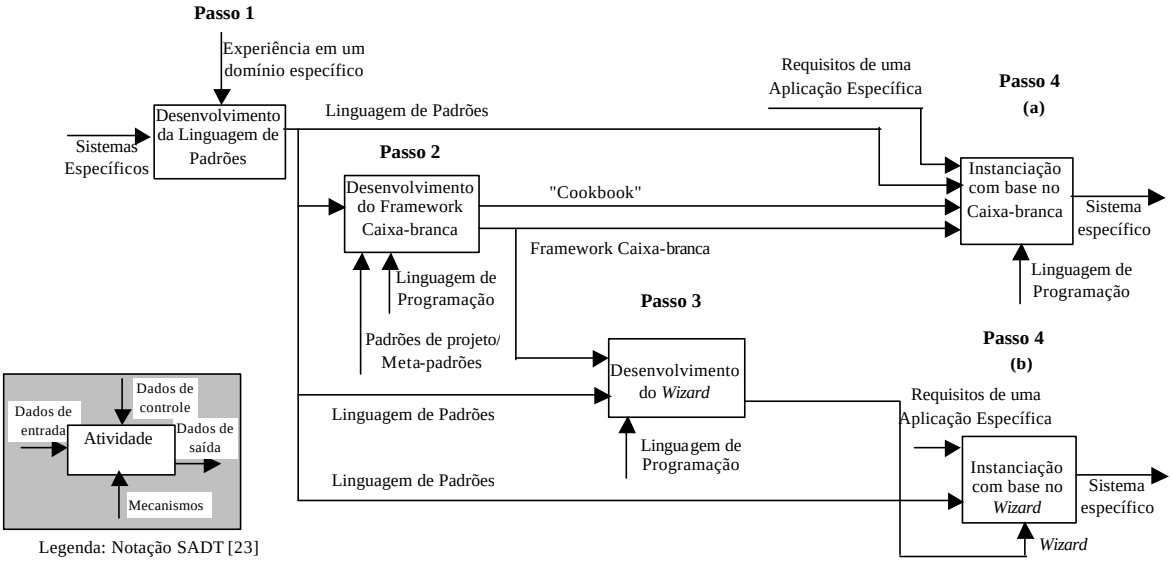


Figura 3.2: Processo Proposto

possui classes concretas e abstratas implementadas em uma linguagem de programação específica, que correspondem, embora não necessariamente de forma direta, às classes de cada um dos padrões de análise da linguagem de padrões. É um framework caixa branca (ver seção 2.2.2) porque, para derivar aplicações a partir dele, é necessário criar subclasses que herdem de suas classes abstratas. Recursos como padrões de projeto (Gamma et al., 1995) e meta-padrões (Pree, 1995) são utilizados para oferecer a flexibilidade desejada ao framework. Além disso, outro resultado desse passo é um manual de uso do framework (*cookbook*), que inclui o mapeamento da linguagem de padrões para as classes do framework, de forma que, sabendo-se quais foram os padrões aplicados para modelar um sistema específico, pode-se identificar quais classes do framework devem ser especializadas, bem como quais métodos devem ser sobrepostos durante a instanciação.

No terceiro passo, descrito na seção 5.2, é construída uma ferramenta para automatizar a instanciação do framework caixa branca a aplicações específicas. A interface gráfica dessa ferramenta é totalmente baseada na linguagem de padrões, de forma que sua utilização torna-se intuitiva para desenvolvedores que conhecem a linguagem de padrões. Tendo o desenvolvedor fornecido à ferramenta todas as informações sobre os padrões aplicados para modelar seu sistema alvo, a ferramenta baseia-se no mapeamento da linguagem de padrões para as classes do framework, a fim de gerar o código das novas classes. O resultado é um sistema executável acompanhado do código-fonte, que pode ser modificado manualmente pelo engenheiro de software para atender requisitos não cobertos pela linguagem de padrões.

O quarto e último passo do processo geral é a instanciação do framework caixa branca a aplicações específicas. Duas alternativas de instanciação são possíveis: com base no framework caixa branca e com base no *Wizard*, descritas nas seções 4.4 e 5.6, respectivamente. Na primeira alternativa, utiliza-se diretamente o framework caixa branca e seu *cookbook*, criando manualmente as novas classes da aplicação e seus respectivos métodos. Na segunda alternativa, o *Wizard* é utilizado para obtenção das novas classes e métodos. Em ambas as alternativas a linguagem de padrões é utilizada inicialmente para modelar o sistema, de acordo com seus requisitos específicos.

Deve-se ressaltar que o processo é iterativo, podendo haver, durante quaisquer de suas atividades, o retorno de informações que podem ser úteis no refinamento de atividades já realizadas. Por exemplo, durante o desenvolvimento do framework caixa branca podem ser identificadas funcionalidades não tratadas pela linguagem de padrões. Essa informação pode ser utilizada para complementar a linguagem de padrões, por meio do acréscimo de novos participantes, novos variantes, ou até mesmo novos padrões, ou pode ser documentada adequadamente para inclusão em futuras versões da linguagem de padrões e do framework. Outro exemplo que reforça o aspecto iterativo do processo é o passo de instanciação do framework, tanto pelo framework caixa branca quanto pelo *Wizard*. A instanciação é o momento no qual uma nova aplicação deve ser desenvolvida por meio da adaptação dos pontos flexíveis do framework. Durante os estudos de caso

efetuados, notou-se que é nesse momento que são identificadas a maioria das funcionalidades não cobertas, mas desejáveis de serem incorporadas ao framework e à linguagem de padrões. Para não poluir o diagrama da Figura 8.1, optou-se por não acrescentar as setas de retorno para representar a iteratividade do processo.

3.3 O Processo de Construção da Linguagem de Padrões

O primeiro passo do processo proposto consiste em um sub-processo para construção de uma linguagem de padrões para um domínio específico. Deve-se ressaltar que tal linguagem de padrões é composta de padrões de análise, cada qual contendo um ou mais diagramas de classe que ilustrem a(s) solução(ões) para o problema em questão. É também desejável que sejam fornecidos os relacionamentos entre os padrões que compõem a linguagem de padrões resultante, podendo ser denotados por um grafo ou, mais informalmente, por uma seção "Próximos padrões" adicionada em cada um dos padrões.

A Figura 8.2 mostra os sub-passos necessários para construção de uma linguagem de padrões, apresentados nas seções seguintes. A versão aqui exposta foi obtida após o refinamento de uma versão preliminar, proposta durante um trabalho de mestrado (Ré e Masiero, 2002). O passo 1.1 da Figura 8.2 encontra-se mais detalhado no trabalho de mestrado mencionado, que teve seu enfoque principal na fase de engenharia reversa para obtenção do modelo do domínio. Aqui, essa atividade é apresentada de maneira mais geral, podendo ser feita por meio de engenharia reversa ou por análise de domínio. Foram feitas algumas melhorias nos passos 1.2 a 1.5 da Figura 8.2 desde a publicação da dissertação de mestrado, acrescentando mais sugestões para tornar esses passos mais fáceis de serem seguidos por desenvolvedores de linguagens de padrões.

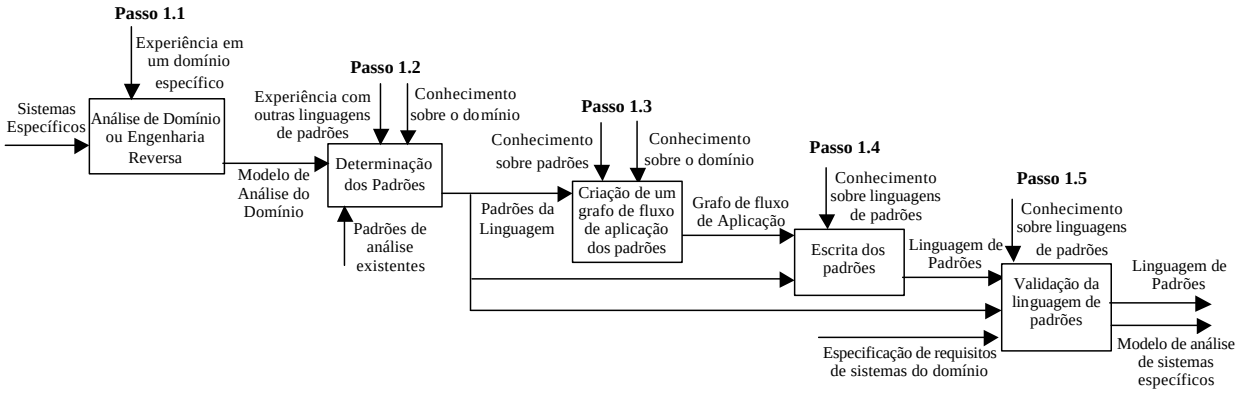


Figura 3.3: Processo de Construção da Linguagem de Padrões

3.3.1 Análise de Domínio ou Engenharia Reversa

Conforme visto na seção 2.2.1, padrões são documentados a partir de experiência no desenvolvimento de software. Portanto, para construir uma linguagem de padrões que abranja aplicações em um dado domínio, é necessário observar as soluções comumente empregadas para resolver problemas recorrentes desse domínio.

Assim, o ponto de partida para criação de uma linguagem de padrões é obter um modelo que represente o domínio alvo, ou seja, capte a funcionalidade presente na grande maioria das aplicações do domínio. Isso pode ser conseguido de diversas maneiras: pode ser conduzida a análise do domínio, utilizando técnicas como a descrita por Gomaa (Gomaa, 1996) ou aquelas selecionadas por Prieto-Diaz (Prieto-Diaz e Arango, 1991); pode ser realizada a engenharia reversa de aplicações do domínio, similarmente ao que foi feito para o domínio de leilões virtuais por Ré et. al. (Ré et al., 2001); ou pode ser utilizada a própria experiência prática de desenvolvimento de aplicações no domínio, o que pode ser considerado uma combinação das duas alternativas anteriores, pois as informações que o engenheiro de software possui sobre aplicações já desenvolvidas são similares ao resultado que se obteria via engenharia reversa, e tais informações devem ser utilizadas na construção de um modelo de análise do domínio.

Uma das maneiras de conseguir reunir informações sobre as funcionalidades básicas do domínio é por meio da engenharia reversa de alguns sistemas já existentes. Para tal, é necessário que a engenharia reversa produza modelos intermediários representando cada um desses sistemas, para depois avançar para um processo de generalização dos modelos que culminará com o modelo de análise do domínio (Ré et al., 2001).

Independentemente da forma utilizada para conseguir as informações sobre o domínio, o resultado desse passo é o modelo de análise do domínio, que para a proposta desta tese, é constituído de um modelo estático e de um modelo dinâmico. O modelo estático pode ser expresso usando uma notação orientada a objetos, como por exemplo o modelo de classes da UML (do inglês *Unified Modeling Language*) (Rational, 2000). Nele, cada classe possui como atributos e métodos somente aqueles que forem comuns ao domínio, com nomes genéricos que devem ser explicados em um glossário do domínio. O relacionamento entre as classes também pode possuir nomes genéricos que reflitam a semântica inerente ao domínio. O modelo dinâmico pode ser expresso, por exemplo, pelo diagrama de seqüência da UML, que mostra como funciona a comunicação entre os objetos para implementar as operações do sistema, sendo também definido com nomes genéricos.

3.3.2 Determinação dos Padrões

O modelo de análise do domínio, resultante do passo anterior, é utilizado para definir os padrões da linguagem de padrões (passo 1.2). Essa atividade depende bastante do conhecimento e experiência

sobre padrões do desenvolvedor da linguagem de padrões. Entretanto, algumas recomendações devem ser seguidas para que os padrões sejam definidos de forma uniforme e com maior possibilidade de reuso:

1. Padrões de análise existentes na literatura devem ser analisados. É provável que alguns deles estejam presentes no modelo de análise do domínio tratado. No caso de serem encontrados esses padrões, o problema resolvido por eles deve ser especializado para o caso de tal domínio e o padrão deve receber um novo nome que reflita o problema específico do domínio.

2. Outras linguagens de padrões de análise, para domínios similares, devem ser estudadas, observando quais são seus padrões constituintes, como estão documentados e como se relacionam entre si. Isso contribui para o conhecimento sobre padrões e aumenta as chances do desenvolvedor da linguagem de padrões definir os padrões de forma correta e mais fácil de reusar por outros usuários da linguagem de padrões. Nesse momento, pode-se optar por um formato para os padrões, ou pelo menos identificar dois ou três formatos mais apropriados.

3. A determinação dos padrões deve iniciar pela análise das classes básicas do modelo de domínio obtido no passo anterior. Classes básicas são aquelas que estão envolvidas nas funções básicas ou principais representadas no modelo do domínio, ou seja, aquelas que estão presentes em todos os sistemas desse domínio.

4. As classes básicas devem ser estudadas em busca de grupos de duas ou mais classes que sejam responsáveis por funções importantes no sistema. Isso pode ser feito no próprio modelo de classes do domínio, por exemplo destacando-as com uma cor diferente, ou criando-se modelos menores referentes a cada função. Esses grupos de classes básicas representarão os principais padrões da linguagem, já que cada padrão deve referir-se a uma função específica realizada no domínio. Isso facilita o reuso, pois conduz a padrões com problemas/soluções sucintos e objetivos.

5. As classes complementares, diferentemente das classes básicas, adicionam melhorias em relação a um padrão, ou adicionam uma função diferente em relação às funções básicas do modelo do domínio. As classes complementares geralmente representam funcionalidades que são opcionais para o funcionamento de sistemas do domínio estudado. Assim, classes complementares têm maior possibilidade de serem padrões opcionais, ou de se juntarem a padrões já existentes na linguagem, formando variantes dos padrões principais e se tornando elementos opcionais dos padrões principais. Novamente, fica a cargo do analista a decisão de como estabelecer quais classes fazem parte de um padrão. A existência de padrões op-

cionais permite que eles sejam ignorados durante o uso da linguagem de padrões, caso a funcionalidade por eles coberta não for necessária na aplicação específica.

6. Cada padrão deve receber um nome que deve abstrair o conteúdo do padrão, facilitando sua identificação e utilização por outros analistas. Uma forma de facilitar a nomeação do padrão é por meio da observação de suas funções. O nome escolhido deve ser significativo o suficiente para que sua simples citação traga consigo o valor semântico envolvido no problema que o padrão resolve. Podem ser utilizadas metáforas ou expressões, desde que sejam conhecidas no domínio coberto pela linguagem de padrões.

7. Após identificar e nomear os padrões, uma tabela pode ser elaborada contendo apenas o nome do padrão, o problema resolvido e uma síntese da solução proposta. Essa tabela ajuda o desenvolvedor da linguagem de padrões a manter-se fiel aos seus objetivos para cada um dos padrões e será utilizada tanto na criação do grafo da linguagem de padrões quanto durante a fase de escrita dos padrões.

3.3.3 Criação de um grafo de fluxo de aplicação dos padrões

O terceiro passo para a construção da linguagem de padrões é a análise da interação entre os diversos padrões que a compõem. Deve-se estabelecer a ordem na qual os padrões devem ser aplicados, de forma a facilitar a modelagem de sistemas usando a linguagem de padrões. Além disso, deve-se mostrar quais são os padrões principais e opcionais da linguagem, informações que são obtidas por intermédio da descrição geral do padrão. É importante salientar que o grafo deve indicar a interação entre os padrões, mas não tem a intenção de mostrar como é a ordem de funcionamento do sistema resultante.

Uma das estratégias para criar o grafo é iniciar pelos padrões que representam as funcionalidades mais básicas do domínio, e acrescentar, de forma gradual, os padrões que correspondem às funcionalidades mais específicas, terminando com os padrões que são opcionais, ou seja, os padrões que representam problemas nem sempre encontrados em aplicações do domínio. Entretanto, deve-se avaliar com cuidado o melhor local para posicionar os padrões opcionais, pois muitas vezes eles dependem da aplicação de algum padrão obrigatório e devem ser aplicados logo na seqüencia desses.

Além de elaborar um diagrama ou grafo para ilustrar a interação entre os padrões, é desejável repetir essa informação sobre a interação por meio de uma seção em cada padrão, denominada “Próximos Padrões”. O intuito principal é guiar o usuário da linguagem de padrões a respeito dos próximos padrões que ele deve considerar após ter aplicado ou não determinado padrão.

O fluxo de aplicação dos padrões, além de ser influenciado pelos padrões opcionais, também pode ser influenciado por outros elementos dos padrões, como por exemplo, os variantes, as clas-

ses e os elementos opcionais. Podem ocorrer casos em que a aplicação de um determinado padrão implica na adição de elementos em outros padrões, outros casos em que a aplicação de um padrão exige a aplicação de outro padrão ou, ainda, casos em que um padrão pode ser aplicado somente se outro padrão for aplicado também. Dependendo da notação utilizada para representar o grafo, talvez essas situações não consigam ser mostradas diretamente no grafo, sendo então documentadas apenas na seção "Próximos Padrões" de cada padrão.

3.3.4 Escrita dos Padrões

No passo 1.4 são escritos os padrões da linguagem de padrões. O analista deve estabelecer um padrão para descrever os padrões da linguagem, como por exemplo, "nome", "contexto", "problema", "estrutura", "participantes", "padrões relacionados" e "próximos padrões", entre outros. Existem várias propostas de formatos de padrões, dentre elas as sugeridas por Alexander et al. (1977), Appleton (1997) e Meszaros e Doble (1998). Independentemente do formato adotado, os componentes básicos de um padrão devem estar presentes, que são:

- nome: deve ser relacionado com a intenção do padrão;
- contexto: deve descrever como e quando o padrão pode ser aplicado;
- problema: deve descrever, de maneira direta, o problema que o padrão se propõe a resolver, frente ao contexto anteriormente especificado;
- forças: deve mostrar quais são os fatores que, frente ao contexto especificado e ao problema estabelecido, influenciam a solução proposta e como ela é implementada. Também pode apresentar limitações e restrições da solução, desvantagens em utilizar o padrão ou o porquê da solução não utilizar outras estratégias;
- solução: pode ser escrito de várias formas, desde que apresente de maneira clara a solução proposta pelo padrão. Podem ser criadas ilustrações da solução, com sub-seções para descrever os participantes da solução bem como a colaboração entre eles.

Escolhido o formato do padrão, sua escrita deve basear-se na descrição geral dos padrões e nas classes que compõem os padrões, obtidas no passo 1.2. É importante notar que um padrão é muito mais que uma estrutura de classes e uma descrição sobre elas, ele apresenta basicamente todas as informações do contexto no qual pode ser aplicado, o problema que resolve, bem como as forças que atuam para formar a solução (Fowler, 1997).

Com o objetivo de melhorar os padrões da linguagem, o analista pode pesquisar outros padrões do mesmo domínio ou de um domínio próximo. Durante essa pesquisa, o analista pode descobrir

que os padrões por ele definidos são variantes de outros padrões, ou complementam outra linguagem de padrões. Assim, essa pesquisa também é importante para fazer as referências corretas a outros padrões ou linguagens de padrões, fornecendo outras alternativas de solução caso a proposta pelo padrão não seja adequada. O analista também pode procurar por padrões de análise e projeto que melhorem as soluções propostas, complementando a linguagem de padrões com referências aos padrões ou adotando os padrões como parte da linguagem.

A documentação de cada um dos padrões é uma tarefa que demanda tempo, sendo necessário, na maioria das vezes, efetuar diversas iterações até conseguir-se um resultado satisfatório. A comunidade de padrões de software recomenda que toda linguagem de padrões seja submetida a uma sessão do escritor (em inglês, *writers' workshop*), por exemplo por meio da submissão da mesma para uma Conferência PLoP (ver primeira nota de rodapé na seção 2.2). Além de oferecer um período de melhoria do padrão alguns meses antes da Conferência, por meio da interação com um "pastor" (do inglês *shepherd*) especialmente designado para avaliar o artigo, durante a Conferência o padrão (ou conjunto de padrões) é discutido individualmente em sessão de cerca de uma hora, o que contribui bastante para sua melhoria.

3.3.5 Validação da Linguagem de Padrões

A validação (passo 1.5) finaliza o processo de criação de uma linguagem de padrões. Ela pode ser feita por intermédio de sua aplicação a diferentes sistemas do domínio estudado, seguindo o processo de uso proposto na seção 3.4. Basicamente, essa atividade é constituída pelo estudo do documento de requisitos da aplicação a ser modelada, estudo e aplicação da linguagem de padrões com base no documento de requisitos e avaliação do modelo de classes, confrontando os requisitos desejados com os modelos de classes da aplicação. Quanto mais aplicações forem modeladas com a linguagem de padrões, melhor será sua validação, além de contribuir para a melhoria da própria linguagem, pois novas características podem ser a ela incorporadas.

3.4 Processo de Uso de uma Linguagem de Padrões

Em geral, linguagens de padrões de análise são auto-explicativas, já que possuem todos os elementos necessários para serem utilizadas por desenvolvedores inexperientes. No entanto, mostra-se, a seguir, alguns passos que devem ser seguidos ao utilizar uma linguagem de padrões para assegurar que seja tirado o máximo proveito das vantagens que elas oferecem.

1. Estude a linguagem de padrões detalhadamente, para conhecer qual o domínio por ela tratado, quais os padrões disponíveis e quando e como aplicá-los.

2. Produza o documento de requisitos do sistema específico, seguindo as recomendações da Engenharia de Software. Tal documento deve descrever toda a funcionalidade desejada para o sistema a ser desenvolvido. Esse passo é livre, não sendo tratado em detalhes aqui.

3. Leia com atenção o documento produzido no passo 2, para ter um conhecimento amplo dos requisitos do sistema, necessário para decidir se a linguagem de padrões de análise e o respectivo framework, caso exista, podem ser utilizados na construção de tal sistema.

4. Use a linguagem de padrões de análise para modelar o sistema específico, empregando as informações presentes nos diversos elementos de cada padrão para decidir se ele é aplicável ou não.

5. Para cada padrão aplicável esboce o diagrama de classes referente à porção do sistema que utilizará tal padrão, usando cores ou símbolos diferentes para destacar possíveis atributos, métodos ou operações acrescentados aos do padrão. Esse diagrama de classes cresce de maneira gradual, à medida que novos padrões são aplicados.

6. Para cada padrão aplicável faça uma marcação no documento de requisitos para indicar quais dos requisitos foram atendidos pela aplicação de tal padrão.

7. Para cada padrão aplicável utilize textos explicativos, por exemplo em forma de "balões", para indicar os papéis desempenhados por cada classe em tal padrão.

8. Para cada padrão aplicável anote o nome do padrão utilizado e seu variante ou sub-padrão (se houver).

9. Se o padrão possuir um elemento "próximos padrões" (ou equivalente), então este indica os possíveis padrões a investigar após ter aplicado (ou não) o padrão atual. Isso define diversos caminhos a seguir, que devem ser anotados pelo desenvolvedor e investigados um a um.

10. Confira o documento de requisitos em busca de requisitos não atendidos ou atendidos parcialmente pela linguagem de padrões. Complemente o diagrama de classes com tais requisitos, por meio da adição de novos atributos, classes, relacionamentos, métodos ou operações, usando cor diferente e fazendo uma anotação no documento de requisitos que identifique a não cobertura do requisito pela linguagem de padrões.

11. Elabore uma tabela contendo o "Histórico de Padrões e Variantes Utilizados", com quatro colunas: a primeira mostra o número e o nome do padrão utilizado, a segunda mostra o variante ou sub-padrão utilizado (use a palavra *Default* se tiver utilizado o padrão tal qual apresentado na solução), a terceira apresenta o nome da classe participante do padrão e a

quarta indica o nome da classe da aplicação que desempenha o papel do participante da terceira coluna.

Os passos acima definidos fornecem diretrizes para aplicação da linguagem de padrões de forma sistemática. O resultado é um diagrama de classes, complementado com informações sobre os padrões utilizados e os requisitos não atendidos pela linguagem de padrões. Isso facilita a futura implementação do sistema usando um framework que apóie a linguagem de padrões, já que são fornecidas informações sobre os padrões utilizados. Além disso, o fato da funcionalidade não coberta pelo framework ter especial destaque no diagrama, facilita a identificação das partes da aplicação que precisarão ser implementadas à parte. Essas informações podem, também, ser utilizadas para melhoria da linguagem de padrões, incluindo novos padrões ou oferecendo soluções alternativas para contextos diferentes.

3.5 A Linguagem de Padrões para Gestão de Recursos de Negócios

3.5.1 Visão Geral

A Linguagem de Padrões para Gestão de Recursos de Negócios, denominada GRN, é o resultado de uma evolução de mais de dez anos de prática da autora no desenvolvimento de sistemas para pequenas e médias empresas no domínio de gestão de recursos de negócios, que é um domínio particular de sistemas de informação. As similaridades existentes entre tais sistemas motivou a criação de uma linguagem de padrões que pudesse ser utilizada por analistas durante o desenvolvimento de sistemas nesse domínio. A GRN 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. A GRN aqui apresentada (Braga et al., 2002) foi estendida a partir da linguagem de padrões originalmente apresentada em uma Conferência PLoP (Braga et al., 1999).

A linguagem de padrões GRN oferece a desenvolvedores inexperientes, material substancial para desenvolvimento de novos sistemas, juntamente com soluções alternativas, quando necessário. 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. Neste trabalho, a palavra “transação” possui o mesmo significado definido por Coad (Coad, 1992; Coad et al., 1997): um

evento significativo que precisa ser lembrado pelo sistema através 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 3.4 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 reforçada. 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.

A notação utilizada para expressar os padrões é a UML (do inglês *Unified Modelling Language*) (Eriksson, 1998; Rational, 2000). Para cada padrão é incluído um exemplo de sua instanciação, na qual novos atributos podem ser adicionados de acordo com a aplicação específica. Métodos básicos para criar um objeto, atribuir e recuperar valores de atributos, adicionar e remover conexões de objetos, buscar um objeto específico e excluir objetos não são mostrados nos modelos de classes, já que somariam mais complexidade a esses diagramas sem trazer ganhos na efetividade do modelo (Coad et al., 1997; Larman, 1997). Ao invés disso, assume-se que esses métodos estejam presentes em cada classe, sempre que for apropriado. Quanto à formatação do texto do padrão, o NOME DO PADRÃO e de outros padrões mencionados no texto é escrito utilizando capitalização de maiúsculas; cada Elemento do padrão é sublinhado; nomes de métodos, classes e atributos são escritos em fonte Courier.

De acordo com o padrão Controller (Larman, 1997), operações do sistema devem ser atribuídas a uma das seguintes classes: i) aquela que representa o sistema como um todo; ii) aquela que representa o negócio ou organização como um todo; iii) aquela que representa alguma coisa ativa do mundo real e que possa estar envolvida com a tarefa; ou iv) aquela que representa um tratador abstrato de todos os eventos do sistema em um caso de uso. Nesta tese escolheu-se a terceira alternativa, isto é, as operações do sistema foram atribuídas à classe cuja funcionalidade fosse mais

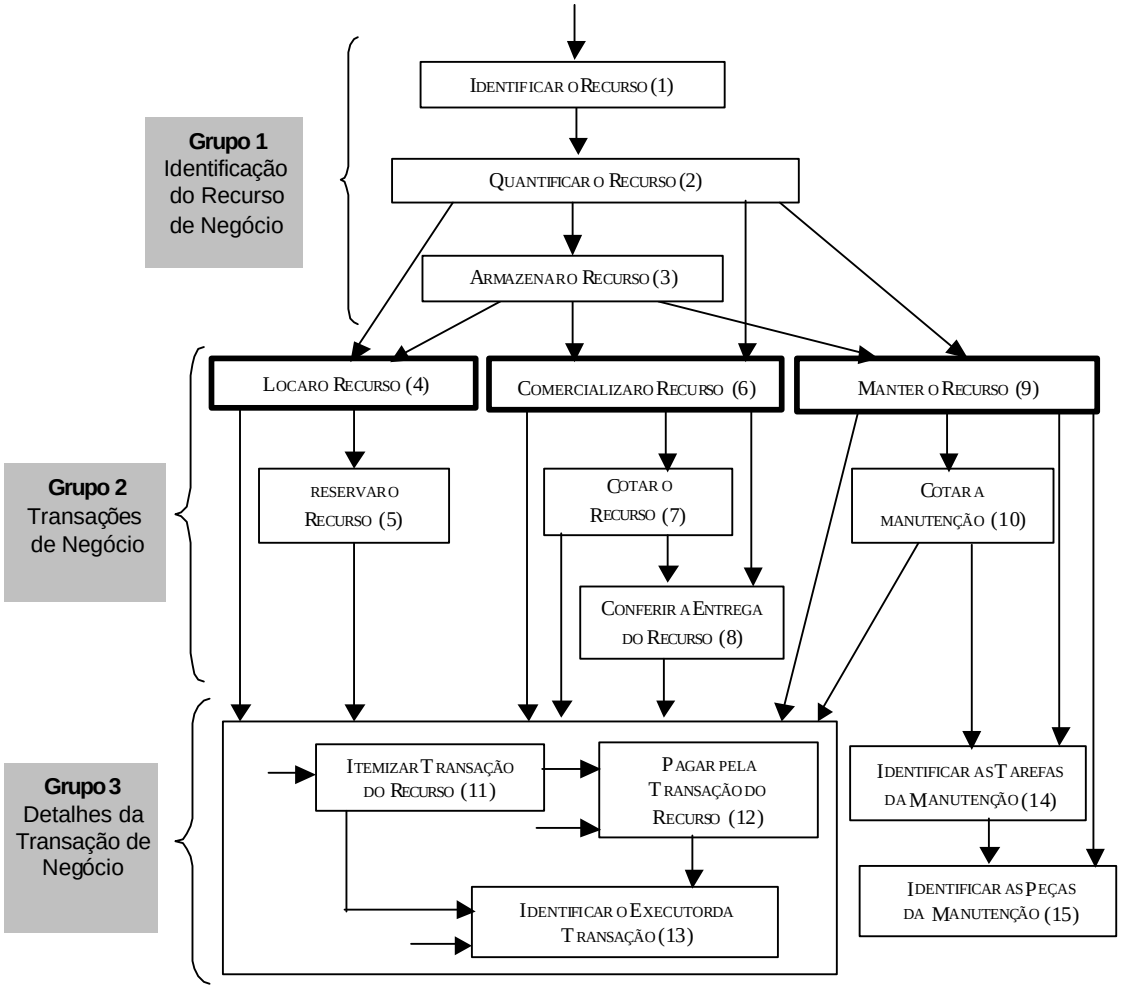


Figura 3.4: GRN: Uma Linguagem de Padrões para Gestão de Recursos de Negócios

próxima da funcionalidade da operação. Coleman et al. (1994) dão a essa classe a denominação "controladora da operação". Outros autores, como Coad e Fowler, não incluem operações em seus diagramas de classes, representando apenas os métodos (Coad et al., 1997; Fowler, 1997). Na verdade, as operações do sistema são mais que métodos, já que são executadas em resposta a eventos do sistema que ocorrem no mundo real. Sua funcionalidade é implementada por meio da chamada de diversos métodos, possivelmente de diversas classes diferentes. Neste trabalho é feita distinção entre dois tipos de operações por meio do uso de um marcador especial como prefixo. É utilizado o prefixo "?" para operações de entrada que modificam o estado interno do sistema e o prefixo "!" para operações de saída, as quais geram saídas do sistema, sem contudo modificar seu estado. Além disso, o prefixo "*" é utilizado em nomes de métodos quando a mensagem invocada é enviada para uma coleção de objetos em vez de para uma instância única. Evitou-se adicionar uma

classe recipiente (do inglês *container*) para lidar com essa situação, porque tal adição anteciparia para a fase de análise questões de projeto e implementação .

Os padrões estão agrupados de acordo com seu propósito, como pode ser visto na Figura 3.4. No grupo 1 são incluídos os padrões que tratam da identificação e possível qualificação, quantificação e armazenagem dos recursos de negócios, no grupo 2 os padrões que lidam com as transações efetuadas pelo sistema e no grupo 3 os padrões que cuidam de detalhes associados à maioria das transações de negócio. A GRN pode ser facilmente encontrada na Web (Braga et al., 2002) e a seção seguinte ilustra um de seus padrões.

3.5.2 Exemplo de um Padrão da GRN

Padrão 2: QUANTIFICAR O RECURSO

Contexto

Você identificou o recurso gerenciado por sua aplicação e suas qualidades relevantes. Uma questão importante a ser considerada agora é a forma de quantificação do recurso. Existem certas aplicações nas quais é necessário ter controle sobre instâncias específicas do recurso, porque elas são negociadas individualmente. Por exemplo, em uma biblioteca um livro pode possuir diversas cópias, cada qual emprestada a um leitor diferente. Outras aplicações lidam com uma certa quantidade do recurso ou com lotes de recursos. Em tais aplicações, não é necessário saber que instância particular do recurso foi negociada. Por exemplo, uma certa quantidade de aço foi vendida. Em outras aplicações, o recurso é único, como por exemplo um carro que sofre manutenção ou um médico que examina um paciente.

Problema

Como a aplicação quantifica o recurso de negócio?

Influências

- É muito importante saber, já na fase de análise do sistema, exatamente qual a forma de quantificação do recurso adotada. Uma decisão errada nesse ponto pode comprometer a evolução futura.
- Se é necessário controlar instâncias específicas do recurso, então informações redundantes podem ser armazenadas para as diversas instâncias do mesmo recurso. Porém, essa redundância é indesejável.
- Para evitar redundância, uma nova classe pode ser criada, na qual as informações comuns a todas as instâncias do mesmo recurso podem ser armazenadas apenas uma vez. Mas um preço tem que ser pago por lidar com duas classes ao invés de uma: por exemplo, o tempo de processamento será provavelmente maior.

Estrutura

Existem quatro soluções ligeiramente diferentes para esse problema, dependendo da forma de quantificação. A Figura 3.5 mostra os quatro sub-padrões do padrão QUANTIFICAR O RECURSO. Quando o recurso é único, utilize o sub-padrão RECURSO SIMPLES (Figura 3.5 (a)). Quando o recurso é tratado em quantidade específicas, utilize o sub-padrão RECURSO MENSURÁVEL (Figura 3.5 (b)). Quando é importante distinguir entre instâncias do recursos, utilize o sub-PADRÃO RECURSO INSTANCIÁVEL (Figura 3.5 (c)). Quando o recurso é tratado em lotes, utilize o sub-padrão RECURSO EM LOTES (Figura 3.5 (d)).

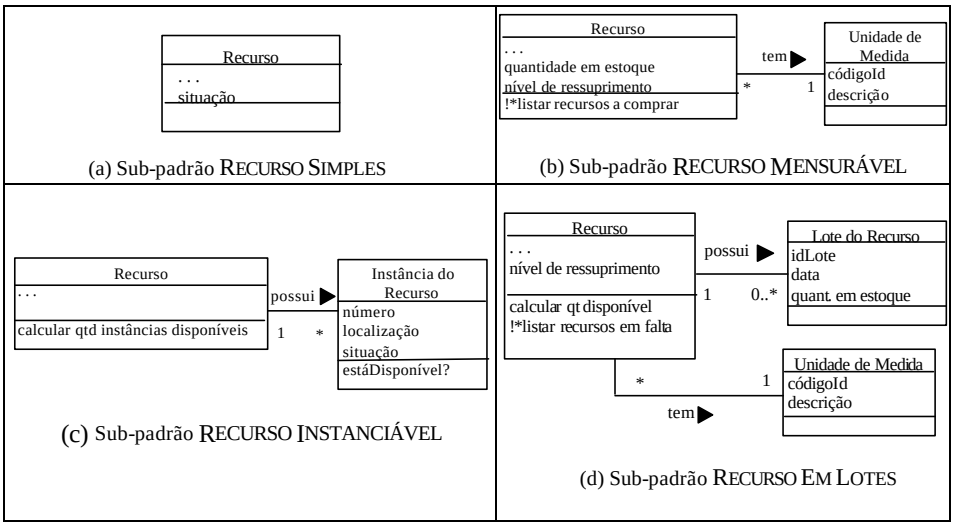


Figura 3.5: Padrão 2 - Quantificar o Recurso

Participantes

- `Recurso`: como já descrito no padrão 1, a menos de alguns novos atributos e métodos, incluídos de acordo com o sub-padrão no qual aparece. Por exemplo, quando o sub-padrão RECURSO SIMPLES é utilizado, o atributo `situação` controla o ciclo de vida do recurso. Em uma oficina de veículos, a situação do veículo poderia ser: "pronto", "aguardando conserto" ou "em conserto". Outro exemplo é a inclusão dos atributos quantidade em estoque e nível de reabastecimento para lidar com o controle de estoque quando se utiliza o sub-padrão RECURSO MENSURÁVEL. Nesse caso, o atributo `situação` não se aplica, pois o sistema lida com quantidades grandes do recurso de uma vez e, portanto, não pode controlar o ciclo vida do recurso individualmente.

- `Instância do Recurso`: representa cada exemplar ou cópia de um recurso de negócio. O atributo `situação` controla o ciclo de vida de cada recurso individualmente; por exem-

plo, durante o ciclo de vida de uma cópia do livro há quatro tipos diferentes de situação: “disponível”, “apenas reservado”, “apenas emprestado” e “reservado e emprestado”.

- `Lote do Recurso`: representa cada lote individual de recursos de negócios, em geral composto de uma quantidade específica de recursos. Em alguns casos é importante adicionar um atributo `data de vencimento` para controlar os recursos a serem utilizados primeiramente ou a serem descartados por vencimento de validade.

- `Unidade de Medida`: representa todas as possíveis unidades de medida pelas quais os recursos de negócio podem ser medidos, como por exemplo gramas, quilogramas, pacotes, etc.

Exemplo

A Figura 3.6 mostra instanciações dos quatro sub-padrões do padrão QUANTIFICAR O RECURSO.

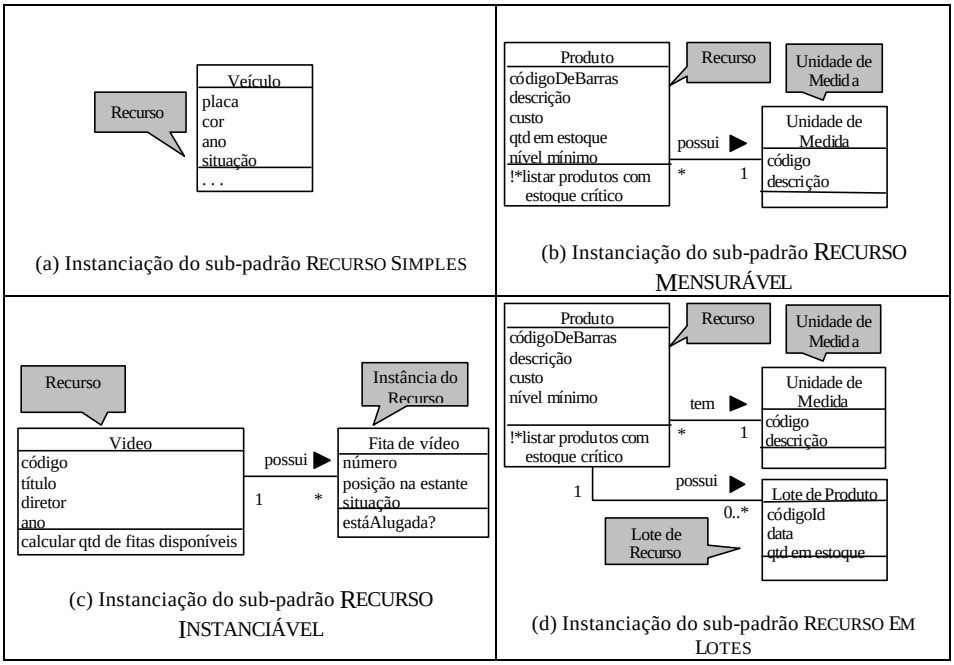


Figura 3.6: Exemplo de uso do Padrão 2

Próximos padrões

Depois de QUANTIFICAR O RECURSO (2), examine sua aplicação para verificar se é importante manter informações sobre o armazenamento dos recursos. Em caso positivo, aplique ARMAZENAR O RECURSO (3). Em caso negativo, continue examinando sua aplicação para verificar que tipos de transação são efetuadas. Se a aplicação estiver relacionada com a locação ou

aluguel de recursos, você deve LOCAR O RECURSO (4). Se a aplicação referir-se ao comércio de recursos, por exemplo compra ou venda, você deve COMERCIALIZAR O RECURSO (6). Se a aplicação lidar com conserto de recursos, você deve MANTER O RECURSO (9). Observe, entretanto, que existem aplicações nas quais vários desses padrões podem ser aplicados. Por exemplo, em um sistema de locação de veículos, além da reserva e aluguel dos veículos, pode-se efetuar o controle de aquisição, manutenção e venda dos veículos. Outros tipos de transação, não previstas pela linguagem de padrões GRN, podem ser efetuadas com recursos de negócios. Por exemplo, o leilão de recursos é uma transação na qual uma administradora de leilões coloca à venda produtos pertencentes a um proprietário, que podem ser adquiridos por um comprador. Uma extensão da linguagem GRN está sendo feita para incorporar o leilão de recursos.

3.5.3 O processo de construção da GRN

A linguagem de padrões GRN foi construída em duas etapas: primeiramente, com base na experiência prática da autora no desenvolvimento de sistemas no domínio da GRN (primeiro passo do processo da seção 3.3), os diagramas de classes de alguns sistemas existentes foram analisados e foram detectados três padrões principais: Locar o Recurso, Comercializar o Recurso e Manter o Recurso. São padrões razoavelmente grandes, contendo, cada um deles, entre sete e nove classes. Tais padrões foram escritos usando o formato “Problema/Contexto/Forças/Solução/Exemplo” e submetidos a uma sessão da Conferência PLoP de 1998 (Braga et al., 1998). Durante a sessão, autores mais experientes com a escrita de padrões observaram que os padrões eram muito grandes e que havia partes comuns entre os três padrões. A sugestão recebida foi de sub-dividir os padrões em padrões menores, verificando as partes comuns aos três padrões, e finalmente formando uma linguagem de padrões para o domínio de gestão de recursos de negócios. Devido à inexperiência da autora na escrita de padrões, princípios básicos não foram seguidos, como os citados no segundo passo do processo da seção 3.3. Isso resultou em padrões difíceis de reutilizar e com partes repetidas (redundância). Entretanto, a experiência adquirida possibilitou a definição de um processo para ajudar outros autores na mesma situação.

Para construir a Linguagem de Padrões GRN a partir dos três padrões citados, uma análise foi feita nesses padrões e foram isoladas as funcionalidades básicas de cada classe ou pequeno grupo de classes. Isso permitiu que nove padrões pudessem ser definidos a partir dos três. Depois, analisando outras funcionalidades do domínio foram propostos mais cinco padrões, totalizando quatorze padrões que formaram a linguagem de padrões submetida à Conferência PLoP de 1999 (Braga et al., 1999). A interação entre esses padrões foi representada por meio de um grafo, como pode ser visto na Figura 3.4, e documentada de forma textual na seção “Próximo Padrões” de cada um dos padrões, conforme exemplificado na seção 3.5.2.

A linguagem de padrões GRN continuou evoluindo após sua publicação inicial, com a inclusão de outros aspectos não considerados em sua primeira versão, como por exemplo um melhor tratamento ao pagamento das transações de recursos de negócios e armazenamento dos recursos. A versão disponível atualmente na Web[2] possui quinze padrões, alguns deles tendo sido estendidos com variantes apresentando soluções para o mesmo problema sob forças e contexto diferentes. Existe previsão de continuar a evolução da GRN em trabalhos futuros, conforme pode ser visto na seção 7.5.

3.5.4 O Processo de Uso da GRN

O processo de uso de uma linguagem de padrões, apresentado na Seção 3.4, pode ser especializado para a linguagem de padrões GRN, dando origem ao seguinte processo:

1. Estude a linguagem de padrões GRN detalhadamente, de forma a conhecer o domínio por ela tratado, que é o de gestão de recursos de negócios, seus quinze padrões constituintes, os variantes e sub-padrões e quando e como aplicá-los. Estude também o exemplo contido na linguagem, que é de um sistema para aluguel e venda de produtos para festas, que dá uma boa idéia de como aplicá-la.

2. Produza o documento de requisitos do sistema específico, conforme as recomendações da Engenharia de Software.

3. Leia com atenção o documento produzido no passo 2, para ter um conhecimento amplo dos requisitos do sistema, necessário para decidir se a linguagem de padrões GRN pode ser utilizada na modelagem de tal sistema.

4. Use a linguagem de padrões GRN para modelar o sistema específico, tomando como base as informações presentes nas diversas seções de cada padrão para decidir sobre a conveniência de utilizá-lo. Os padrões da linguagem GRN possuem os seguintes elementos: contexto, problema, influências, estrutura, participantes, exemplo, variantes (opcional) e próximos padrões. Todos eles são importantes para a aplicação da linguagem, que é auto-explicativa. Conseqüentemente, não há necessidade de explicar detalhadamente como aplicar cada um dos padrões individualmente.

5. Para cada padrão aplicável esboce o diagrama de classes referente à porção do sistema que utilizará tal padrão, destacando (por exemplo, com cor diferente) possíveis atributos, métodos ou operações incluídos. Esse diagrama de classes cresce de maneira gradual, na medida em que novos padrões são aplicados.

[2] `http://www.icmc.usp.br/~rtvb/linguagem_padroes_grn.zip` (versão atualizada em 20 de agosto de 2002)

6. Para cada padrão aplicável faça uma marcação no documento de requisitos para indicar quais dos requisitos foram atendidos pela aplicação de tal padrão.

7. Para cada padrão aplicável utilize balões para indicar os papéis desempenhados por cada classe em tal padrão.

8. Para cada padrão aplicável anote o nome do padrão utilizado e seu variante ou sub-padrão (se houver).

9. O último elemento de cada padrão, denominado ”próximos padrões”, contém as instruções dos possíveis padrões a investigar após ter aplicado (ou não) o padrão atual. Isso define diversos caminhos a seguir, que devem ser anotados pelo desenvolvedor e investigados um a um.

10. Confira o documento de requisitos em busca de requisitos não atendidos ou atendidos parcialmente pela linguagem de padrões GRN. Complemente o diagrama de classes com tais requisitos, por meio da adição de novos atributos, classes, relacionamentos, métodos ou operações, usando cor diferente e fazendo uma anotação no documento de requisitos que identifique a não cobertura do requisito pela linguagem de padrões.

3.5.5 Processo detalhado para Aplicação da GRN

Ainda que a linguagem de padrões GRN seja auto-explicativa, usuários inexperientes podem beneficiar-se de um processo que mostre de forma minuciosa como aplicá-la para modelar um sistema específico, como mostrado a seguir:

1. A partir do documento de requisitos, verifique quais são os candidatos a Recurso. As características a serem observadas para obter os possíveis candidatos encontram-se nos elementos “contexto”, “problema” e “influências” do primeiro padrão da linguagem GRN, que é IDENTIFICAR O RECURSO. O resultado será a enumeração de todos os candidatos a Recurso.

2. Conforme pode-se observar no contexto desse primeiro padrão, deverão ser descartados candidatos a Recurso que não estejam envolvidos em transações gerenciadas pelo sistema, tais como: pedido, compra, venda, locação, aluguel, reserva, conserto ou manutenção. Após descartar os falsos candidatos, o resultado será uma lista de recursos gerenciados pelo sistema.

3. Para cada um dos recursos identificados no passo 2, analise a estrutura do padrão IDENTIFICAR O RECURSO, seus participantes e variantes. Com base nessa análise, esquematize

o diagrama de classes resultante, podendo acrescentar novos atributos às classes participantes (use uma cor diferente para destacá-los). O resultado será um diagrama de classes para cada um dos recursos, contendo a classe que representa o recurso e uma ou mais classes que representam as possíveis categorias do recurso, dependendo do variante do padrão utilizado.

4. Repita este passo seqüencialmente, a partir do passo 4a até o passo 4e, para cada um dos recursos definidos no passo 3, percorrendo a linguagem GRN a partir do padrão 2, que é QUANTIFICAR O RECURSO.

   (a) Verifique se o contexto exposto no padrão é compatível com o encontrado no sistema a ser desenvolvido e se o problema tratado pelo padrão faz parte dos requisitos desse sistema. Caso não seja, tal padrão provavelmente não se aplica, vá para o passo 4g.

   (b) Analise os aspectos abordados nas influências do padrão. Esses aspectos devem ser importantes para a solução do problema em mãos. Caso positivo, vá para o passo 4c. Caso contrário, esse padrão provavelmente não se aplica, mas pode haver, nas influências, algumas sugestões sobre como resolver o problema sem usar o padrão. Siga essas sugestões, se for o caso.

   (c) Analise a estrutura da solução proposta pelo padrão. Estude cada um dos participantes da solução, tentando mapeá-lo para objetos do sistema a ser desenvolvido. Alguns dos participantes podem ser opcionais, portanto analise cuidadosamente a conveniência de utilizá-los ou não. Caso encontre dificuldade em executar esse mapeamento, estude com atenção o exemplo do padrão, que pode proporcionar melhor compreensão do padrão e de sua aplicação.

   (d) Faça um diagrama de classes para esquematizar a aplicação do padrão, colocando as classes, os relacionamentos e os papéis desempenhados por cada classe no padrão enfocado. Complemente as classes com os demais atributos não presentes no padrão. Para facilitar a identificação dos atributos adicionados, dê um destaque especial a eles (por exemplo, use caneta de cor diferente).

   (e) Se não estiver satisfeito com a solução, estude a possibilidade de utilizar um dos variantes do padrão, se existir.

   (f) Se ainda assim não estiver satisfeito com a solução, siga as recomendações presentes em próximos padrões, aplicando sempre os passos 4a a 4e para cada um dos padrões seguintes.

   (g) Caso decida pela não aplicação de um determinado padrão, tente percorrer outros caminhos deixados em aberto anteriormente. Pode haver diversos caminhos a percorrer antes de chegar a uma boa solução ou decidir descartar o padrão ou até mesmo a linguagem de padrões. Assegure-se de que todos os caminhos foram seguidos.

5. Quando se esgotarem todos os recursos definidos no passo 3 e todos os caminhos tiverem sido percorridos, a aplicação da linguagem de padrões terá terminado. Pode-se então prosseguir com a checagem do documento de requisitos em busca de requisitos não atendidos pela linguagem de padrões GRN.

3.5.6 Exemplo de Aplicação da GRN

Apresenta-se nesta sub-seção um exemplo de aplicação da linguagem de padrões GRN na modelagem de uma Sistema de Acompanhamento e Reparo de Buracos (SARB), descrito por Pressman (2002) (página 326) e reproduzido a seguir.

Descrição dos requisitos principais do SARB

"Os cidadãos podem obter acesso a um site da Web e relatar a localização e gravidade dos buracos. À medida que os buracos são relatados eles são registrados num "sistema de reparo do departamento de obras públicas" e lhes é atribuído um número de identificação, armazenado por endereço da rua, tamanho (numa escala de 1 a 10), localização (no meio da rua, na calçada, etc.), distrito (determinado pelo endereço da rua) e prioridade de reparo (determinada pelo tamanho do buraco). Dados da ordem de serviço são associados com cada buraco e incluem a localização e tamanho do buraco, número de identificação da equipe de reparo, número de pessoas na equipe, equipamento atribuído, horas aplicadas no reparo, estado do buraco (trabalho em andamento, reparado, reparo temporário, não reparado), quantidade de material de enchimento usado e custo do reparo (calculado a partir de horas aplicadas, quantidade de pessoas, material e equipamento usados). Finalmente, um arquivo de danos é criado para conter informação sobre danos relatados devido ao buraco e incluem nome do cidadão, endereço, número do telefone, tipo de dano e quantia em reais de prejuízo causado pelo dano. O SARB é um sistema on-line; todas as consultas devem ser feitas interativamente".

Aplicação da GRN para modelagem do SARB

Com base nos requisitos do SARB, a GRN foi utilizada com o objetivo de modelar o sistema, produzindo um modelo de classes com a indicação dos papéis desempenhados por cada classe nos padrões utilizados. Além disso, produz-se um histórico dos padrões e variantes utilizados, bem como uma lista de decisões de análise tomadas quando a linguagem de padrões não atende aos requisitos do sistema.

Conforme recomendado no processo apresentado na Seção 3.4, a linguagem de padrões GRN deve ser estudada previamente pelo desenvolvedor e os requisitos do sistema devem ser obtidos usando-se quaisquer técnicas da Engenharia de Software. No caso do SARB, consideram-se os

requisitos contidos no texto apresentado nesta seção. Dois sub-sistemas podem ser identificados nos requisitos: o reparo do buraco e o controle de danos causados aos cidadãos. Portanto a GRN deve ser aplicada em dois ciclos, cada um deles para um dos sub-sistemas. O modelo de análise final produzido aplicando-se a GRN ao SARB é mostrado na Figura 3.7. Balões indicam os padrões da GRN usados para modelar o SARB, sendo que uma mesma classe pode desempenhar papéis diferentes em mais do que um padrão. O formato utilizado no interior dos balões é "P#n: papel", onde "n" é o número do padrão e "papel" é o papel desempenhado pela classe em tal padrão. Deve-se salientar que métodos e operações canônicas não são incluídas.

O modelo da Figura 3.7 foi obtido de forma gradual, à medida que os padrões foram aplicados. Iniciando-se pelo sistema de reparo de buracos, o primeiro padrão da GRN — IDENTIFICAR O RECURSO — é aplicado, sendo que o *Buraco* é identificado como sendo o *Recurso*. As possíveis classificações do buraco, por exemplo, com relação ao distrito, ao tamanho e à localização, levam à aplicação do variante "Múltiplos Tipos de Recursos" desse padrão. Aplica-se então o padrão 2 – QUANTIFICAR O RECURSO, no qual o buraco pode ser caracterizado como sendo simples, pois possui características que levam a gerenciá-lo individualmente (outras opções, tais como recursos medidos, instanciados ou tratados em lotes, são inadequadas nesse exemplo).

Após identificado e quantificado o recurso, inicia-se a modelagem da gestão do recurso. Nesse caso em particular, o tipo de gestão desejado para o buraco é o reparo, que encaixa-se no padrão 9 – MANTER O RECURSO. Analisando-se detalhadamente esse padrão, percebe-se que um dos participantes opcionais — *Origem* — pode ser removido, já que não é importante para o sistema que sejam registradas unidades específicas do Departamento de Obras, mas apenas o próprio departamento. Assim, o papel de *Destino* é atribuído ao *Departamento de Obras Públicas*, que, na verdade, desempenha também o papel de *Origem*, porque é ele quem requisita um reparo de buraco e, depois, realiza tal reparo.

Finalmente aplicam-se os padrões para complementar detalhes da transação de reparo do buraco, tais como: o padrão 13 – IDENTIFICAR O EXECUTOR DA TRANSAÇÃO, no qual pode-se lidar com a equipe de reparo responsável pelo buraco; o padrão 14 – IDENTIFICAR AS TAREFAS DA MANUTENÇÃO, que permite a discriminação de cada uma das tarefas necessárias para o reparo do buraco; e o padrão 15 – IDENTIFICAR AS PEÇAS DA MANUTENÇÃO, que permite registrar cada um dos materiais consumidos no reparo.

Durante a análise do sistema usando a GRN, faz-se necessário, conforme recomendado no passo 4(c) do processo proposto na seção 3.5.5, fazer o mapeamento entre os atributos, métodos e operações do padrão *versus* os atributos, métodos e operações das classes concretas do sistema sendo modelado. No caso do SARB, novos atributos foram adicionados, como por exemplo o atributo `numeroDePessoasEquipe` (classe *Ordem de Serviço*).

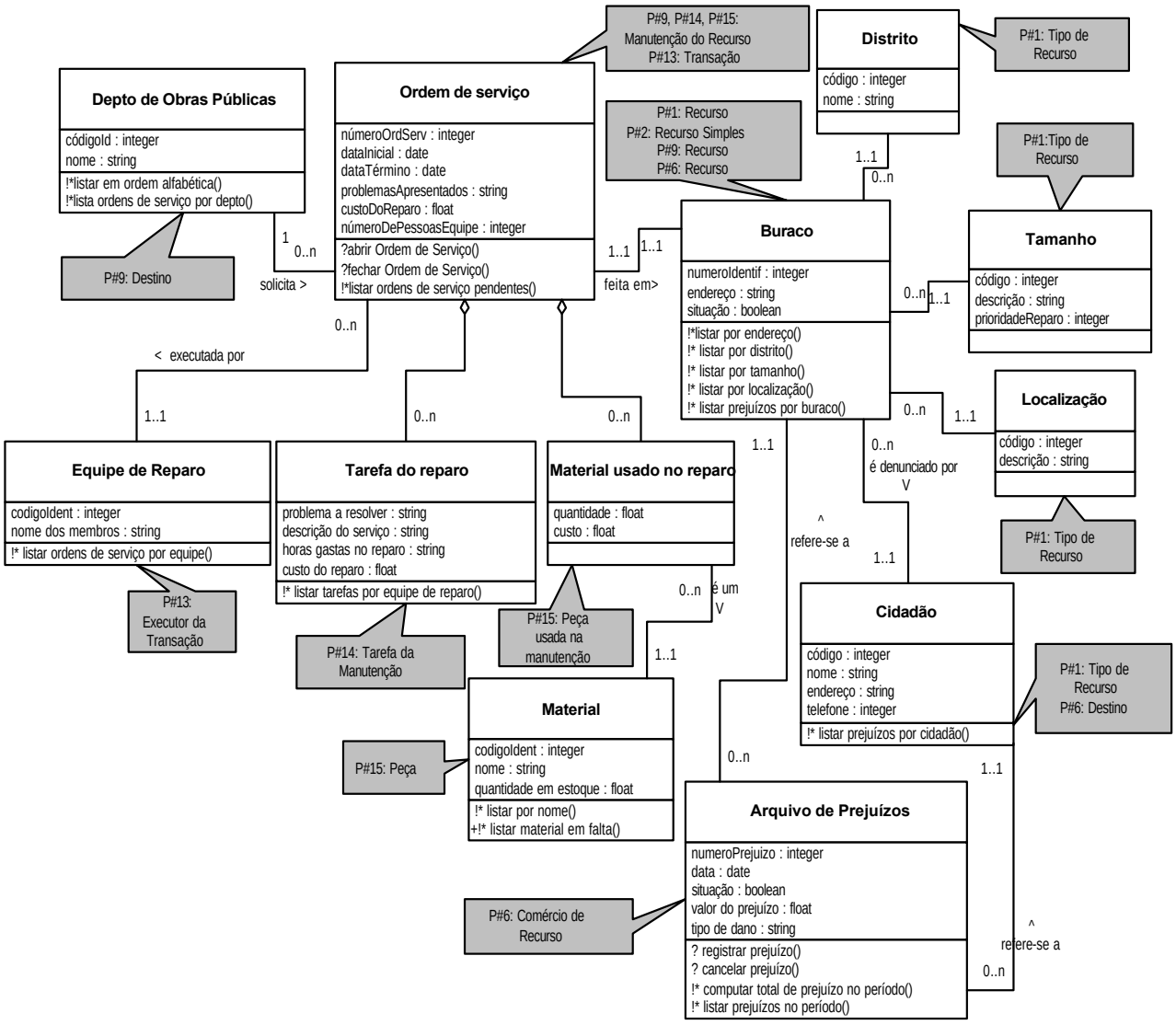


Figura 3.7: Modelo de Análise do SARB com padrões

Tendo modelado o sub-sistema de reparos, parte-se agora para o sub-sistema de controle dos prejuízos causados pelos buracos. Analisando-se os aspectos semânticos da GRN, pode-se concluir que esse sub-sistema refere-se a um tipo de transação não coberta pela GRN: o controle do prejuízo não se encaixa em quaisquer das três principais atividades abordadas na GRN: não se trata de uma locação, na qual um recurso muda temporariamente de proprietário, não se trata tampouco de uma comercialização, em que a posse do recurso é transferida para outro proprietário, e também não se trata de uma manutenção, na qual um recurso com problemas passa por serviços e trocas de peças para voltar a funcionar.

Entretanto, se analisarmos apenas a sintaxe dos padrões da GRN e compararmos os atributos envolvidos no controle dos prejuízos causados pelos buracos, vemos que a classe *Comercialização*

do Recurso, do padrão 6 – COMERCIALIZAR O RECURSO, possui semelhanças que a tornam atrativa para ser reutilizada na modelagem desse sub-sistema. Feitas as devidas adaptações, o padrão 6 é então utilizado para concluir a etapa de análise do SARB. Esse tipo de uso de um padrão é denominado "abstração do domínio", ou, em inglês, "*flexing*" por Butler et al. (2000). A pré-condição para fazer a abstração de domínio é que o padrão possua a estrutura computacional requerida, embora não carregue a semântica do domínio antecipada por seu projetista. Neste trabalho, chamamos isso de "processo de substituição semântica/sintática", e usamos, na Tabela 3.1, a referência geral ao padrão (USAR O RECURSO).

O resultado da modelagem do SARB usando a GRN é o modelo de classes mostrado na Figura 3.7 e o histórico de padrões usados na modelagem (Tabela 3.1). Outro possível resultado da modelagem poderia ter sido uma lista das decisões de análise tomadas durante o processo, na qual estariam discriminados os requisitos não atendidos pela GRN e como tais requisitos foram modelados (por meio de classes adicionais, relacionamentos, etc.). Essa lista de decisões seria importante durante a implementação da aplicação específica e em sua futura manutenção. Ela não foi necessária no caso do SARB, já que a GRN apoiou a modelagem de todos os seus requisitos.

Tabela 3.1: Histórico de padrões usados na instanciação do SARB

Padrão	**Variante**	**Participante**	**Classe da Aplicação**
1-Identificar o Recurso	Múltiplos tipos	Recurso	Buraco
		Tipo de Recurso	Distrito
		Tipo de Recurso	Tamanho
		Tipo de Recurso	Localização
		Tipo de Recurso	Cidadão
2 - Quantificar o Recurso	Recurso Simples	Recurso	Buraco
9 - Manter o Recurso	Sem origem	Recurso	Buraco
		Manutenção do Recurso	Ordem de Serviço
		Destino	Departamento de Obras Públicas
13 - Identificar o Executor da Transação	Sem comissão	Executor da Transação	Equipe de Reparo
		Transação	Ordem de Serviço
14 - Identificar as Tarefas da Manutenção	Executor da Transação ao invés de executor da tarefa	Manutenção do Recurso	Ordem de Serviço
		Tarefa da Manutenção	Tarefa do Reparo
15 - Identificar as Peças da Manutenção	Default	Manutenção do Recurso	Ordem de Serviço
		Peça usada na Manutenção	Material usado no reparo
		Peça	Material
6 - Comercializar/Usar o Recurso	Sem origem	Recurso	Buraco
		Comércio do Recurso	Arquivo de Prejuízos
		Destino	Cidadão

3.6 Considerações Finais

Este Capítulo apresentou uma visão geral do processo proposto nesta tese, possibilitando a compreensão do escopo do trabalho. Descreveu em detalhes o primeiro passo desse processo, que trata da construção de uma linguagem de padrões para um domínio específico. Como exemplo, apre-

sentou a linguagem de padrões GRN, que continuará sendo utilizada nos demais capítulos para ilustrar o processo proposto.

Em síntese, pode-se dizer que o processo geral proposto atende à necessidade de novos métodos para desenvolvimento e instanciação de frameworks que, embora sendo uma tecnologia poderosa de reuso, tem seu uso inibido tanto pela complexidade de desenvolvimento quanto pela dificuldade de instanciação.

A linguagem de padrões GRN também contribui para o reuso, porém num nível de abstração mais alto: seus padrões proporcionam a desenvolvedores inexperientes um caminho a seguir quando precisam modelar sistemas no domínio de gestão de recursos de negócios. O resultado do uso da GRN é um diagrama de classes para uma aplicação específica, composto de diversas partes que representam as soluções para problemas comuns no domínio. Informações sobre o comportamento dinâmico do sistema também poderiam ser fornecidas pela GRN, por exemplo por meio de diagramas de seqüência da UML, o que ajudaria o usuário inexperiente a entender mais rapidamente o domínio tratado. Porém, a versão atual da GRN mostra apenas o comportamento estático dos padrões, ficando como sugestão para trabalho futuro a inclusão de modelos dinâmicos.

O próximo capítulo descreve em detalhes mais dois passos do processo proposto: a construção do framework caixa branca (passo 2) e a instanciação desse framework para aplicações específicas, tendo como base apenas o framework caixa branca (passo 4a).

CAPÍTULO

4

O Processo de Construção e Instanciação de um Framework Caixa Branca

4.1 Considerações Iniciais

Neste Capítulo, detalham-se os processos de construção e instanciação de um framework caixa branca com base em uma linguagem de padrões para um domínio específico. Esses dois processos referem-se aos passos 2 e 4a do processo geral da Figura 8.1 e foram agrupados em um mesmo capítulo por serem estritamente relacionados: o framework caixa branca construído no passo 2 é instanciado utilizando-se o processo mostrado no passo 4a. O processo de construção e instanciação do framework GREN é utilizado para ilustrar os processos gerais apresentados neste Capítulo.

O Capítulo está organizado da seguinte forma: na seção 4.2 descreve-se o processo de construção de um framework caixa branca com base em uma linguagem de padrões, exemplificado na seção 4.3 pelo processo de construção do framework GREN. Na seção 4.4 é detalhado o processo de instanciação do framework, usando como base a linguagem de padrões com a qual ele foi construído. O exemplo de instanciação do framework GREN, usando a linguagem de padrões

GRN, mostrado na seção 4.5, ilustra tal processo. Na seção 4.6 são apresentadas as conclusões relativas aos dois processos propostos.

4.2 O Processo de Construção

O processo geral para construção de um framework baseado em uma linguagem de padrões (Braga e Masiero, 2001b, 2002d) é ilustrado pela Figura 8.3. Esse processo é constituído de quatro passos, detalhados nas sub-seções seguintes. Deve-se salientar que a seqüência de passos é flexível, podendo-se iterar quantas vezes for necessário para obtenção do framework final. Isso ocorre principalmente após a validação do framework, durante a qual podem ser identificados novos pontos variáveis que precisam ser projetados e implementados.

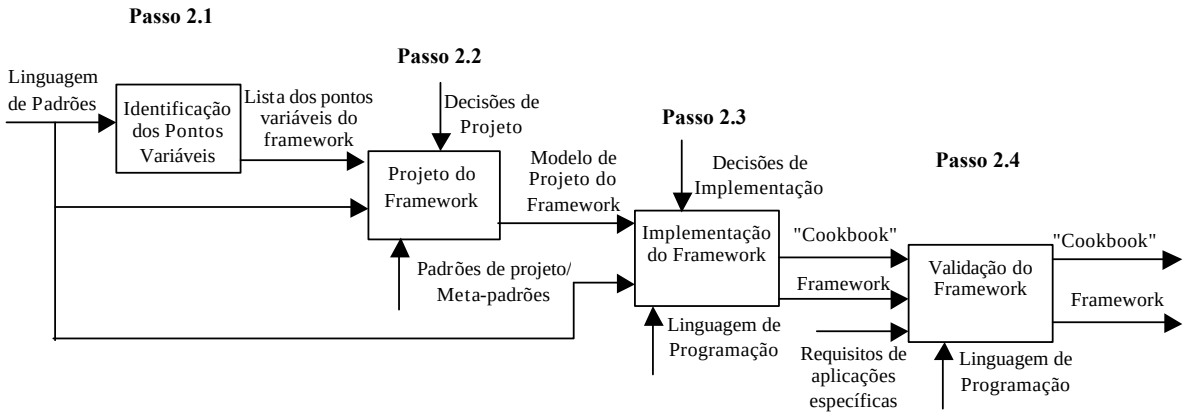


Figura 4.1: Processo para construção de um framework baseado em uma linguagem de padrões

4.2.1 Identificação dos Pontos Variáveis

A primeira atividade para desenvolvimento de um framework é a identificação de seus pontos variáveis (Braga e Masiero, 2001a), isto é, dos pontos nos quais requer-se flexibilidade no software, já que eles variam de aplicação para aplicação do domínio. O primeiro passo do processo proposto resultou em uma linguagem de padrões para o domínio. O passo 2.1 da Figura 8.3, que consiste na identificação dos pontos variáveis do framework, é aplicado basicamente utilizando informações presentes nessa linguagem de padrões. Esta atividade é composta de cinco sub-passos, conforme ilustrado na Figura 8.4.

O primeiro sub-passo consiste da análise do grafo da linguagem de padrões (se existir) ou, alternativamente, das seções “Próximos Padrões”. O objetivo desta análise é encontrar pontos variáveis decorrentes de padrões opcionais da linguagem, isto é, padrões que não são usados por

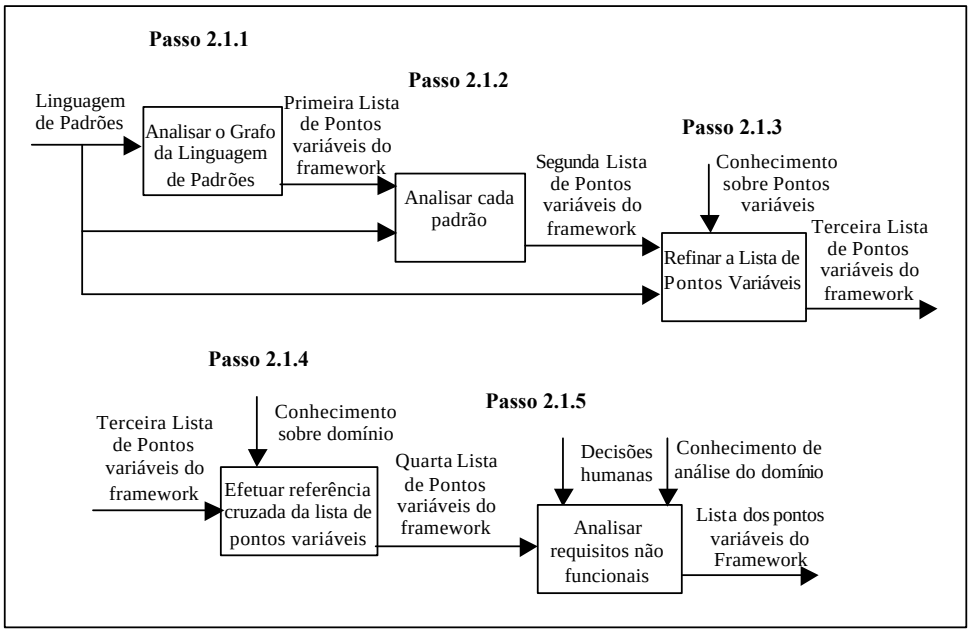


Figura 4.2: Identificação dos pontos variáveis do framework

todas as aplicações pertencentes ao domínio, mas que podem ser opcionalmente utilizados. O grafo da linguagem de padrões indica a interação entre os padrões e a ordem na qual eles podem ser aplicados. A seção "Próximos Padrões", comumente utilizada nos padrões de uma linguagem de padrões, indica os caminhos a seguir após a aplicação (ou não) do padrão atual. Portanto, esses são locais propícios para encontrar pontos variáveis do framework a ser construído, uma vez que, muitas vezes, partes da funcionalidade do domínio estão associadas a um ou mais padrões opcionais. Assim, essa análise deve considerar os possíveis caminhos a percorrer durante a aplicação da linguagem de padrões. Caminhos que pulem um ou mais padrões devem ser analisados, pois o framework deverá permitir que o sistema funcione corretamente mesmo sem a aplicação de tais padrões.

No segundo sub-passo deve ser feita a análise individual de cada padrão, examinando-se cada uma de suas seções constituintes, já que elas podem indicar diversos pontos variáveis do framework. Na seção "Variantes" ou na seção "Sub-padrões", poderão ser encontradas soluções alternativas para o problema resolvido pelo padrão. Além disso, pode haver indicações de aspectos variáveis que deveriam estar disponíveis ao usuário do framework. Por exemplo, na Figura 3.5 existem diversas soluções alternativas para o problema de quantificação do recurso de negócios, e tais soluções são mostradas por meio de diversas soluções ou sub-padrões, tais como RECURSO INSTANCIÁVEL e RECURSO MENSURÁVEL.

As seções "Participantes" e "Colaborações", presentes em padrões que seguem o formato GoF (Gamma et al., 1995), também podem indicar pontos variáveis do framework, por exemplo, um

participante opcional é uma clara indicação de um ponto variável. A seção "Implementação" contém sugestões de implementações alternativas para a solução proposta, de forma que, de acordo com as restrições impostas por cada aplicação em particular, diferentes implementações possam ser escolhidas. Outra fonte de pontos variáveis é a seção "Estrutura", a qual contém uma representação diagramática das classes do padrão e de seus relacionamentos. Uma análise detalhada dessa seção pode auxiliar na identificação de comportamentos alternativos, frequentemente omitidos na seção "Participantes". Portanto, novos pontos variáveis podem ser implementados para permitir, por exemplo, novos atributos ou métodos das classes e algoritmos alternativos para computar o valor dos atributos.

No terceiro sub-passo refina-se a especificação de cada ponto variável, com o intuito de suprir informação suficiente para que ele possa ser posteriormente projetado e implementado. Além do mais, identificam-se outros pontos variáveis não explícitos na linguagem de padrões, mas que poderiam ser incluídos para aumentar a flexibilidade do framework. Essa tarefa pode ser realizada com base no conhecimento geral do projetista do framework a respeito de implementação de frameworks. Informações sobre esses novos pontos variáveis devem ser utilizadas para melhorar a linguagem de padrões, no próximo ciclo de iteração.

No quarto sub-passo deve ser efetuada uma referência cruzada entre os pontos variáveis identificados até então, o que ajuda a detectar inconsistências na lista como um todo, gerando mais alguns pontos variáveis. Conhecimento do domínio é imprescindível nesse ponto e, assim como no terceiro sub-passo, os novos pontos variáveis descobertos são utilizados na melhoria da linguagem de padrões.

No quinto sub-passo, outros aspectos não funcionais da aplicação são considerados, uma vez que eles podem originar novos pontos variáveis, incluindo portabilidade, usabilidade, segurança e confiabilidade. Também devem ser avaliados os aspectos de projeto e implementação que possam trazer mais flexibilidade ao framework, aumentando seu potencial de reuso.

O resultado do passo 2.1 é uma lista dos pontos variáveis do framework, cujos elementos são compostos de um código de identificação do ponto variável, sua descrição, seu tipo, a seção fonte correspondente na linguagem de padrões (opcional) e o padrão no qual esse ponto variável foi identificado. O tipo do ponto variável permite saber o que deve ser feito para obter uma aplicação a partir do framework. Exemplos de possíveis tipos de ponto variável são: PARTIC_ESCOLHA: estabelece que haverá uma escolha entre participantes de um padrão; PADRÃO_OPCIONAL: determina que todo o padrão pode ser aplicado opcionalmente em aplicações específicas; e PARTIC_OPCIONAL: permite que um participante de um padrão não seja utilizado em instanciações particulares.

4.2.2 Projeto do framework

Identificados os pontos variáveis do framework, inicia-se então o projeto do framework (passo 2.2 da Figura 8.3). O framework deve possuir uma estrutura flexível de modo a acomodar todos os pontos variáveis identificados no passo anterior. Dois sub-passos são necessários para projetar o framework: projeto arquitetural e projeto da hierarquia de classes, conforme ilustrado na Figura 8.5.

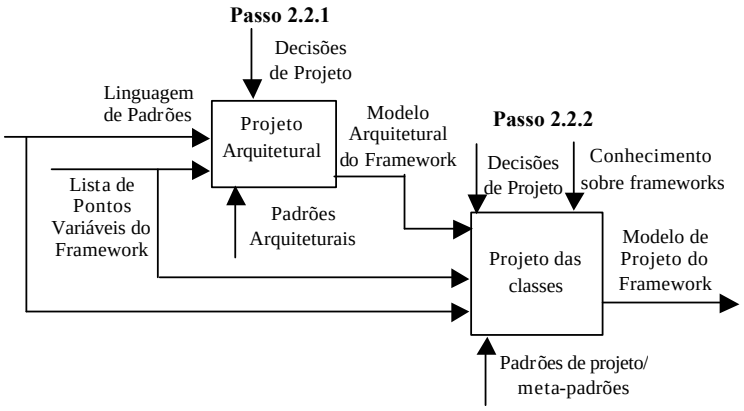


Figura 4.3: Projeto do Framework

O primeiro sub-passo é responsável pelo projeto arquitetural do framework, durante o qual o desenvolvedor toma decisões a respeito da arquitetura de software que melhor atende aos requisitos do framework e de seu público alvo. O mecanismo de persistência dos objetos deve ser escolhido e projetado, bem como a interface gráfica com o usuário, as questões de segurança e controle de acesso, e, finalmente, aspectos de distribuição. Esse sub-passo envolve não somente o desenvolvedor do framework mas também a gerência da organização, pois a escolha da arquitetura de software implica em maiores gastos e tempo de desenvolvimento, ao mesmo tempo em que proporciona um framework mais ou menos reutilizável a longo prazo.

O segundo sub-passo consiste da criação da hierarquia de classes do framework, a partir da estrutura proposta por cada padrão da linguagem de padrões. Assume-se que os padrões possuem uma seção com sua estrutura, na forma de um diagrama de classes ou modelo de análise similar. Assim, pode-se utilizar esse diagrama de classes quando for dado início ao projeto da porção do framework que trata do problema específico solucionado pelo padrão. Também são entradas para esse processo a lista dos pontos variáveis do framework, padrões de projeto, como os de Gamma et al. (1995), e meta-padrões como os de Pree (1995).

Diversas atividades fazem parte desse segundo sub-passo, ilustradas na Figura 8.6. Inicia-se com a criação de um diagrama geral do framework, que combina todas as classes de cada padrão da linguagem, incluindo sub-padrões e variantes. Esse diagrama geral é então refinado usando os

mecanismos da orientação a objetos, como generalização e especialização. Depois disso, investiga-se a possibilidade de usar padrões de projeto e meta-padrões para implementar a flexibilidade almejada. Na verdade, essa investigação envolve não somente a tomada de decisões, mas também o conhecimento de técnicas usuais de implementação de frameworks. Se necessário, pode-se estudar outros frameworks ou a literatura disponível para adquirir tal conhecimento. O diagrama resultante representa, basicamente, toda a funcionalidade coberta pela linguagem de padrões. Então, analisa-se o modelo arquitetural do framework, produzido no primeiro sub-passo, para identificar outros elementos da arquitetura que devem ser implementados por meio de novas classes não previstas na linguagem de padrões. Por exemplo, devem ser incluídas todas as classes que lidam com a arquitetura de software escolhida, incluindo camadas de persistência, segurança, etc. O resultado dessa fase é o modelo completo de projeto do framework.

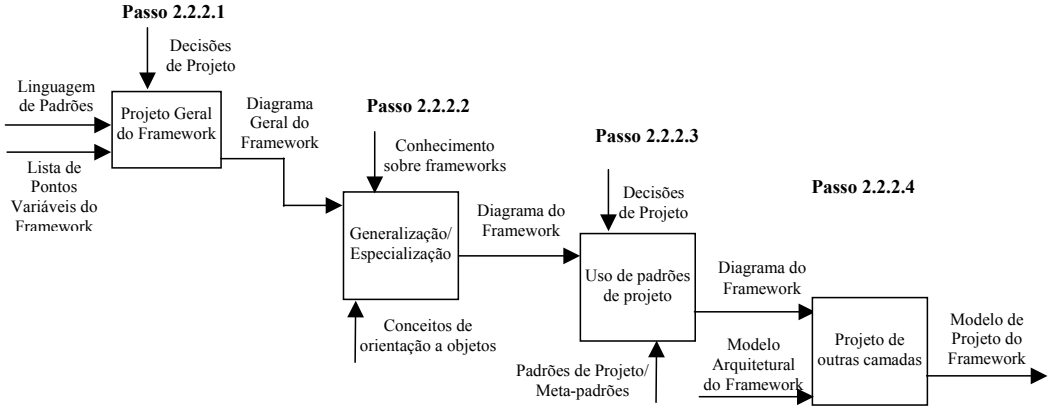


Figura 4.4: Criação da Hierarquia de Classes do Framework

4.2.3 Implementação do framework

Encerrado o projeto do framework, a próxima atividade a ser conduzida é sua implementação (Passo 2.3 da Figura 8.3), constituída de dois sub-passos, ilustrados na Figura 8.7: implementação das classes em uma linguagem de programação específica e documentação do mapeamento entre a linguagem de padrões e o framework.

No primeiro sub-passo as classes do framework, definidas pelo modelo de projeto obtido no passo anterior, são implementadas usando uma linguagem de programação específica. A linguagem de padrões fornece o apoio necessário ao programador, quando surgirem dúvidas quanto à funcionalidade das classes participantes, já que ela é uma fonte excelente de informação sobre o domínio. Outras decisões específicas sobre a implementação do sistema devem ser tomadas neste passo como, por exemplo, a escolha das estruturas de dados, a escolha dos tipos de dispositivos

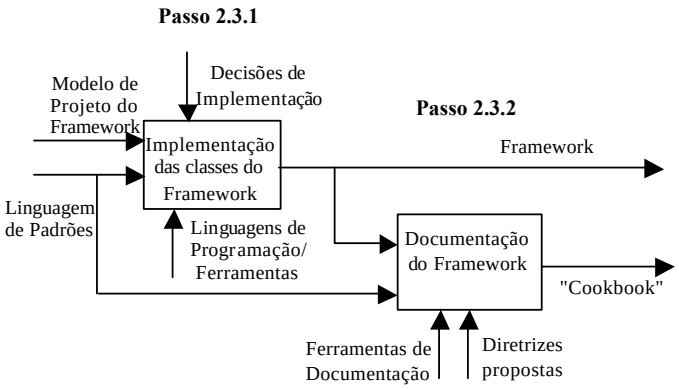


Figura 4.5: Implementação do Framework

utilizados para entrada de dados na interface gráfica com o usuário e o estilo dos relatórios do sistema. O resultado deste passo é o código do framework.

Ressalta-se, aqui, que este sub-passo pode ser bastante complexo, dependendo da arquitetura de software definida no passo anterior. Componentes de software ou bibliotecas de classes podem ser reusadas de outros projetos. Além disso, ferramentas de Engenharia de Software podem facilitar a execução deste sub-passo.

A linguagem de padrões pode ser utilizada para guiar a implementação do framework de forma incremental. Cada padrão pode ser considerado como uma unidade funcional, de maneira que a implementação pode começar pelo primeiro padrão e prosseguir com os demais padrões até que todo o framework esteja implementado. Todavia, isso não é sempre trivial de se fazer: durante a fase de projeto (passo 2.2), as classes participantes de cada padrão podem ter sido decompostas e especializadas. Além disso, o modelo de projeto do framework pode conter classes que participam de diversos padrões ao mesmo tempo. Torna-se necessário, então, implementar essas classes de forma parcial, ou seja, implementar apenas os atributos e métodos relativos à funcionalidade esperada para o padrão sendo implementado. Portanto, os padrões devem ser estudados previamente e deve ser estabelecida uma estratégia para implementação gradual do sistema.

O segundo sub-passo trata da documentação adequada do framework. As diretrizes fornecidas no processo proposto nesta tese conduzem a uma documentação que fornece o mapeamento da linguagem de padrões para as classes do framework. O propósito geral é de facilitar a instanciação do framework para aplicações específicas, de forma que, tendo em mãos o histórico de padrões e variantes aplicados na modelagem, a documentação do framework forneça meios de identificar quais classes devem ser criadas, bem como quais classes do framework devem ser especializadas para originá-las, e quais métodos devem ser sobrepostos. Resumindo, usuários do framework podem facilmente saber o que deve ser feito em cada parte do framework para obter sua aplicação

específica, de acordo com a forma na qual ele usou os padrões da linguagem de padrões para modelar sua aplicação.

Assim, recomenda-se a criação de duas tabelas neste sub-passo: a "Tabela de Mapeamento de Classes" e a "Tabela de Mapeamento de Métodos". A primeira tabela estabelece o relacionamento entre as classes participantes de cada padrão e as classes do framework. Ao instanciar aplicações utilizando um framework caixa branca, desenvolvedores precisam criar classes relativas à aplicação específica, que devem herdar das classes adequadas do framework. De acordo com o processo proposto nesta tese, como a instanciação é apoiada pela linguagem de padrões, então o desenvolvedor sabe exatamente quais padrões, variantes ou sub-padrões foram utilizados, bem como os papéis desempenhados por cada classe da aplicação no padrão correspondente. Assim, essa tabela deve conter o nome do padrão, o variante ou sub-padrão utilizado, as classes participantes do padrão e as classes correspondentes do framework. Esse mapeamento não é, necessariamente, de um para um, já que uma classe que participa de um padrão pode ter sido implementada no framework por meio de diversas classes. Da mesma forma, várias classes diferentes da linguagem de padrões podem corresponder a uma única classe do framework. Um exemplo de "Tabela de Mapeamento de Classes" é dado na Tabela 4.2 da seção 4.3.

A segunda tabela indica quais métodos devem ser sobrepostos nas novas classes criadas. Na verdade, ela documenta os métodos-gancho do framework (ver seção 2.2.2), isto é, os métodos declarados em classes abstratas do framework, para as quais espera-se que o código (ou parte dele) seja fornecido pelas classes herdeiras. Essa tabela deve ter pelo menos três colunas: a primeira com o nome da super-classe, isto é, o nome da classe do framework que possui métodos abstratos a serem sobrepostos pelas classes descendentes; a segunda com a assinatura do método (nome, parâmetros e tipo de retorno); e a terceira com uma descrição completa do método, como sobrepô-lo e um exemplo de código-fonte.

De posse dessas duas tabelas, o desenvolvedor de aplicações que utiliza o framework para instanciar suas aplicações pode descobrir quais classes especializar, sem ter que conhecer e estudar a fundo a hierarquia de classes do framework. Mais do que isso, a forma com que essas informações estão armazenadas permite construir uma ferramenta para automatizar a tarefa de instanciação, que é o passo 3 do processo geral proposto (Figura 8.1).

4.2.4 Validação do framework

A atividade final do processo de construção do framework é sua validação (Passo 2.4 da Figura 8.3), composta dos quatro sub-passos mostrados na Figura 8.8. O objetivo da validação é testar o framework para diversas aplicações específicas antes que ele seja liberado para uso geral. As entradas para esse processo são o framework, seu manual de instanciação e os documentos de requisitos para várias aplicações específicas do domínio. A linguagem de padrões pode auxiliar

o desenvolvedor do framework a definir quais aplicações deveriam ser testadas. Por exemplo, um conjunto de aplicações pode ser criado para exercitar, no mínimo, todos os padrões da linguagem de padrões, inclusive seus variantes e sub-padrões.

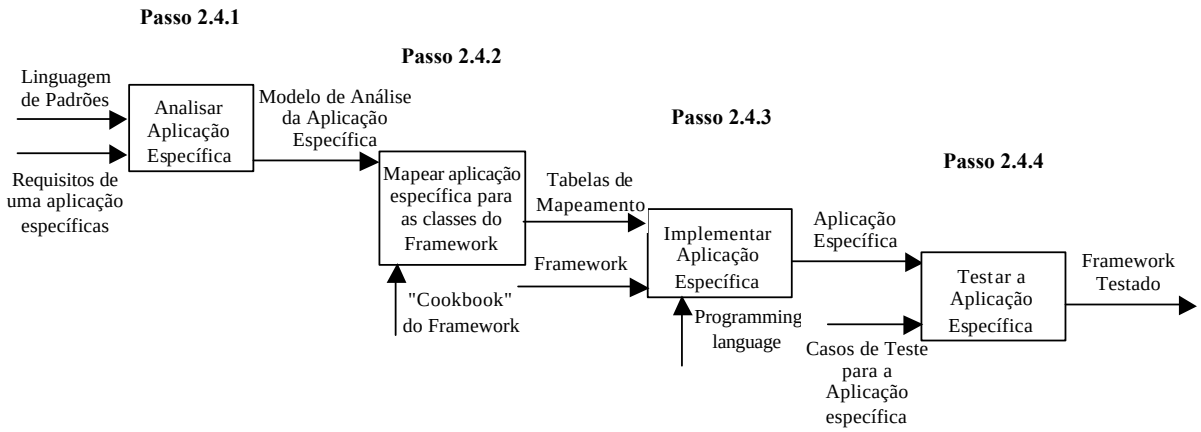


Figura 4.6: Validação do Framework

Deve-se ressaltar que o processo de validação utiliza o mesmo processo de instanciação de aplicações, descrito na seção 4.4, já que, para testar o framework, ele deve ser instanciado para aplicações concretas do domínio.

Para cada aplicação diferente a ser testada, seus requisitos são analisados com base na linguagem de padrões, tendo como resultado seu modelo de análise. A seguir, faz-se o mapeamento entre esse modelo de análise e as classes do framework, utilizando as tabelas de "Mapeamento de Classes" e de "Mapeamento de Métodos" criadas no passo anterior. A próxima atividade é implementar as classes concretas da aplicação, utilizando a mesma linguagem de programação na qual o framework foi implementado. O resultado é o código da aplicação específica, que deve ser devidamente testado. O teste não faz parte do escopo desta tese, mas sugere-se que seja selecionada uma estratégia de testes que leve em conta as particularidades de frameworks como, por exemplo, o fato de que os erros encontrados podem ser do framework ou da aplicação específica em si. Feita a correção desses erros, novos testes devem ser feitos, em diversos ciclos, até que a decisão de liberar o framework seja tomada, seja pelo desenvolvedor do framework ou pela gerência do projeto.

4.3 Exemplo – GREN: Um Framework para Gestão de Recursos de Negócios

O framework GREN (Gestão de REcursos de Negócios) foi desenvolvido com base na linguagem de padrões GRN (ver Seção 3.5), usando o processo apresentado na Seção 4.2. É um framework

caixa branca, já que o reuso é feito por meio de herança (ver Seção 2.2.2), e permite a criação de aplicações no domínio de sistemas de gestão de recursos de negócios.

No primeiro passo do processo de construção do GREN, a linguagem de padrões GRN foi a principal fonte para identificação dos pontos variáveis. A Tabela 4.1 mostra alguns dos quarenta pontos variáveis identificados inicialmente. Deles, trinta e seis foram encontrados por meio da GRN. Os demais foram encontrados ao refinar e fazer a referência cruzada na lista de pontos variáveis. Por exemplo, o segundo ponto variável foi derivado do padrão 2 – QUANTIFICAR O RECURSO (ver seção 3.5.2), pela análise de seus participantes, estrutura e sub-padrões. Esse ponto variável permite quatro tipos diferentes de quantificação para o recurso, de acordo com os quatro sub-padrões apresentados na estrutura do padrão: RECURSO SIMPLES, RECURSO MENSURÁVEL, RECURSO INSTANCIÁVEL e RECURSO EM LOTES. É um ponto variável do tipo PARTIC_OPCIONAL, porque o usuário do framework deverá escolher qual das classes participantes utilizará, de acordo com a estratégia de quantificação desejada para o sistema alvo.

Tabela 4.1: Lista parcial dos Pontos Variáveis do Framework GREN

Nº P.V.	**Nome do Padrão**	**Descrição**	**Tipo**	**Fonte na GRN**	**Nº Padrão**
1	Qualificação do Recurso	Um recurso pode ou não ter um tipo a ele relacionado. Pode também ter múltiplos tipos ou tipos aninhados.	PARTIC-ESCOLHA	Participantes, Variantes	1
2	Quantificação do Recurso	Um recurso pode ser único, pode ter múltiplas instâncias, pode ser gerenciado em quantidades ou em lotes.	PARTIC-ESCOLHA	Participantes, Estrutura, Variantes (sub-padrões)	2
3	Armazenagem do Recurso	Pode ser ou não desejável que a aplicação gerencie a armazenagem do recurso.	PADRÃO-OPCIONAL	Grafo da Linguagem + Contexto	3

O segundo passo foi projetar o framework GREN, começando pelo projeto de sua arquitetura, que 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 4.7. A primeira versão do GREN não inclui aspectos de segurança, controle de acesso, distribuição e interface Web, sendo que esses aspectos poderão ser incorporados em suas futuras versões.

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

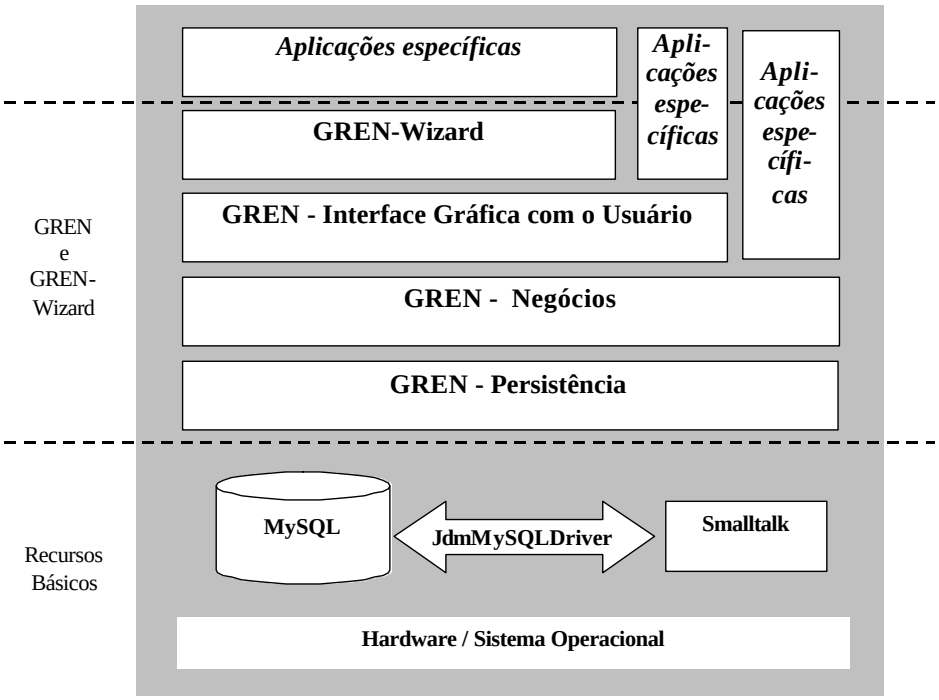


Figura 4.7: Arquitetura do GREN

negócios. Existe ainda o GREN-Wizard, apresentado em detalhes na seção 5.5, que situa-se acima da camada de interface gráfica com o usuário, pois utiliza todas as demais camadas.

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. Podem também ser implementadas por meio do GREN-Wizard, que se encarrega de fazer a comunicação com as demais camadas do GREN.

Definida a arquitetura do GREN, foi então projetada sua hierarquia de classes, começando 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* (Gamma et al., 1995), para conseguir a flexibilidade necessária para atender aos pontos variáveis do framework.

A Figura 4.8 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 *PersistenceLayer* (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 o segundo ponto variável do GREN, conforme descrito na seção 4.2.1. Cada objeto da classe *Resource* refere-se a um objeto da classe (*QuantificationStrategy* da Figura 4.8), o qual é responsável por seus aspectos de quantificação, permitindo que o framework implemente as quatro diferentes soluções requeridas. Outros padrões de projeto foram utilizados na implementação do GREN, em especial os padrões *Factory Method* e *Template Method*.

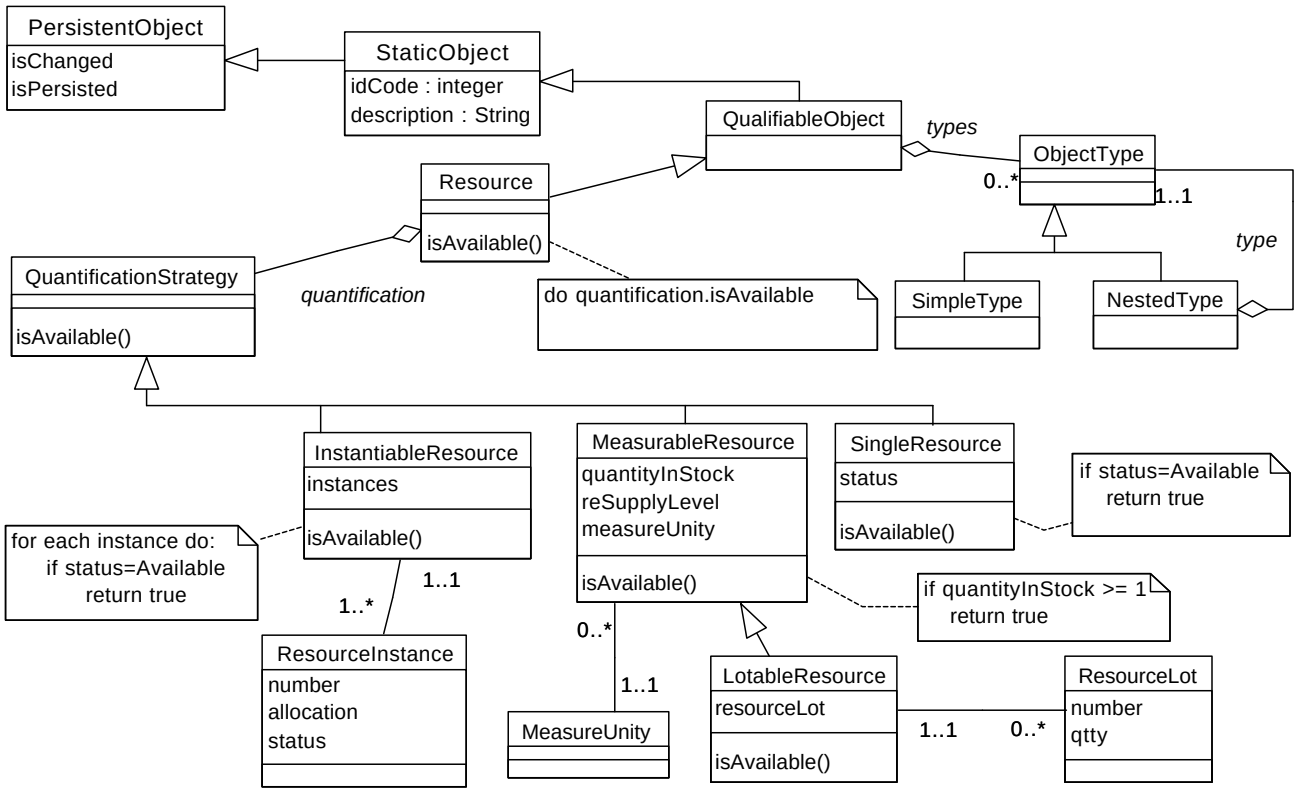


Figura 4.8: Exemplo de algumas classes do GREN

O terceiro passo da construção do GREN foi a implementação de suas classes, que utilizou 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. A hierarquia de classes do GREN, bem como mais alguns diagramas de classes, é apresentada no Apêndice F.

A linguagem de padrões GRN foi utilizada para implementar o GREN de forma gradual, seguindo seqüencialmente seus quinze padrões. A implementação do primeiro padrão, IDENTIFICAR O RECURSO, deu origem a um pequeno framework que, ao ser instanciado, originava aplicações para manutenção de registro de recursos, como por exemplo, um sistema para controle de discos compactos (CDs) de um indivíduo. Algumas operações possíveis nesse sistema são: cadastrar

os CDs, imprimir listagens de CDs em ordem alfabética, eliminar CDs, etc. A seguir foi implementado o segundo padrão, QUANTIFICAR O RECURSO, que estendeu o framework para tratar das possíveis formas de quantificar o recurso, como as já mencionadas anteriormente nesta seção. Assim, no caso do sistema de controle de CDs, seria agora possível cadastrar várias cópias de um mesmo CD. Usando esse mesmo raciocínio, cada um dos demais padrões da GRN foi implementado, culminando com a versão final do framework GREN.

Para completar o passo de implementação do GREN, foi então elaborada sua documentação segundo as diretrizes do processo proposto, o que resultou no Manual de Instanciação do GREN[1], denominado "*Cookbook* do GREN". Ele contém quatro tabelas que permitem mapear a GRN ao GREN. As duas primeiras tabelas correspondem à "Tabela de Mapeamento de Classes" e as duas últimas tabelas à "Tabela de Mapeamento dos Métodos". Essas tabelas foram divididas para separar as classes da camada de negócios das classes da camada de interface gráfica com o usuário.

Uma amostra dessas duas primeiras tabelas é mostrada nas Tabelas 4.2 e 4.3. A Tabela 4.2 é utilizada na criação das classes concretas da camada de negócios e a Tabela 4.3 na criação das classes da camada GUI. As classes da camada de negócios preocupam-se apenas com a funcionalidade do sistema. Cada classe dessa camada pode ter uma ou mais classes correspondentes na camada GUI, voltadas aos aspectos de entrada e saída de dados. Deve-se notar que essas tabelas foram construídas com base na linguagem de padrões GRN.

Tabela 4.2: Exemplo da documentação do GREN - Tabela usada para identificar novas classes da aplicação

Padrão	**Variante**	**Classe do Padrão**	**Classe do GREN**	**Cod.Ref.**
1-Identificar o Recurso	Todas	Recurso	Resource	N1
	Default, Tipos Múltiplos	Tipo de Recurso	SimpleType	N2
	Tipos Aninhados	Tipo de Recurso	NestedType	N2
2 - Quantificar o Recurso	Recurso Instanciável	Instância do Recurso	ResourceInstance	N3
	Recurso Mensurável ou em Lotes	Unidade de Medida	MeasureUnity	N4
	Recurso em Lotes	Lote do Recurso	ResourceLot	N5
	Recurso Simples	*nada a fazer*	–	–
9 - Manter o Recurso	Todas	Manutenção do Recurso	ResourceMaintenance	N21
		Origem	SourceParty	N6
		Destino	DestinationParty	N14

Um exemplo das tabelas de mapeamento dos métodos do GREN é apresentado na Tabela 4.4. Ela lista os métodos a serem sobrepostos pelas novas classes criadas e deve ser percorrida seqüencialmente, usando informações sobre a hierarquia das novas classes da aplicação sendo instanciada. Quando encontra-se um método que pertence a uma classe abstrata do GREN, especializada

[1]BRAGA, R. T. V.; MASIERO, P. C. Manual de Instanciação do Framework GREN usando a GRN. ICMC-USP - São Carlos, documento de trabalho, 94p., 2002.

Tabela 4.3: Exemplo da documentação do GREN - Tabela usada para identificar novas classes da GUI

Padrão	Variante	Cod.Ref.	Classe GREN
1-Identificar o Recurso	Default, Tipos Múltiplos	N2	StaticObjectForm
	Tipos aninhados	N2	QualifiableObjectForm
2 - Quantificar o Recurso	Recurso Simples	N1	SingleResourceForm
	Recurso Mensurável	N1	MeasurableResourceForm
	Recurso Instanciável	N1	InstantiableResourceForm
	Recurso em Lotes	N1	LotableResourceForm
	Recurso Instanciável	N3	ResourceInstanceForm
	Recurso Mensurável	N4	MeasureUnityForm
	Recurso em Lotes	N5	*ainda não implementado*
9 - Manter o Recurso	Aplicando padrões 14 e 15	N21	OneResourceMaintWPWTForm
	Sem apliar padrões 14 e 15	N21	OneResourceMaintNPNTForm
	Todas	N6	SourcePartyForm
	Todas	N14	DestinationPartyForm

durante a instanciação, então esse método deve ser sobreposto. A coluna "Instância/Classe" determina se o método será invocado por instâncias da classe ou pela própria classe. A coluna "Comentários" descreve a função do método de forma suficiente para que o desenvolvedor possa codificá-lo.

O "*Cookbook* do GREN" possui também algoritmos para serem utilizados em conjunto com as tabelas, para permitir o uso disciplinado dessas tabelas na obtenção das classes e métodos a serem implementados. Um exemplo desse tipo de algoritmo é mostrado na seção 4.4.2.

Tabela 4.4: Exemplos de métodos a serem sobrepostos no GREN

Super-classe	Método	Instância/Classe	Comentário
QualifiableObject	typeClasses	C	retorna um objeto List com as classes que representam o tipo do recurso (zero ou mais).
ResourceMaintenance	resourceClass	C	retorna a classe que representa o recurso sendo mantido.
	hasSourceParty	C	retorna um booleano, sendo true se o participante "Origem" foi utilizado durante a instanciação do padrão 2, ou false caso contrário, já que esse participante é opcional nesse padrão.
	sourcePartyClass	C	retorna o nome da classe da aplicação que desempenha o papel de Destino no padrão. Esse método somente deve ser sobreposto se o participante Destino foi utilizado durante a instanciação desse padrão.
	destinationPartyClass	C	retorna o nome da classe da aplicação que desempenha o papel de Destino no padrão.

O quarto e último passo da construção do GREN foi sua validação, com o intuito de verificar se aplicações produzidas a partir dele funcionavam corretamente. O GREN foi testado, pela própria autora, para três aplicações diferentes, escolhidas cuidadosamente de modo a exercitar todos os quinze padrões da GRN. Entretanto, não foi possível, durante esses testes, validar todos os variantes e sub-padrões, o que pretende-se fazer em futuros esforços. As aplicações utilizadas no teste foram: uma oficina de consertos de veículos, uma locadora de vídeos e uma loja de venda e aluguel de produtos para festas. Após o término desses testes, foram feitos alguns estudos com alunos de graduação e pós-graduação, nos quais eles receberam a tarefa de desenvolver duas aplicações, mais

especificamente para um hotel e para uma locadora de carros, com base em requisitos fornecidos pela autora. Esses estudos estão descritos detalhadamente no Capítulo 6, e também serviram para testar o framework, dessa vez para sistemas que exercitavam outros variantes dos padrões. Diversos erros foram encontrados durante os testes e os estudos de caso, que foram corrigidos ora no código-fonte, ora no "*Cookbook* do GREN", ou em ambos, se fosse o caso.

4.4 O Processo de Instanciação

A reutilização de um framework caixa branca é feita por meio de herança. Os requisitos da aplicação a ser instanciada devem ser analisados para descobrir quais são suas classes e a partir de quais classes do framework elas devem ser especializadas. No processo proposto nesta tese, como o framework foi desenvolvido com base em uma linguagem de padrões, a instanciação pode ser por ela guiada. Assim, é sugerido o processo de instanciação ilustrado na Figura 8.9. Ele é composto de quatro passos, detalhados nas próximas sub-seções: análise do sistema, mapeamento entre o modelo de análise e o framework, implementação das classes específicas e teste do sistema resultante.

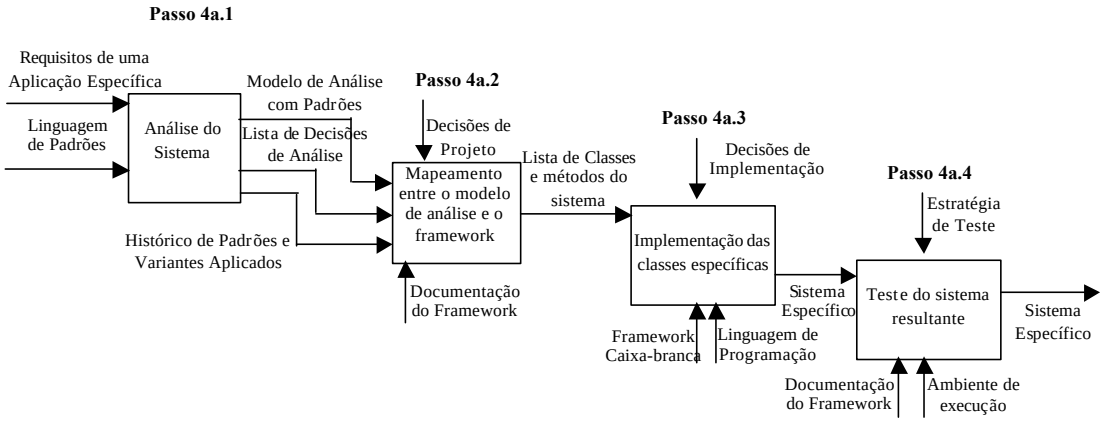


Figura 4.9: Processo de Instanciação do Framework Caixa branca

4.4.1 Análise do Sistema

O primeiro passo para instanciar um framework caixa branca é a análise do sistema, com base na linguagem de padrões (passo 4a.1 da Figura 8.9). Esse passo é explicado detalhadamente na Seção 3.4, que mostra o processo de uso de uma linguagem de padrões para modelagem um sistema específico no mesmo domínio da linguagem. São entradas para esse passo o documento de requisitos de um sistema específico e a linguagem de padrões. O resultado é um modelo de análise

do sistema, o histórico dos padrões e variantes utilizados e uma lista de decisões tomadas quando a linguagem de padrões não atende a todos os requisitos do sistema.

4.4.2 Mapeamento entre o modelo de análise e o framework

De posse do modelo de análise do sistema e do histórico de padrões e variantes utilizados, aplica-se então o segundo passo do processo geral (passo 4a.2 da Figura 8.9) para produzir o mapeamento entre o modelo de análise e as classes do framework. Assume-se que o framework tenha sido documentado de acordo com as diretrizes sugeridas no passo de implementação do framework (ver Seção 4.2.3), ou seja, o relacionamento entre os padrões da linguagem de padrões e as classes do framework deve estar documentado apropriadamente. O resultado deste passo é uma lista das classes da aplicação e métodos correspondentes a serem implementados.

O procedimento a ser seguido para fazer o mapeamento entre o modelo de análise e o framework é específico de cada par "linguagem de padrões e framework". O algoritmo genérico, fornecido a seguir, contém uma abstração das atividades importantes na realização desse mapeamento:

1. Considere o "Histórico de Padrões e Variantes Aplicados" criado durante a análise do sistema (item 11 da Seção 3.4) que contém, para cada padrão aplicado, o variante ou sub-padrão utilizado e os papéis desempenhados por cada classe. Seja TH o nome dessa tabela. Para cada linha de TH, seja P o padrão aplicado, V o variante ou sub-padrão utilizado, R a classe participante do padrão e A a classe da aplicação, tal que A desempenha o papel R no padrão P e variante V.

2. Construa uma tabela contendo as classes a serem criadas no novo sistema. Seja TC essa tabela. Ela deve ter quatro colunas: classe da aplicação, classe a criar, super-classe do framework e novos atributos.

3. Para cada linha de TH, encontre, na documentação do framework, as classes que devem ser criadas no novo sistema para a tupla $\{$A, P, V, R$\}$. Como resultado são obtidos os nomes das novas classes a serem criadas referentes a A, denominadas $A_1, A_2, ..., A_n$, bem como as super-classes do framework das quais elas devem herdar, denominadas $S_1, S_2, ..., S_n$. Crie então, na tabela TC, "n" linhas, cada qual contendo: A, na primeira coluna; A_i, na segunda coluna; S_i, na terceira coluna; e novos atributos acrescentados à classe A, na quarta coluna (tais atributos foram destacados no modelo de análise).

4. Finalmente, considere a tabela TC resultante. Para cada linha de TC, examine a documentação do framework para identificar quais são os métodos que precisam ser sobrepostos.

Defina o conteúdo desses métodos. Inclua também os métodos para implementar os novos atributos e operações.

4.4.3 Implementação do sistema específico

No passo 4a.2 foram definidas todas as classes a serem implementadas na nova aplicação, juntamente com seus métodos. Nesta seção são fornecidas sugestões para implementação propriamente dita dessas classes (passo 4a.3 da Figura 8.9). Deve ser utilizada, na implementação, a mesma linguagem de programação na qual o framework foi implementado. O resultado é o código-fonte das novas classes da aplicação, que deve ser utilizado em conjunto com as classes do framework para compor a nova aplicação. As seguintes recomendações podem ser usadas para guiar a implementação:

1. Inicialmente, crie todas as classes identificadas durante o mapeamento (Seção 4.4.2), assim como os métodos correspondentes.

2. Crie os métodos *set* e *get* (para atribuição e recuperação de valor, respectivamente) para todos os atributos adicionais das classes (quarta coluna da tabela TC).

3. No caso de sistemas de informação convencionais, é comum a existência de uma GUI representando a janela principal da aplicação, juntamente com o menu a ser executado pelo usuário final. Esse menu deve conter atalhos para todas as operações do sistema.

4. Implemente outras classes e operações não previstas no framework. Esta atividade está fora do escopo desta tese. Envolve profundo entendimento do framework caixa branca, para que sejam feitas as conexões da nova funcionalidade implementada com a funcionalidade provida pelo framework.

5. Finalmente, compile a nova aplicação.

4.4.4 Teste do sistema resultante

O sistema obtido no passo 4a.3 deve ser testado, tanto para garantir que atende aos requisitos estabelecidos, quanto para verificar se funciona no ambiente do usuário final. Embora a etapa de teste esteja fora do escopo desta tese, sugere-se que seja escolhida uma estratégia de testes, e que casos de teste sejam elaborados para checar o cumprimento dos requisitos funcionais e não funcionais. O resultado desejado é uma aplicação devidamente testada, que pode ser entregue ao usuário final.

4.5 Exemplo – Instanciação do GREN para Sistema de Acompanhamento e Reparo de Buracos

Nesta seção mostra-se, passo a passo, a instanciação do GREN para o Sistema de Acompanhamento e Reparo de Buracos (SARB), descrito na Seção 3.5.6.

4.5.1 Análise do SARB

O primeiro passo da instanciação é a análise do sistema SARB com base na linguagem de padrões GRN. Essa análise encontra-se descrita em detalhes na Seção 3.5.6 e, portanto, não é repetida nesta seção. O resultado da análise do SARB é o modelo de classes mostrado na Figura 3.7 e o histórico de padrões e variantes utilizados apresentado na Tabela 3.1 (ver Capítulo 3).

4.5.2 Mapeamento do SARB para o GREN

O segundo passo consiste em fazer o mapeamento entre o SARB e o framework GREN, utilizando a documentação do GREN e os resultados do primeiro passo. Conforme já explicado na Seção 4.3, a documentação do GREN é constituída de diversas tabelas que mostram, para cada padrão da GRN, as classes correspondentes do GREN a serem especializadas e os métodos a serem sobrepostos durante a instanciação. Além disso, a documentação do GREN fornece diversos algoritmos que auxiliam na criação de novas classes, adaptação da interface gráfica com o usuário para incluir novos atributos, e programação dos novos menus do sistema. A implementação das classes deve ser feita utilizando o ambiente VisualWorks, para Smalltalk, que é a linguagem na qual o GREN está implementado.

No algoritmo genérico da Seção 4.4.2, estabelece-se que dada uma classe A da nova aplicação, pode ser necessário criar classes $A_1, A_2, ..., A_n$, herdando das classes $S_1, S_2, ..., S_n$ do framework. No caso do GREN, para a maioria das classes da nova aplicação são criadas duas classes: uma delas pertence à camada de negócios e tem o mesmo nome (“A”, no caso) e a outra pertence à camada de interface com o usuário e tem esse mesmo nome, seguido do sufixo “Form”. Por exemplo, a classe *Cidadao* do SARB será implementada por meio de duas classes: uma referente à camada de negócios, que terá o nome “Cidadao” e outra referente à camada da GUI, que terá o nome “CidadaoForm”.

A identificação dessas classes é feita com base no histórico dos padrões e variantes utilizados – Tabela 3.1 (TH), na Tabela 4.2 do GREN para identificar as novas classes da camada de negócios do SARB a serem criadas, e na Tabela 4.3 para identificar as novas classes da camada GUI do

SARB a serem criadas. O resultado é mostrado, parcialmente, na Tabela 4.5 (TC)[2]. As Tabelas 4.2 e 4.3 são utilizadas da seguinte forma:

1. Para todas as linhas da tabela TH, faça uma busca na Tabela 4.2 usando como parâmetros as três primeiras colunas da tabela TH. Se a busca tiver sucesso, o conteúdo da quarta coluna da linha resultante é a classe do GREN a ser especializada, portanto adicione uma nova linha a TC, na qual a classe da aplicação é a quarta coluna de TH, a classe a ser criada tem o mesmo nome da classe da aplicação e a super-classe do GREN é a quarta coluna da Tabela 4.2. Por exemplo, no SARB deve-se percorrer a Tabela 3.1 e buscar as linhas correspondentes na Tabela 4.2. Assim, inicia-se pelo padrão número 1, variante *default*, classe Recurso. A busca têm sucesso e retorna a primeira linha da Tabela 4.2. Assim, cria-se em TC uma linha com as seguintes colunas: Buraco, Buraco e Resource (ver linha número 1 da Tabela 4.5). Fazendo o mesmo para as demais linhas da Tabela 3.1, obtêm-se as demais linhas ímpares da Tabela 4.5. Observe que na coluna "Classe a criar" dessa tabela foi anotado o código de referência da classe, obtido a partir da quinta coluna da Tabela 4.2.

2. Para todas as classes da camada de negócios criadas no item 1, use seu código de referência para pesquisar a Tabela 4.3. Se o código de referência não for encontrado, então não é necessário criar uma classe de interface gráfica com o usuário para a classe A. Se ele for encontrado, provavelmente ocorrendo mais de uma vez na tabela, então as linhas resultantes da tabela devem ser analisadas de acordo com os variantes ou sub-padrões utilizados. O resultado será uma nova linha adicionada à TC, na qual a classe da aplicação é a quarta coluna de TH, a classe a ser criada possui o mesmo nome, com o sufixo "Form" (ou outro sufixo significativo), e a super-classe do GREN é a quarta coluna da Tabela 4.3. Por exemplo, se for feita a busca do código de referência N1 na Tabela 4.3, serão recuperadas quatro linhas. Como o variante utilizado para quantificar o *Buraco* foi "Recurso Simples", então a classe da camada GUI do GREN a ser usada como super-classe de *BuracoForm* é a classe *SingleResourceForm*. Assim, a linha 2 da Tabela 4.5 é obtida. Usando esse mesmo raciocínio são obtidas as demais linhas pares da Tabela 4.5.

Ainda no segundo passo da instanciação do SARB, é necessário examinar as classes criadas e consultar a documentação do GREN para definir os métodos a serem sobrepostos (nomes e conteúdos). Por exemplo, a Tabela 4.4 mostra uma pequena parte da tabela do GREN responsável por esse mapeamento. Essa tabela deve ser utilizada da seguinte forma:

[2]Foi acrescentada uma coluna à esquerda dessa tabela para numerar suas linhas, a fim de facilitar a explicação de seu conteúdo no decorrer do texto

Tabela 4.5: Algumas classes a serem criadas no SARB (TC)

Linha n^o	Classe da Aplicação	Classe a criar	Super-classe do GREN	Atributos adicionais
1	Buraco	Buraco (N1)	Resource	
2	Buraco	BuracoForm	SingleResourceForm	
3	Tamanho	Tamanho (N2)	SimpleType	prioridadeReparo
4	Tamanho	SizeForm	StaticObjectForm	
5	Ordem de Serviço	OrdemDeServico (N21)	ResourceMaintenance	numeroDePessoasEquipe
6	Ordem de Serviço	OrdemDeServicoForm	OneResourceMaintWPWTForm	
7	Cidadão	Cidadao (N14)	DestinationParty	endereco, telefone
8	Cidadão	CidadaoForm	DestinationPartyForm	

- A primeira coluna dessa tabela contém o nome de uma classe abstrata do GREN, isto é, uma classe que possui métodos que devem ser sobrepostos por quaisquer classes descendentes. Chamemos essa classe de X.
- Para cada classe do SARB, verifique se a classe X faz parte de sua hierarquia. Em caso positivo, analise cada um dos métodos da classe X (segunda coluna da Tabela 4.4) para verificar se é necessário sobrepor esse método ou não (examine o comentário da quarta coluna para descobrir isso).
- Caso encontre um método a ser sobreposto, a quarta coluna da Tabela 4.4 fornece informações sobre como codificar tal método.

A Figura 4.10 mostra parte da hierarquia de classes do GREN, usando a notação UML-F (Fontoura et al., 2001). Classes que contém métodos-gancho são classes abstratas, pois esses métodos devem ser sobrepostos pelas classes descendentes (note que eles aparecem na Tabela 4.4). Na verdade, alguns desses métodos são opcionalmente sobrepostos, dependendo do uso do framework. Por exemplo, o método `hasSourceParty` da classe abstrata *BusinessResourceTransaction* deve ser sobreposto obrigatoriamente, enquanto o método `sourcePartyClass` é opcional, dependendo se o participante *Origem* do padrão MANTER O RECURSO (Figura 3.5) foi usado. Decidiu-se criar a Tabela 4.4 para mostrar esse mapeamento e explicar mais detalhadamente o uso dos métodos, pois o GREN (e outros frameworks também) possui uma quantidade muito grande de métodos-gancho, o que originaria diagramas complexos de serem entendidos.

4.5.3 Implementação das classes do SARB

O terceiro passo na instanciação do GREN para o SARB foi implementar as classes e métodos definidos no passo anterior. A Figura 4.10 mostra alguns trechos de código (por meio de Notas da UML) sobreposto pelos métodos das classes do SARB. Por exemplo, a classe *Buraco* herda da classe *Resource*, que por sua vez herda da classe *QualifiableObject*. Portanto, de acordo com a Tabela 4.4, o método `typeClasses` deve ser sobreposto (a coluna "Comentários" da Tabela 4.4 mostra como esse método foi codificado). Além disso, foram criados métodos para lidar com os

novos atributos das classes, como por exemplo o atributo `prioridadeReparo` da classe *Tamanho*. A documentação do GREN possui uma seção em separado com os métodos a serem sobrepostos no caso de novos atributos, que são os métodos *set* e *get*, os métodos de inicialização do objeto e os métodos de definição da GUI. Um algoritmo especial foi desenvolvido para ensinar, passo a passo, como acrescentar os novos atributos à GUI, de forma a aproveitar todo o projeto oferecido pelo GREN, incluindo o mínimo de código possível.

Além do GREN atender aos requisitos básicos do SARB, estabelecidos na Seção 3.5.6, ele oferece diversos relatórios úteis no gerenciamento dos reparos. Por exemplo, para a classe *Buraco* existe, na GUI correspondente, um botão que permite listar os buracos por endereço, por distrito, por código de identificação, por localização, etc. Para a classe *OrdemDeServico* podem ser obtidos, com facilidade, relatórios de ordens de serviço pendentes, ordens de serviço atendidas em período pré-determinado, valores totais das ordens de serviços em um determinado período, etc.

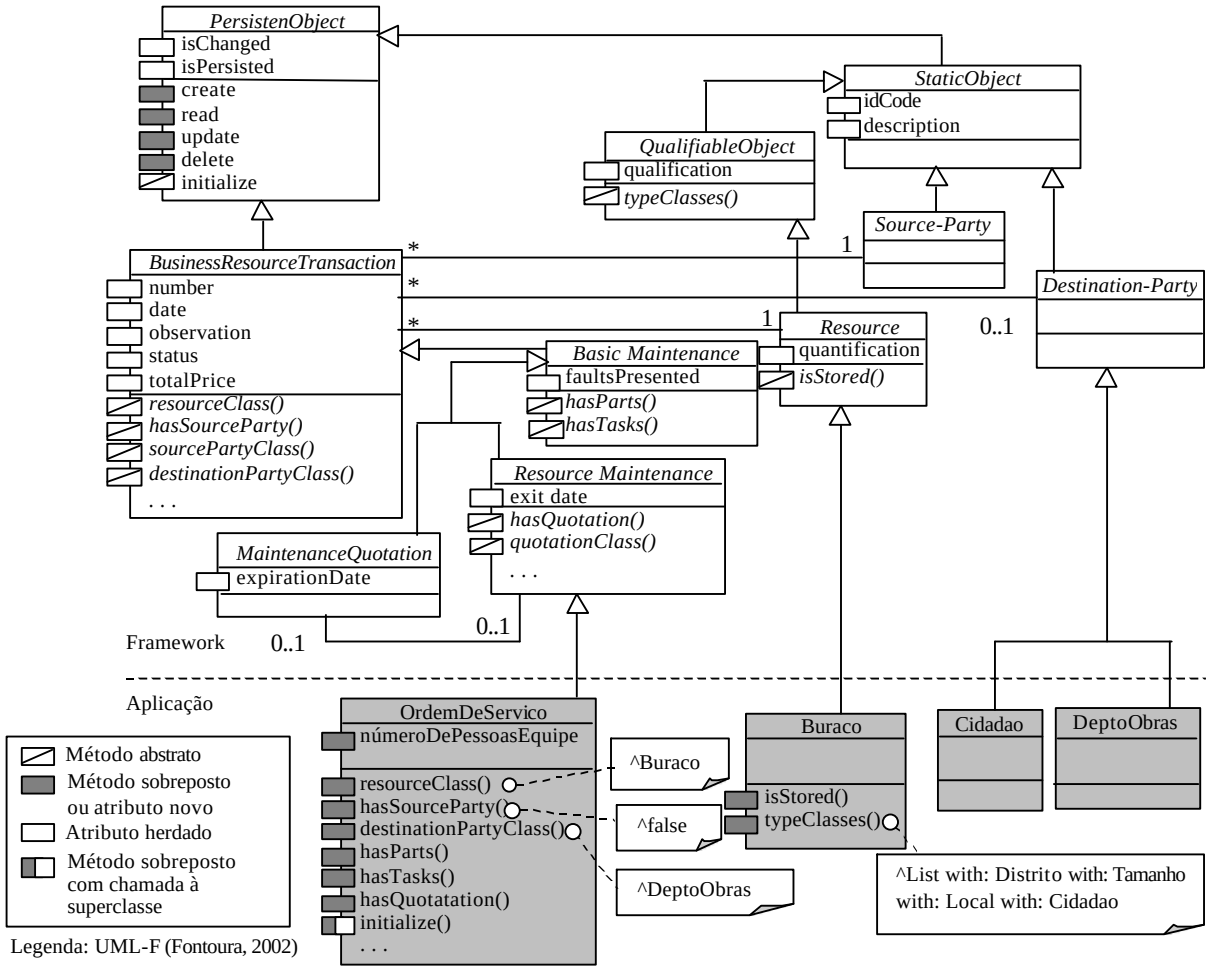


Figura 4.10: Parte da Hierarquia de Classes do GREN e classes derivadas no SARB

Esses relatórios são automaticamente herdados das classes do GREN, sendo necessário sobrepor alguns métodos-gancho somente no caso em que se deseja acrescentar novas colunas aos relatórios.

4.5.4 Teste do SARB

Para cumprir o último passo de instanciação do GREN para o SARB, o GREN foi instalado, as classes do SARB foram incorporadas ao código do GREN e a base de dados MySQL foi criada, de acordo com as recomendações fornecidas na documentação do GREN. Tabelas específicas mapeando os padrões da GRN às respectivas tabelas que devem ser criadas na base de dados podem ser encontradas na documentação do GREN. Isso facilita a etapa de testes, pois pode-se saber exatamente as tabelas a criar, quais são suas colunas e tipos de dados.

No caso do SARB, após criar as tabelas, foi elaborado um conjunto de casos de teste para exercitar o sistema. O menu de opções da Figura 4.11 foi executado pelo menos para três objetos novos cada, executando-se todas as operações disponíveis nos formulários de entrada (inclusão, exclusão, alteração e consulta) para cada um dos objetos. A Figura 4.12 ilustra a GUI para a classe *OrdemDeServicoForm*, na qual pode-se observar algumas das operações permitidas. Os resultados dos testes foram satisfatórios, tendo atendido aos requisitos inicialmente propostos.

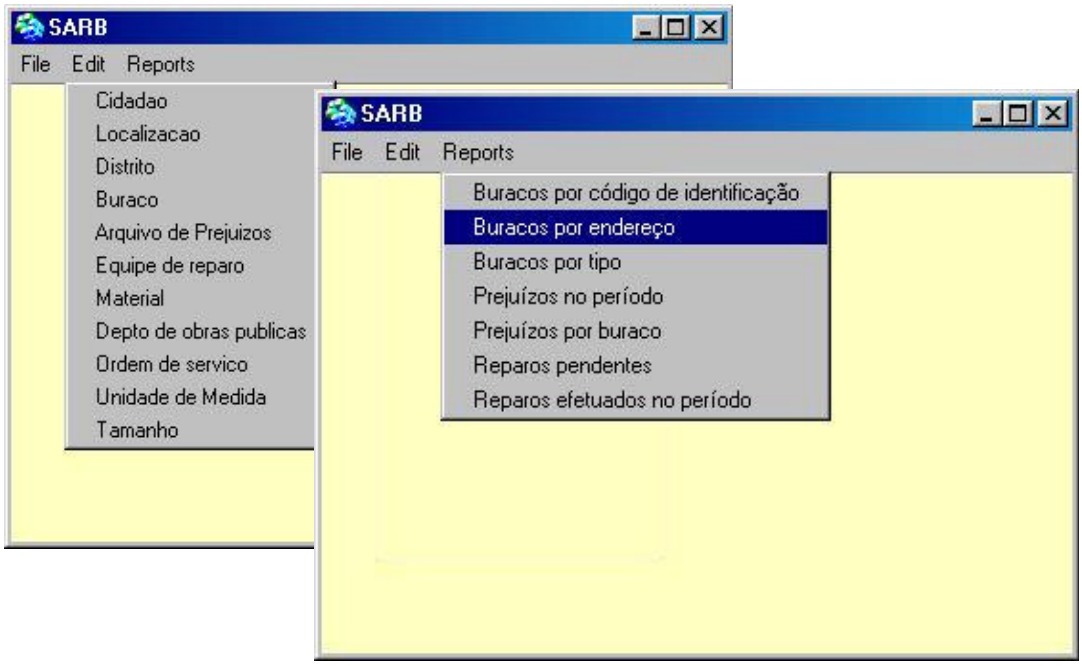


Figura 4.11: Menus e janela principal do SARB

4.5.5 Comentários

Foram gastas aproximadamente nove horas para obter o código final do SARB (duas horas na análise, uma hora no mapeamento, quatro horas na implementação e duas horas em testes). Cerca de mil e seiscentas LOC foram produzidas para adaptar o GREN a esse exemplo específico. Esses números são pequenos se for considerado o número de Pontos por Função (PF) (Albrecht, 1984; Jones, 1991) do sistema resultante, que é de aproximadamente 370 (foi utilizado um fator de 0.9 para calcular o PF). Para obter a mesma funcionalidade programando esta aplicação do nada seriam exigidas, no mínimo, cinco vezes mais linhas de código do que foram necessárias usando o GREN, considerando que a linguagem Smalltalk requer vinte e uma linhas de código por ponto por função (Jones, 1989). Além do mais, a maioria do código adicionado consiste de métodos para lidar com os novos atributos adicionados às classes, que não faziam parte dos padrões. Por exemplo, para cada novo atributo da classe *Cidadão* existem métodos para atribuir, recuperar e inicializar o objeto, além do código na classe da camada GUI para permitir sua edição nos formulários de entrada de dados. Ainda, parte do código é gerado automaticamente por programação visual. Isso ocorre para os formulários da GUI que foram adaptados ao SARB, por exemplo, por meio da mudança de alguns campos de entrada para o local apropriado. Portanto, o código que deve ser

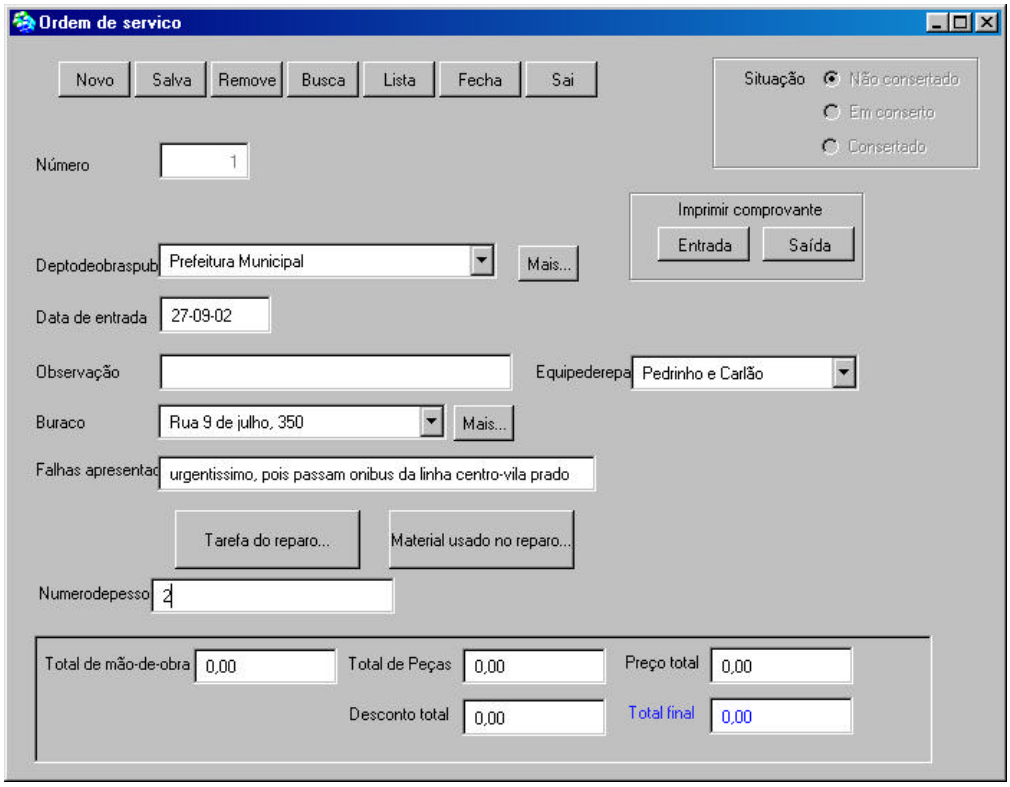


Figura 4.12: Exemplo de GUI para Ordem de Serviço (SARB)

criado é bem mais simples do que o código de uma aplicação convencional, na qual deve-se tratar da implementação de todas as regras de negócio do domínio.

4.6 Considerações Finais

Este Capítulo apresentou em detalhes o processo de construção de um framework caixa branca com base em uma linguagem de padrões para um domínio específico. Para exemplificar esse processo apresentou-se o framework GREN, que foi construído com base na linguagem de padrões GRN. Definiu-se então um processo para instanciação de um framework caixa branca usando a linguagem de padrões correspondente. Esse outro processo também foi ilustrado usando o GREN e a GRN.

Os processos propostos fornecem uma forma sistemática de construir e instanciar um framework, facilitando seu reuso por meio de uma linguagem de padrões que guia a criação de novas aplicações no domínio. A linguagem de padrões é usada durante todo o processo de construção do framework, desde sua concepção e identificação da flexibilidade desejada, até sua validação e documentação. Ainda que trabalhosa, a instanciação do framework para sistemas específicos poderá ser feita com o mínimo de conhecimento sobre o framework e a linguagem de programação na qual ele está escrito, pois a documentação sugerida no processo proposto possui o mapeamento dos padrões de análise pertencentes à linguagem de padrões para as classes e métodos do framework que devem ser adaptados para obter a aplicação resultante. Assim, sabendo-se quais padrões foram aplicados, pode-se consultar essa documentação para saber exatamente o que deve ser feito no framework para obter o código da aplicação final desejada.

Conforme descrito na seção 3.5.6, a GRN foi utilizada para modelar o sub-sistema de controle de prejuízos causados pelos buracos, mesmo não abrangendo essa funcionalidade. Isso foi feito considerando-se apenas a sintaxe dos padrões, ao invés de sua semântica. Da mesma forma, o GREN pôde ser usado para implementar esse sub-sistema, bastando para isso seguir o *cookbook* normalmente, como se faz para os padrões cuja semântica está de acordo com o domínio coberto. Esse fato confirma a afirmação de Johnson (1992), de que “um bom framework será utilizado de maneiras jamais concebidas por seus projetistas”. O sistema resultante comportou-se de forma satisfatória após esse uso mais flexível, evitando que se programasse a partir do zero essa parte da funcionalidade.

O próximo capítulo descreve detalhadamente o processo de desenvolvimento de uma ferramenta (em ingês *wizard*) para automatizar a instanciação do framework caixa-branca com base na linguagem de padrões correspondente (passo 3 da Figura 8.1). Descreve também o processo de instanciação do framework usando esse *wizard*, que consiste no passo 4b da Figura 8.1.

CAPÍTULO

5

O Processo de Construção e Utilização de um *Wizard* baseado na Linguagem de Padrões

5.1 Considerações Iniciais

A dificuldade de reuso de frameworks, ou seja, sua instanciação para atender aos requisitos de sistemas específicos, foi um dos fatores fundamentais para motivação do trabalho descrito nesta tese. O processo proposto na seção 4.4, de instanciação de frameworks com base em uma linguagem de padrões, sistematiza essa tarefa e torna-a passível de ser executada por desenvolvedores não familiarizados com a estrutura interna do framework. No entanto, tal atividade envolve trabalho repetitivo e bastante programação em uma linguagem específica. Por outro lado, a sistematização dessa tarefa, que se tornou viável após a definição do processo de instanciação, motiva a criação de uma ferramenta que automatize esse processo.

Assim, neste Capítulo descrevem-se dois processos: o primeiro detalhando a construção de um *Wizard* para instanciação do framework caixa branca e o segundo para instanciação do framework utilizando esse *Wizard*. Esses processos referem-se aos passos 3 e 4b do processo geral da Figura 8.1 e foram agrupados neste Capítulo devido ao seu relacionamento: o processo de instanciação definido no passo 4b atua como um manual de uso do *Wizard* construído pelo pro-

cesso mostrado no passo 3. Para ilustrar esses dois processos gerais, o processo de construção e instanciação do GREN-Wizard é utilizado.

O Capítulo está organizado da seguinte forma: na seção 5.2 dá-se uma visão geral de duas alternativas para o processo de construção de um *Wizard* para instanciação de frameworks caixa branca com base em linguagens de padrões. Na seção 5.3 propõe-se a arquitetura geral do *Wizard*. Na seção 5.4 apresenta-se o processo para construção de um *Wizard* específico para uma linguagem de padrões e seu framework associado. A seguir, na seção 5.5, esse processo é exemplificado usando o GREN-Wizard, uma ferramenta para instanciação automática do framework GREN. Na seção 5.6 detalha-se o processo de instanciação do framework usando o *Wizard* correspondente. Esse processo é exemplificado, na seção 5.7, por meio da instanciação do framework GREN usando o GREN-Wizard. Na seção 5.8 é feita a generalização do processo, mostrando como construir um *Wizard* genérico, que pode ser adaptado de acordo com cada par "framework e linguagem de padrões". Na seção 5.9 são apresentadas as conclusões sobre os dois processos propostos.

5.2 Alternativas para Construção de um *Wizard*

Em geral, a construção de uma ferramenta para automatizar a instanciação de frameworks é uma tarefa totalmente direcionada às características específicas do framework. Como a implementação da ferramenta fica totalmente atrelada ao framework, muitas vezes o custo de sua construção somente se justifica se o framework for instanciado para um número elevado de aplicações. No entanto, as peculiaridades do framework construído segundo o processo proposto nesta tese, torna-o distinto de outros frameworks, possibilitando que a construção de uma ferramenta para automatizar sua instanciação seja feita por um processo bem definido e cujo resultado pode ser reutilizado para outros frameworks.

O processo para construção do *Wizard* pode variar dependendo da abrangência desejada para a ferramenta obtida, conforme ilustrado na Figura 5.1. Duas alternativas podem ser seguidas, que são denominadas alternativas "a" e "b". Quando se pretende utilizar o *Wizard* na instanciação de um único par "linguagem de padrões e framework" (por exemplo, um framework que é utilizado intensamente na geração de inúmeras aplicações do domínio), então utiliza-se a alternativa "a", na qual faz-se o projeto do *Wizard* sob medida, ou seja, totalmente personalizado para o framework e sua linguagem de padrões associada. Isso torna o *Wizard* menos flexível em termos de facilidade de adaptação a outros frameworks, mas, por outro lado, torna-o mais fácil de construir e usar. Quando se pretende especializar o *Wizard* para uso com várias linguagens de padrões e frameworks, então é desejável que o processo de construção considere as devidas generalizações que devem ser feitas para permitir sua adaptação e uso com diferentes pares "framework e linguagem de padrões", o que constitui a alternativa "b".

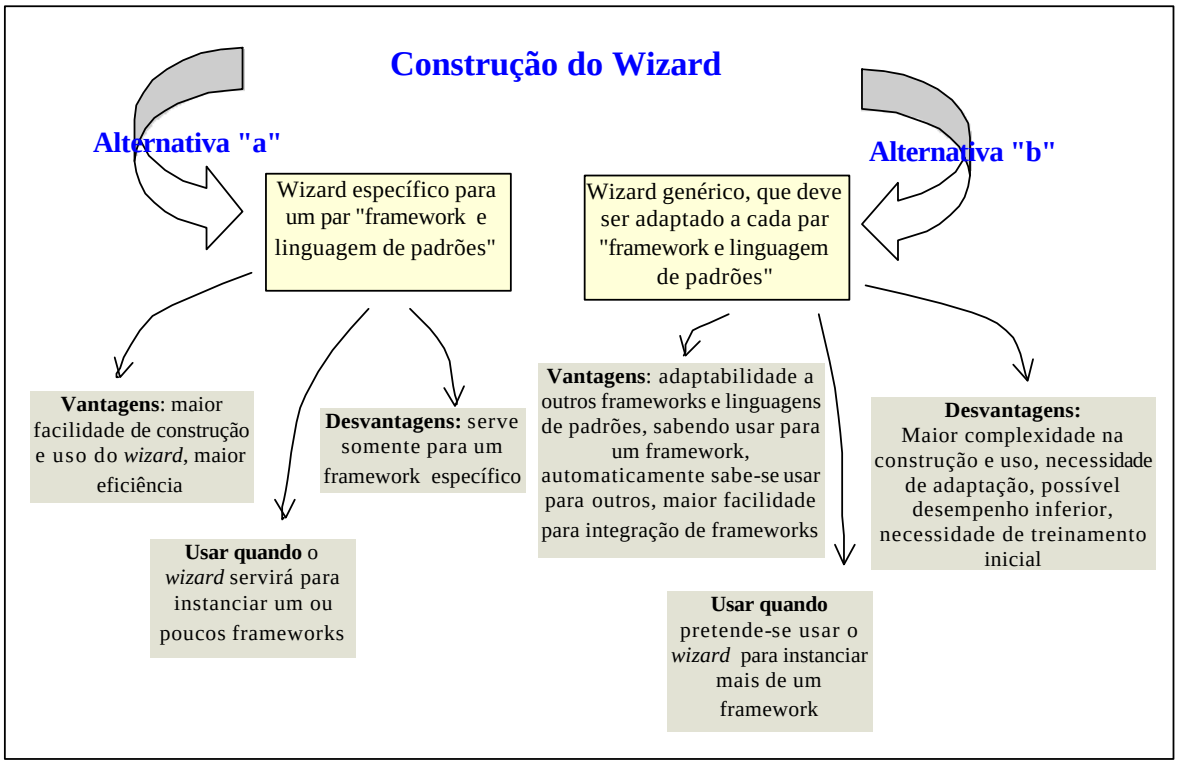


Figura 5.1: Alternativas possíveis para construção de um *Wizard*

Neste trabalho exploram-se as duas alternativas, começando pela alternativa “a” (seção 5.4), que é a construção de um *Wizard* específico para um framework e uma linguagem de padrões, seguida da alternativa “b” (seção 5.8), na qual mostra-se como construir um *Wizard* genérico, que deve ser adaptado para cada par “framework e linguagem de padrões”. A segunda alternativa é explorada em maior nível de detalhamento, por ser mais completa e incluir a primeira. O GREN-Wizard, utilizado para ilustrar o processo de construção, teve um protótipo construído de forma *ad hoc*, que está mais próximo da alternativa “a”. Depois ele foi remodelado usando um processo similar à alternativa “b”. O processo proposto foi elaborado de acordo com a experiência adquirida na construção do GREN-Wizard, bem como da observação das características desejáveis em um *Wizard* genérico. No entanto, para que esse processo possa ser validado, novos *Wizards* devem ser construídos usando-o como base.

5.3 Arquitetura do *Wizard*

A Figura 5.2 mostra a proposta de arquitetura do *Wizard* para instanciação de um framework baseado em uma linguagem de padrões (Braga e Masiero, 2002a). Dois atores interagem diretamente com ele: o engenheiro do domínio, responsável pela construção do *Wizard* (e, provavelmente,

também pelo desenvolvimento da linguagem de padrões e por parte da construção do framework) e o engenheiro de aplicações, que utiliza o *Wizard* para gerar aplicações específicas. O usuário final utiliza o *Wizard* de forma indireta, já que executa o código por ele gerado.

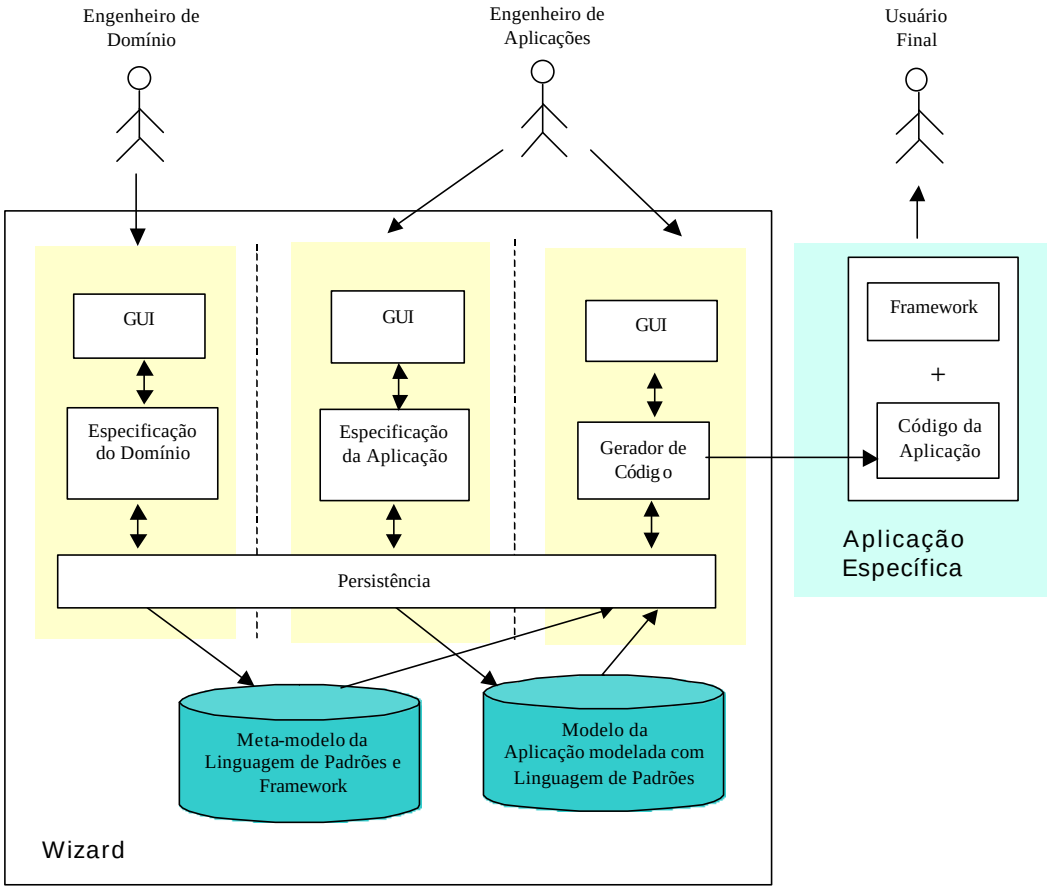


Figura 5.2: Arquitetura do *Wizard* para instanciação de um framework baseado em uma linguagem de padrões

Uma interface gráfica com o usuário (GUI) é disponibilizada para permitir a interação entre os atores e os módulos do *Wizard*. Três módulos fazem parte dele: o módulo de especificação do domínio, o módulo de especificação da aplicação e o módulo de geração de código. O módulo de especificação do domínio tem o propósito de permitir a representação de informações sobre a linguagem de padrões e seu mapeamento para as classes do framework. O módulo de especificação da aplicação é responsável por armazenar as informações sobre a modelagem das aplicações concretas usando a linguagem de padrões. Por fim, o módulo de geração de código baseia-se nas informações disponibilizadas sobre a especificação do domínio e sobre a especificação da aplicação, para gerar o código específico da aplicação, que deve juntar-se ao código do framework para produzir a aplicação a ser entregue ao usuário final.

A arquitetura apresentada na Figura 5.2 é genérica, sendo adequada tanto para *Wizards* construídos segundo a alternativa "a", quanto a alternativa "b" da Figura 5.1. Em ambos os casos há necessidade de implementar os três módulos propostos, embora no caso do *Wizard* específico existam opções menos complexas para implementação dos diversos módulos. Por exemplo, o módulo de interface gráfica correspondente à especificação da aplicação pode ser implementado por diversos formulários de entrada de dados, cada um relativo a um padrão específico. Nesses formulários, os campos a serem preenchidos pelo engenheiro de aplicações podem ser fixados em tempo de compilação, o que é mais fácil de implementar. No entanto, se o *Wizard* for genérico, esses formulários devem ser criados dinamicamente, de acordo com informações vindas da base de dados que contém a especificação da linguagem de padrões.

5.4 Construção de um *Wizard* específico

O processo de construção de um *Wizard* específico para um framework e sua linguagem de padrões segue os passos mostrados na Figura 8.10. Para cada padrão da linguagem de padrões, deve-se permitir que o usuário o aplique (ou a quaisquer de seus variantes), indicando os papéis desempenhados por cada classe da aplicação. Informações sobre o uso dos padrões na modelagem da aplicação específica devem ficar armazenadas para uso futuro, seja para construção de aplicações com base em outras similares já registradas, seja para alteração da especificação de uma aplicação para fins de reengenharia ou manutenção. A seguir descrevem-se os passos envolvidos nesse processo.

Passo 3a.1. Criar o modelo da linguagem de padrões

Com base na linguagem de padrões específica, cria-se um modelo que representa todos os seus elementos, principalmente os que forem necessários para possibilitar a instanciação de aplicações específicas. São eles: os padrões contidos na linguagem e seus possíveis variantes, as classes participantes de cada padrão/variante e os métodos e operações de cada padrão/variante. Esse modelo deve ser implementado pelo *Wizard*, por exemplo, por meio de uma base de dados contendo essas informações. Pode-se desenvolver um sub-sistema para registrar essas informações mas, como o *Wizard* é específico, deve-se considerar o custo/benefício de fazê-lo, pois é provável que essas informações sejam registradas uma única vez. Caso opte-se por não construir esse sub-sistema, pode-se entrar com os dados diretamente na base de dados, utilizando um SGBD.

Passo 3a.2. Criar o mapeamento da linguagem de padrões para o framework

Com base no framework associado à linguagem de padrões (considera-se, aqui, seu código-fonte, manuais técnicos e manuais de uso), cria-se um modelo que mapeia os padrões da linguagem de

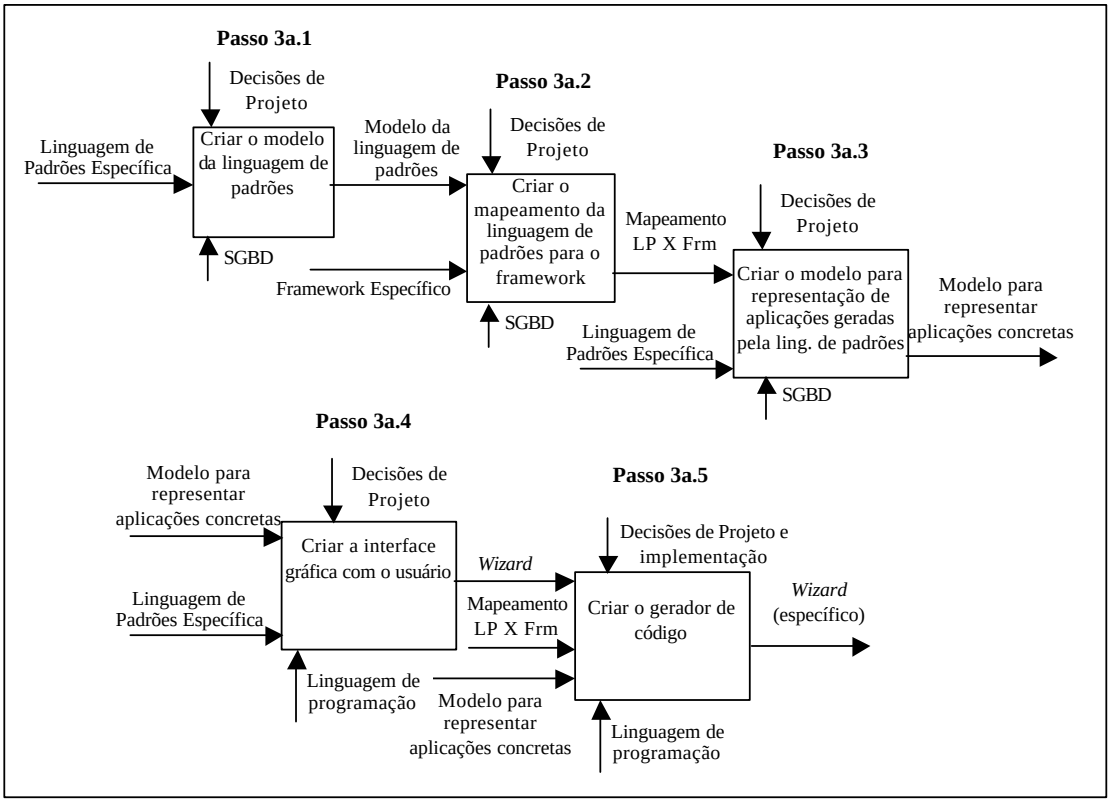


Figura 5.3: Processo para construção de um *Wizard* específico para instanciação de um framework baseado em uma linguagem de padrões

padrões (juntamente com seus variantes e participantes) às classes do framework que os implementam. Dois tipos principais de informação são desejáveis: quais são as super-classes do framework que devem ser especializadas na nova aplicação, de acordo com os padrões e variantes utilizados, e quais são os métodos que devem ser sobrepostos nas classes recém-criadas. Esse modelo deve ser implementado de forma que possa ser recuperado posteriormente pelo gerador de código, que irá criar as novas classes e seus métodos. Assim como no passo 3a.1, pode-se desenvolver um subsistema para permitir o registro desse mapeamento, mas é provável que o custo não se justifique, já que o mapeamento é incluído uma única vez [1].

Passo 3a.3. Criar o modelo para representação de aplicações geradas pela linguagem de padrões

O *Wizard* tem como objetivo instanciar aplicações específicas com base na linguagem de padrões e no framework. Assim, deve-se permitir que uma aplicação específica seja representada em termos

[1] no entanto, pode-se fazer alterações no mapeamento durante o ciclo de vida do framework, de acordo com sua evolução

de quais padrões/variantes foram usados em sua modelagem, bem como os papéis desempenhados pelas classes da aplicação nos padrões da linguagem de padrões. Essa informação deve ficar disponível para que o módulo de geração de código possa criar as devidas classes e métodos. Assim, deve-se criar um modelo capaz de suprir informações tais como: a) dada uma aplicação, quais padrões/variantes foram utilizados para modelá-la (e em que ordem foram aplicados); b) quais os papéis desempenhados por cada classe participante (mantendo-se a ligação com o respectivo padrão e variante); e c) quais atributos, não presentes nos padrões, foram incluídos na nova aplicação.

Passo 3a.4. Criar a interface gráfica com o usuário

É importante que o *Wizard* possua uma interface gráfica com o usuário (GUI) correspondente ao módulo de especificação da aplicação, para facilitar que o engenheiro de aplicações entre com as informações sobre o uso da linguagem de padrões na modelagem de sua aplicação específica. Os requisitos desejáveis para essa GUI são listados no passo 3b.1.4 da seção 5.8.

Passo 3a.5. Criar o módulo de geração de código

Com base nas informações sobre a aplicação específica, o *Wizard* deve ser capaz de gerar o código de suas classes e sobrepor os métodos necessários (pelo menos deve-se fornecer uma lista de tais métodos, para que o engenheiro de aplicações possa, com base nessa lista, criar os métodos manualmente). Algumas diretrizes são fornecidas no passo 3b.1.5 (seção 5.8) para facilitar a criação desse módulo.

Como nesta seção está sendo explorada a alternativa “a” da Figura 5.1 para construção do *Wizard*, então existe a possibilidade de otimizar algumas funções de acordo com as características específicas do framework e da linguagem de padrões, já que a implementação não fica presa a uma arquitetura genérica, como é o caso de *Wizards* construídos segundo o processo da seção 5.8.

Deve-se ressaltar que o processo de criação de um *Wizard* específico foi apenas delineado de maneira superficial nesta seção pois, para evitar redundância, optou-se por abordar com maior nível de detalhes o processo de criação de um *Wizard* genérico (seção 5.8), cujas atividades incluem as atividades aqui descritas.

5.5 Exemplo – GREN-Wizard: Uma ferramenta de suporte ao GREN e à GRN

5.5.1 Visão Geral do GREN-Wizard

O GREN-Wizard (Braga e Masiero, 2002c) é uma ferramenta para auxiliar a instanciação do framework GREN (descrito na seção 4.3), com base na linguagem de padrões GRN (descrita na seção 3.5). 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. Ele foi implementando usando o ambiente VisualWorks 5i.4, com dados persistidos em base de dados MySQL, possuindo 30 classes e aproximadamente vinte mil linhas de código Smalltalk. A hierarquia de classes do GREN-Wizard é apresentada no Apêndice F.

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, das classes especializadas a partir do GREN. Ele cria, também, as tabelas na base de dados MySQL, 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.

5.5.2 Interface Gráfica do GREN-Wizard

A Figura 5.4 mostra um exemplo da GUI do GREN-Wizard. Os princípios utilizados em sua construção foram generalizados e são apresentados como requisitos da GUI do *Wizard* genérico, no passo 3b.1.4 da seção 5.8.

O GREN-Wizard foi originalmente desenvolvido no idioma inglês e, depois, adaptado por um pacote de internacionalização do VisualWorks, que permitiu a escolha da língua desejada. Atualmente ele funciona para Português e Inglês. Entretanto, algumas das tabelas que armazenam

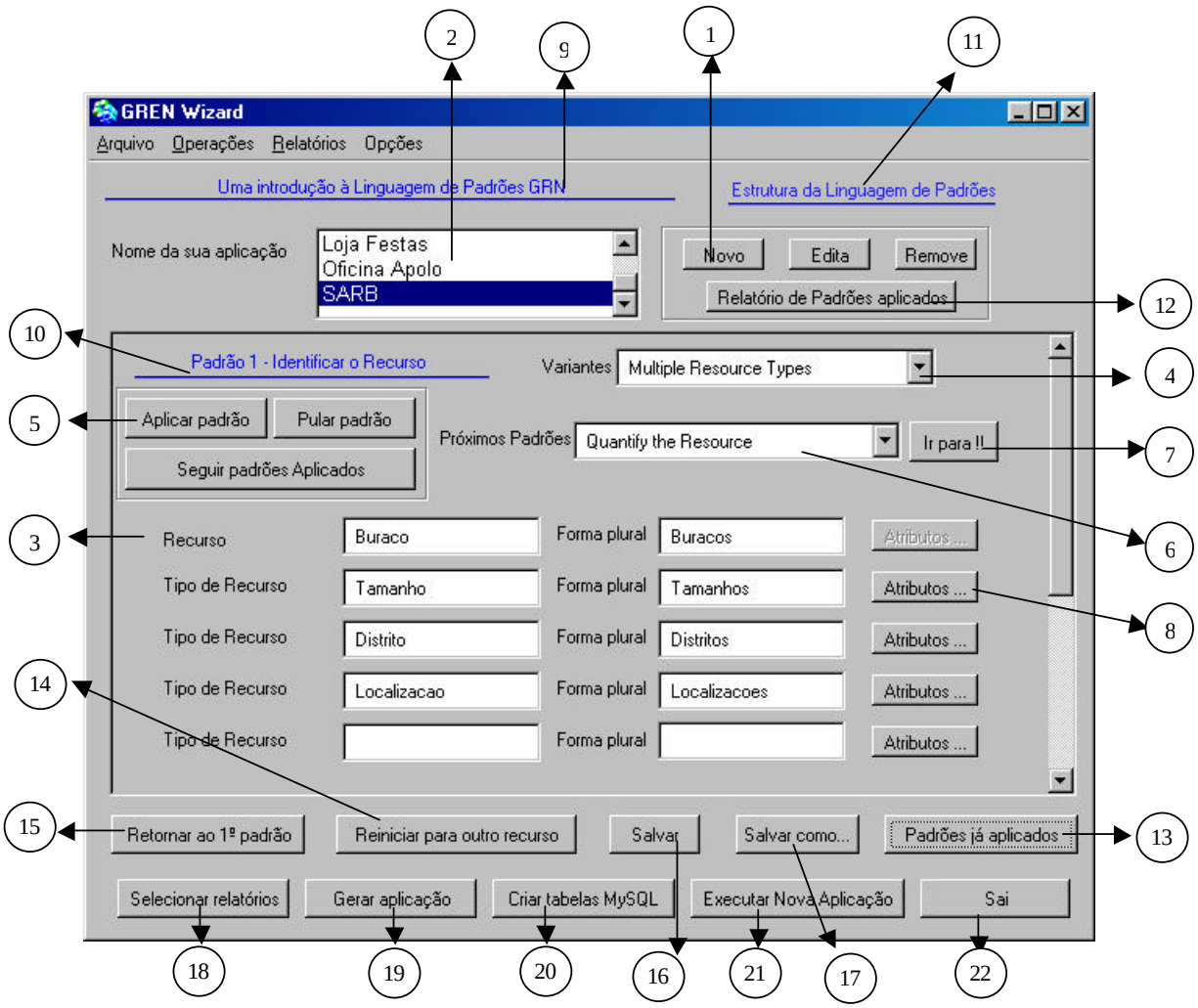


Figura 5.4: Exemplo da GUI do GREN-Wizard

os meta-dados sobre os padrões ainda se encontram somente em inglês. Por isso, alguns conteúdos de listas aparecem em inglês na Figura 5.4 (por exemplo o #4 e o #6).

Novas aplicações podem ser criadas por meio do botão “Novo” (veja #1 na Figura 5.4). A nova aplicação é incluída na caixa com as aplicações existentes (#2). O primeiro padrão da GRN aparece então em uma porção da GUI especial para mostrar o padrão sendo aplicado no momento (#3). Começando pelo primeiro padrão do histórico de padrões aplicados (obtido durante a aplicação da GRN), as classes participantes de cada padrão devem ser preenchidas. O variante do padrão (#4) deve ser escolhido adequadamente, pois as classes participantes mudam dinamicamente de acordo com tal escolha. É importante assegurar que todos os participantes do padrão, requeridos na aplicação específica, estejam presentes no variante escolhido. O botão “Atributos” (#8) pode ser usado para mudar nomes de atributos da classe ou para adicionar novos atributos.

Após preencher todos os participantes do padrão/variante, o botão “Aplicar Padrão” (#5) é usado para salvar o padrão corrente em uma lista de padrões aplicados. Em seguida, deve-se selecionar o próximo padrão a aplicar (por meio da caixa #6) e o botão “Ir para” (#7) deve ser usado para que o GREN-Wizard adapte sua GUI de acordo com as características do próximo padrão escolhido. Esse procedimento deve ser seguido até que o último padrão da lista de padrões aplicados tenha sido preenchido.

Durante a navegação e uso do GREN-Wizard pode-se, a qualquer momento, usar o botão “Padrões já aplicados” (#13) para ver a especificação corrente dos padrões aplicados até o presente momento. Esse botão mostra somente os padrões realmente aplicados até então, mas existe um outro botão, o “Relatório de Padrões aplicados” (#12), que mostra a especificação tal qual gravada na base de dados (esse botão é específico para ver os padrões usados na modelagem dos sistemas já existentes na base de dados).

O GREN-Wizard também oferece ao usuário textos explicativos sobre a GRN, caso ele não esteja com o documento da GRN em mãos. Por exemplo, o botão #9 mostra um texto introdutório; o botão #10 mostra o conteúdo do padrão atualmente selecionado; e o botão #11 mostra o grafo da GRN.

Algumas aplicações exigem que a GRN seja aplicada em diversas iterações, provavelmente para gestão de diferentes recursos ou para diferentes tipos de gestão de um mesmo recurso. Para permitir essa funcionalidade, o GREN-Wizard possui dois botões: “Reiniciar para outro Recurso” (#14) e “Retornar ao primeiro padrão” (#15). Na verdade, o botão #14 é específico do GREN-Wizard, enquanto o botão #15 é mais genérico, ou seja, poderia ser utilizado em *Wizards* para outras linguagens de padrões e frameworks.

Antes de gerar as classes da nova aplicação, o GREN-Wizard exige que a especificação da aplicação seja salva permanentemente na base de dados, o que é possível pelo uso de um dos botões: “Salvar” (#16) ou “Salvar como...” (#17). Uma atividade opcional é a especificação dos relatórios que farão parte do menu contido na janela principal da aplicação, o que pode ser feito por meio do botão “Selecionar Relatórios” (#18).

A GUI do GREN-Wizard integra-se ao módulo de geração de código por meio dos botões “Gerar Código”, “Criar tabelas MySQL” e “Executar Aplicação”. O primeiro (#19) ativa a geração automática de todas as classes da nova aplicação, inclusive sobrepondo os métodos necessários. O segundo (#20) gera um *script* para criação das tabelas MySQL e executa esse *script*, de modo que o uso do terceiro botão (#21) já ativa a janela principal da aplicação e permite sua execução imediata. Finalmente, o botão #22 é utilizado para fechar o GREN-Wizard.

5.5.3 Arquitetura do GREN-Wizard

A Figura 5.5 mostra a arquitetura do GREN-Wizard. Nota-se que ela possui algumas simplificações em relação à arquitetura geral proposta na Figura 5.2, pois o processo de construção do GREN-Wizard foi *ad hoc*. Assim, a arquitetura geral proposta na seção 5.3 foi concebida posteriormente, com base na experiência e visão ampliada proporcionadas pela construção do GREN-Wizard.

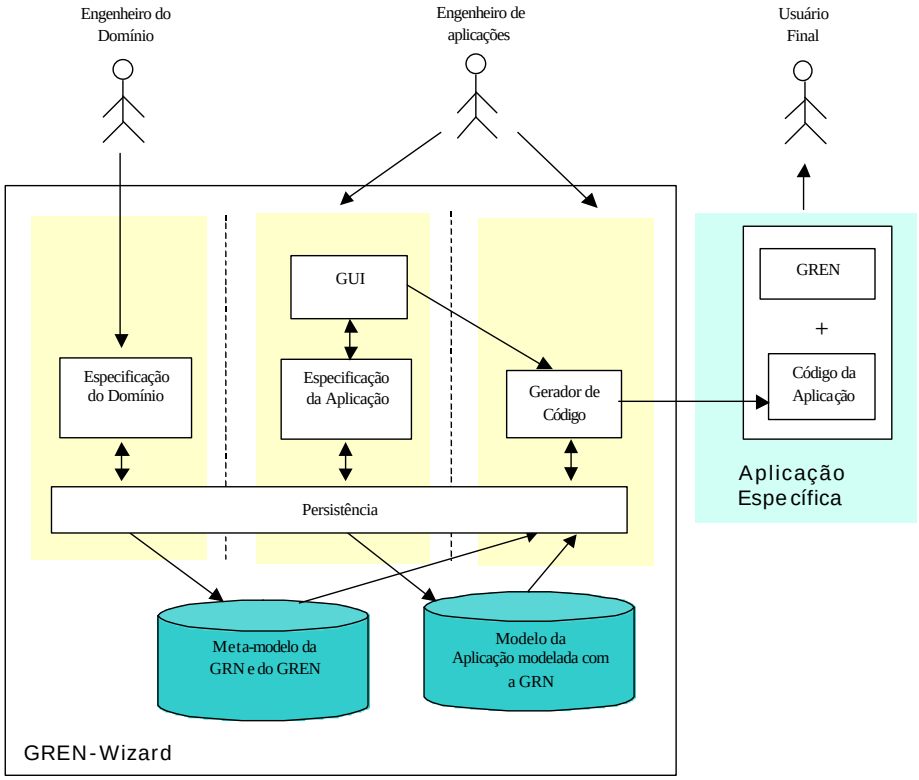


Figura 5.5: Arquitetura do GREN-Wizard

Por conseqüência, o GREN-Wizard não possui o módulo GUI referente à especificação de domínio (para entrada de dados sobre a modelagem da linguagem de padrões e do mapeamento para o framework) e a GUI do gerador de código encontra-se integrada à GUI do módulo de especificação da aplicação (por meio de botões que invocam os métodos daquele módulo). Os diversos módulos que compõem o GREN-Wizard são descritos a seguir.

Módulo de Especificação do Domínio

As classes pertencentes ao módulo de especificação do domínio do GREN-Wizard são mostradas na Figura 5.6[2]. Uma linguagem de padrões é composta de diversos padrões (classe *Pattern*) que possuem um número e um nome. Durante o uso da linguagem de padrões, um padrão pode ser obrigatório, isto é, seu uso é requerido pois a funcionalidade por ele coberta está presente em todas as aplicações do domínio, ou pode ser facultativo. Os padrões possuem um auto-relacionamento que indica a ordem na qual são aplicados (relacionamento *nextPatterns*), por exemplo, na GRN, após a aplicação do padrão 2, os padrões 3, 4, 6 ou 9 podem ser aplicados.

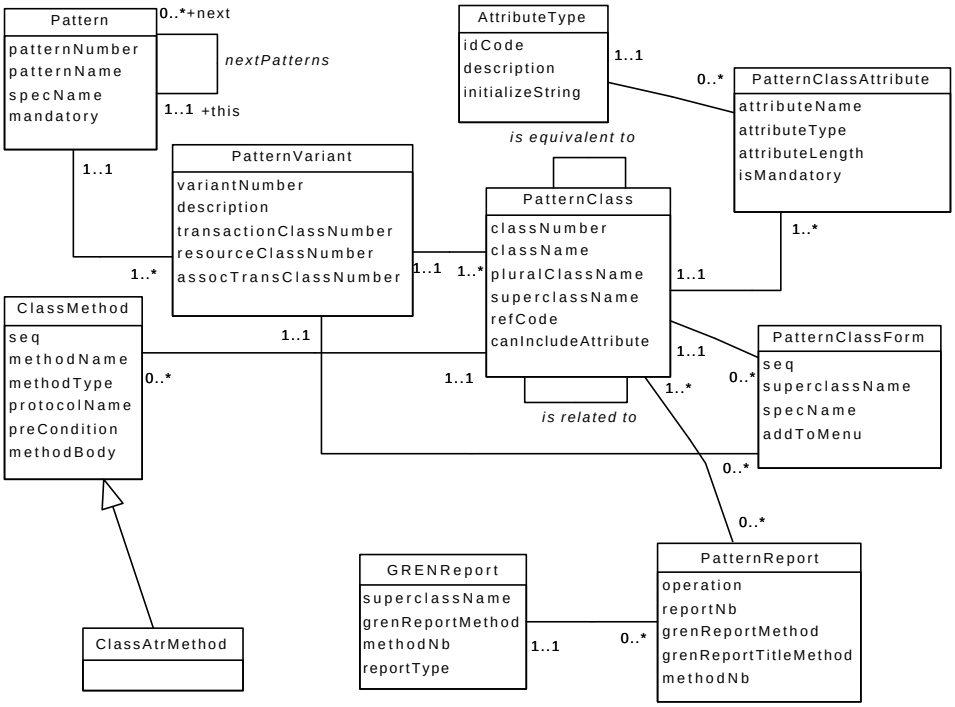


Figura 5.6: Diagrama de classes do módulo de especificação do domínio da GRN

No modelo da Figura 5.6, cada padrão possui pelo menos um variante (classe *PatternVariant*), que é o seu *default*. As classes pertencentes ao padrão estão diretamente associadas ao variante (classe *PatternClass*), já que variantes distintos podem ter diferentes classes participantes. Uma classe possui um nome e sua forma plural (utilizada nas classes da GUI). Algumas classes permitem a inclusão de novos atributos, enquanto outras não o permitem. Uma classe pode ter vários atributos (classe *PatternClassAttribute*), obrigatórios ou não, e vários métodos. Os atributos pos-

[2]Esse diagrama, bem como os diagramas das Figuras 5.7, 5.8 e 5.9, refletem a atual implementação do GREN-Wizard e, por isso, estão no idioma inglês, que foi utilizado para nomear classes, atributos e métodos, de forma a facilitar a comunicação com pesquisadores estrangeiros.

suem um tipo (por exemplo, inteiro, número decimal, caracter, data, etc.), denotado pela classe *AttributeType*.

Dois tipos de auto-relacionamentos são permitidos entre classes: equivalência (relacionamento *is equivalent to*), para o caso no qual a mesma classe aparece em diferentes padrões, mas com o mesmo significado, e relacionamento (relacionamento *is related to*), para denotar associações, agregações ou especializações de classes. A classe *ClassAtrMethod* contém os métodos a serem sobrepostos nas classes da aplicação para as quais foram acrescentados novos atributos durante a instanciação, por meio do botão #8 da Figura 5.4.

Além dos aspectos acima descritos, que referem-se à especificação de quaisquer linguagens de padrões[3], o modelo da Figura 5.6 possui classes e atributos peculiares à GRN e ao GREN, que são explicados a seguir:

- O atributo `specName` da classe *Pattern* retorna, para cada padrão, o nome do método de especificação do canvas contendo o texto explicativo sobre o padrão, a ser mostrado quando o usuário pressiona um botão na GUI do GREN-Wizard (ver o botão #10 na Figura 5.4).

- Os atributos `transactionClassNumber`, `resourceClassNumber` e `assocTransClassNumber` da classe *PatternVariant* denotam o número da classe participante do padrão relativa à transação, recurso de negócios e transação associada, respectivamente. A necessidade desses atributos deveu-se à seguinte característica da GRN: durante a aplicação dos padrões, deve-se ter sempre em mente qual é o recurso sendo gerenciado, qual é a transação principal sendo efetuada com o recurso e, no caso de transações dependentes umas das outras, qual é a transação associada. Essas informações são necessárias para o correto funcionamento do histórico de padrões aplicados mantido na base de dados, já que um padrão pode ser aplicado várias vezes para o mesmo recurso, mas para transações diferentes.

- O atributo `initializeString` da classe *AttributeType* cuida do fato de, no GREN, os objetos serem iniciados pelo método construtor da classe. Assim, quando novos atributos são adicionados às classes da aplicação, é necessário que o método `initialize` seja gerado automaticamente para esses atributos.

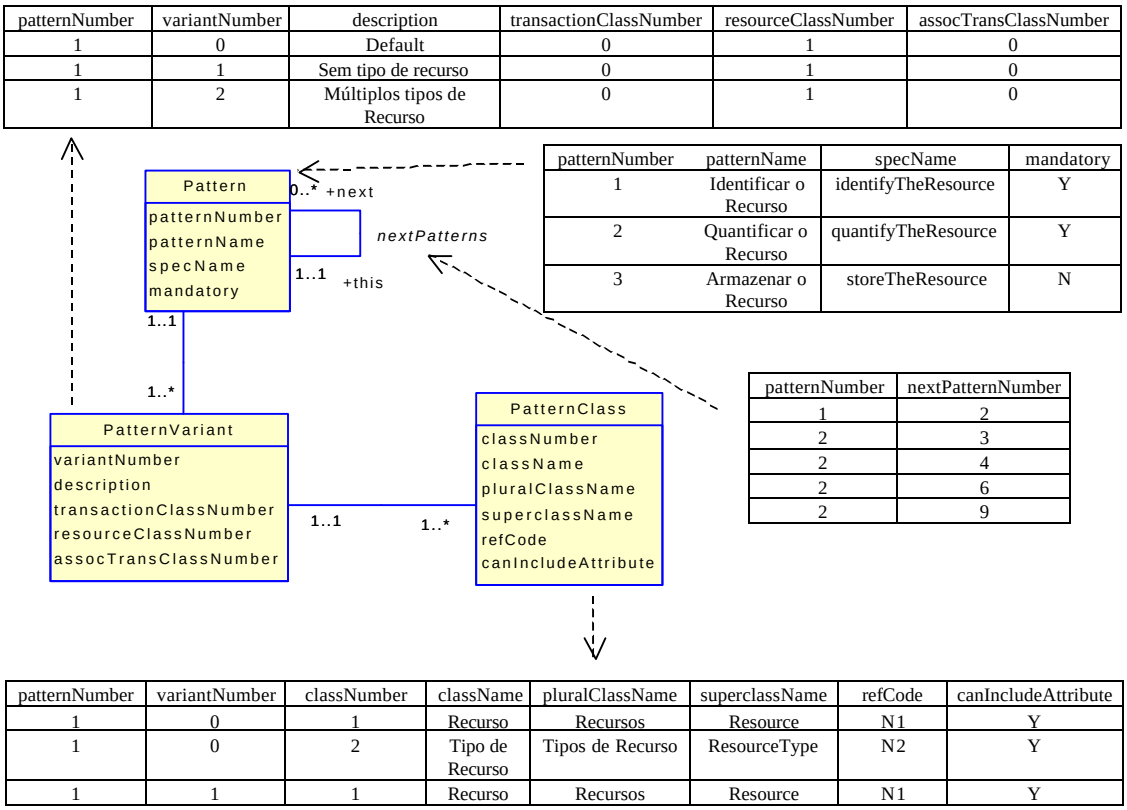
- O atributo `protocolName` da classe *ClassMethod* indica o nome do protocolo no qual o método deve ser inserido, pois o ambiente VisualWorks Smalltalk agrupa os métodos em protocolos. Assim, o gerador de código precisa saber o nome do protocolo para ser capaz de gerar o método no local correto dentro do corpo da classe.

[3] o modelo geral proposto na seção 5.8 baseia-se nesses aspectos

- A classe *PatternClassForm* contém as classes da GUI correspondentes a cada classe da camada de negócios do GREN. Uma classe de negócios do GREN pode ter zero ou mais classes correspondentes na GUI.

- As classes *PatternReport* e *GRENReport* contém os relatórios (operações de saída na GRN) correspondentes a cada classe da camada de negócios do GREN. Uma classe de negócios do GREN pode ter zero ou mais relatórios, que podem ser incluídos no menu da aplicação por meio de um utilitário do GREN-Wizard (ver #18 da Figura 5.4).

A persistência dos objetos do modelo da Figura 5.6 foi feita utilizando-se a base de dados relacional MySQL. Como o GREN-Wizard não possui o módulo GUI para entrada de dados do domínio, esses dados são incluídos diretamente nas tabelas da base de dados usando scripts com os comandos SQL. Exemplos do conteúdo de algumas tabelas dessa base de dados podem ser vistos na Figura 5.7.



patternNumber	variantNumber	description	transactionClassNumber	resourceClassNumber	assocTransClassNumber
1	0	Default	0	1	0
1	1	Sem tipo de recurso	0	1	0
1	2	Múltiplos tipos de Recurso	0	1	0

patternNumber	patternName	specName	mandatory
1	Identificar o Recurso	identifyTheResource	Y
2	Quantificar o Recurso	quantifyTheResource	Y
3	Armazenar o Recurso	storeTheResource	N

patternNumber	nextPatternNumber
1	2
2	3
2	4
2	6
2	9

patternNumber	variantNumber	classNumber	className	pluralClassName	superclassName	refCode	canIncludeAttribute
1	0	1	Recurso	Recursos	Resource	N1	Y
1	0	2	Tipo de Recurso	Tipos de Recurso	ResourceType	N2	Y
1	1	1	Recurso	Recursos	Resource	N1	Y

Figura 5.7: Exemplo de Conteúdo de algumas tabelas de especificação do domínio da GRN

Módulo de Especificação da Aplicação

As classes pertencentes ao módulo de especificação da aplicação do GREN-Wizard são mostradas na Figura 5.8. Caixas na cor cinza indicam que as mesmas pertencem ao modelo da especificação do domínio da GRN (Figura 5.6), porque há um relacionamento entre as classes da aplicação e as classes do padrão. Cada objeto da classe *GRNApplication* representa um sistema particular instanciado usando o framework e a linguagem de padrões.

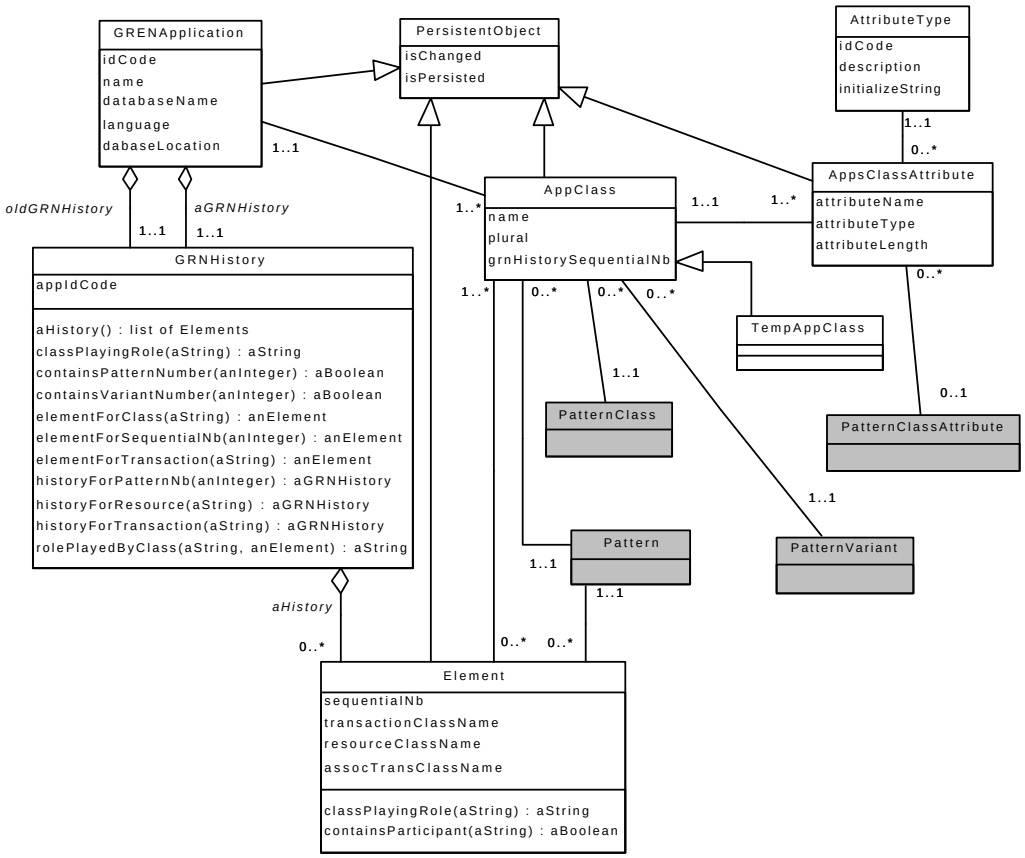


Figura 5.8: Diagrama de classes do módulo de especificação do domínio da GRN

A classe *GRNHistory* representa o histórico de padrões utilizados na modelagem da aplicação e a classe *Element* representa cada padrão/variante desse histórico, na ordem correta na qual foi aplicado. A classe *GRNHistory* dispõe de métodos para obter as classes que desempenham um papel específico em uma determinada aplicação, para determinar se uma aplicação utilizou um padrão em particular e para extrair o histórico de aplicação de um determinado padrão, entre outros. Como um elemento representa a aplicação de um padrão, ele está relacionado a um padrão e, também, a uma ou mais classes da aplicação (*AppClass*) que compõem esse padrão. A classe *AppClass* relaciona-se a uma classe específica de um variante de um padrão. A classe *AppClassAttribute*

representa os atributos específicos de uma classe da aplicação, que pode relacionar-se a atributos do padrão (podem também ter sido incluídos durante a instanciação e, por isso, o relacionamento é opcional).

Assim como ocorre no módulo de especificação do domínio, neste módulo também pode-se observar algumas características particulares do framework GREN e da linguagem de padrões GRN. Na classe *Element* existem os atributos `resourceClassName`, `transactionClassName` e `assocTransClassName`, que facilitam a identificação do recurso de negócios, da transação e da transação associada relativas ao elemento corrente, respectivamente. Na classe *GRNHistory* tem-se o método `historyForResource`, que retorna o histórico de padrões aplicados para um determinado recurso de negócios. Os demais participantes desse modelo são genéricos e poderiam ser utilizados mesmo que o *Wizard* fosse implementado para outra linguagem de padrões ou framework. Assim, quando é feita, na seção 5.8, a generalização do modelo da Figura 5.8, a descrição do modelo não é repetida, por ser similar à descrição aqui apresentada.

Módulo da GUI referente à especificação da aplicação

O módulo GUI (ver Figura 5.5) foi criado para permitir a interação entre o desenvolvedor de aplicações e o módulo de especificação da aplicação. A Figura 5.9 mostra o diagrama de classes desse módulo[4]. Sua função é de coletar os dados sobre a aplicação específica por meio de formulários de entrada de dados, interagindo com o módulo de especificação da aplicação que processa esses dados e envia para a base de dados.

As classes destacadas em amarelo pertencem ao framework GREN, a classe em verde pertence ao VisualWorks Smallltalk, a classe em cinza pertence ao módulo de especificação da aplicação e as classes em rosa pertencem ao módulo de geração de código. A classe *GRENApplicationForm* corresponde ao formulário para inclusão de aplicações modeladas com o GREN-Wizard. A classe *GRENWizardGUI* representa o principal formulário do GREN-Wizard, que é a janela mostrada na Figura 5.4. Ela possui uma sub-janela (em inglês, *subcanvas*) representada pela classe *PatternForm*, que é montada em tempo de execução para mostrar cada um dos quinze padrões da GRN (ver #3 na Figura 5.4). Mantém referências para a aplicação que está sendo incluída ou alterada no momento (`app`) e para todas as aplicações registradas na base de dados (`appsList`), que são mostradas na caixa de aplicações existentes (#2 da Figura 5.4). Alguns de seus atributos são específicos da GRN: `transName`, `transPlName`, `resName`, `resPlName` e `assocTransName` são necessários para que as iterações pela linguagem de padrões ocorram adequadamente, guardando sempre a informação de quem é o recurso sendo gerenciado, qual a transação de gestão corrente e sua transação associada.

[4]na verdade esse diagrama contém também as classes do módulo de geração de código (na cor rosa) descritas logo a seguir

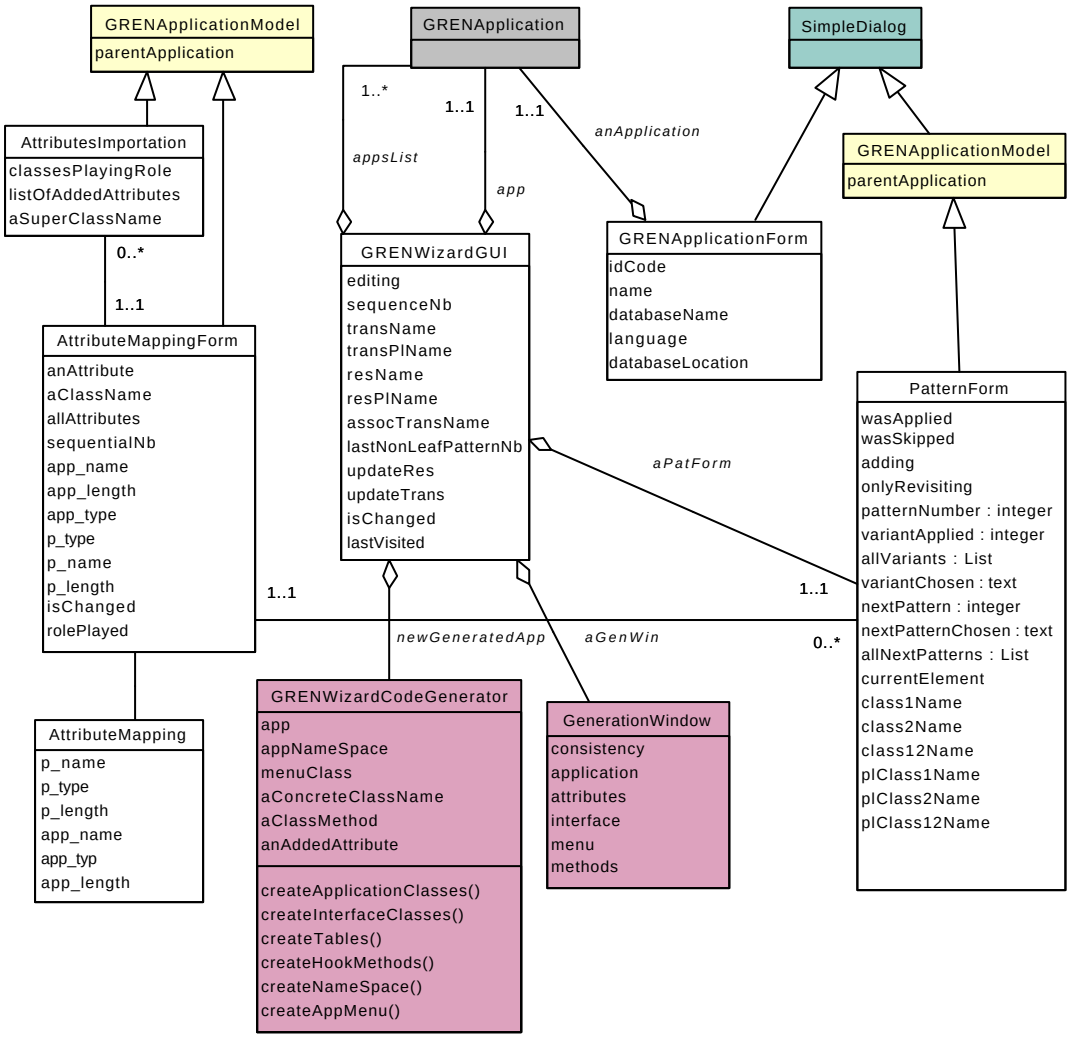


Figura 5.9: Diagrama de classes do módulo GUI do GREN-Wizard

A classe *AttributeMappingForm* representa o formulário para entrada ou alteração dos atributos das classes, enquanto a classe *AttributesImportation* representa o formulário executado quando deseja-se importar a definição dos atributos a partir de outras aplicações similares. A classe *AttributeMapping* é temporária, sendo utilizada para manter valores de atributos mapeados a partir dos padrões.

Módulo Gerador de Código

O módulo gerador de código é composto de apenas duas classes: *GRENWizardCodeGenerator* e *GenerationWindow* da Figura 5.9. A classe *GRENWizardCodeGenerator* é responsável por percorrer o histórico de padrões e variantes aplicados na modelagem de um sistema específico e

gerar as classes em Smalltalk e o *script* SQL para criar as tabelas da base de dados relacional. A classe *GenerationWindow* é utilizada para mostrar, na janela principal do GREN-Wizard, o andamento do processo de geração de código, para que o usuário saiba quais atividades já foram processadas e quais ainda estão previstas.

Para permitir a geração automática das classes da aplicação, a informação sobre o mapeamento entre a GRN e o GREN, conforme ilustrado nas tabelas 4.2 e 4.3 da seção 4.3, foi transferida para tabelas MySQL. Assim, o módulo gerador de código é capaz de identificar quais classes devem ser especializadas de acordo com o histórico de padrões e variantes aplicados.

O algoritmo da Figura 5.10 ilustra como as classes da aplicação são criadas pelo GREN-Wizard. No passo 1 executa-se um laço para todos os elementos que pertencem ao histórico de padrões aplicados na modelagem do sistema instanciado. No passo 1b outro laço mais interno é executado para todas as classes da aplicação pertencentes a esses elementos. Cada classe da aplicação pertence a um ou mais padrões e corresponde a uma de suas classes participantes. A tabela de mapeamento (4.2) informa a classe correta do GREN a especializar (variável `aSuperClass`), a partir da classe participante, padrão e variante (passos 1(b)i,1(b)ii e 1(b)iii). Tendo essa informação, uma nova classe é criada, herdando da classe *aSuperClass* (obtida no passo 1(b)iv), cujo nome é o próprio nome da classe da aplicação (passo 1(b)v). Existem casos nos quais a Tabela 4.2 retorna uma super-classe nula. Isso pode ser devido a duas razões: a) o framework não precisa ser especializado para oferecer essa funcionalidade ou b) essa classe desempenha dois ou mais papéis em diferentes padrões e, provavelmente, já foi criada em um padrão anterior ou somente poderá ser criada após a aplicação de um padrão posterior.

Analogamente, foram criadas tabelas MySQL para armazenar informação sobre os métodos que devem ser sobrepostos nas classes recém-criadas. Por exemplo, a Tabela 4.4 da seção 4.3 mostra alguns métodos-gancho do GREN. O algoritmo da Figura 5.11 ilustra como os métodos-gancho são sobrepostos pelo GREN-Wizard. No passo 1 executa-se um laço para todas as classes da aplicação criadas durante a instanciação. No passo 1b percorrem-se todos os métodos-gancho definidos na base de dados (classe *ClassMethod* da Figura 5.6) para verificar se eles precisam ser sobrepostos na classe da aplicação. Isso é feito averiguando se a nova classe da aplicação herda (não necessariamente diretamente) da classe na qual o método está definido e se a pré-condição para sobreposição do método é satisfeita (passo 1(b)iii). Caso ambos sejam verdadeiros, o método é criado na nova classe. Um exemplo de pré-condição para sobreposição de um método é dado na coluna "Comentário" da Tabela 4.4. Na base de dados que contém os métodos a serem sobrepostos (classe *ClassMethod*) existem duas colunas adicionais para facilitar a criação automática dos métodos: uma contém o nome do método responsável por checar a pré-condição (`preCondition`) e a outra contém o nome do método do GREN-Wizard responsável por criar o corpo do método e

1. Para todos os elementos pertencentes ao histórico de padrões aplicados faça:
   (a) Seja `e` := elemento corrente que representa o padrão aplicado.
   (b) Para todas as classes específicas relativas a `e` faça:
      i. Seja `c` := classe específica relacionada a `e`.
      ii. Seja `p` := padrão relacionado a `c`.
      iii. Seja `v` := variante relacionado a `c`.
      iv. Seja `s` := super-classe do framework que corresponde à tupla: {`c`, `p`, `v`}.
      v. Se `s` não é nula então crie a classe `c` com `s` como sua super-classe. Caso contrário, essa classe não precisa ser criada agora.

Figura 5.10: Parte do algoritmo para criar novas classes da aplicação no GREN

compilá-lo na nova classe (`methodBody`). Assim, inicialmente a pré-condição é verificada e, caso seja verdadeira, o método pode ser criado e compilado.

1. Para todas as classes da aplicação criadas durante a instanciação faça:
   (a) Seja `c` := classe criada durante a instanciação.
   (b) Para todos os métodos a serem sobrepostos faça:
      i. Seja `m` := método a ser sobreposto.
      ii. Seja `s` := classe na qual `m` está definido.
      iii. Seja `p` := pré-condição para sobreposição de `m`.
      iv. Se `c` herda de `s` e `p` é verdadeira então sobreponha o método `m` da classe `c`

Figura 5.11: Parte do algoritmo para sobrepor métodos-gancho no GREN

A criação do corpo dos métodos-gancho a serem sobrepostos é feita individualmente, usando as facilidades da linguagem Smalltalk para compilar dinamicamente os novos métodos, a partir de

uma variável de texto. Por exemplo, o algoritmo para criar o corpo do método `resourceClass` da Tabela 4.4 é mostrado na Figura 5.12. Esse método deve retornar o nome da classe da aplicação que desempenha o papel de *Recurso* no padrão 9. Por exemplo, em uma oficina de conserto de carros, *Carro* é o recurso sendo consertado e *ConsertoDoCarro* desempenha o papel de *Manutencao do Recurso*, portanto um método `resourceClass` deve ser sobreposto na classe *ConsertoDoCarro*, retornando: ˆCarro. Para obter a classe que desempenha o papel de *Recurso* e *Manutenção do Recurso*, os passos 1, 2 e 3 são executados. Chamando-se o método `classPlayingRole(aString)` da classe *PatternHistory*, obtém-se como resultado a classe que desempenha o papel passado como parâmetro. No passo 4 o nome do método é concatenado ao símbolo de retorno de método do Smalltalk ("ˆ") e ao nome da classe, para compor o corpo do método a ser compilado (passo 5) na classe *ConsertoDoCarro*.

1. Seja `hist` := myApp.historyForPatternNumber(9).
2. Seja `rm` := hist.classPlayingRole("ResourceMaintenance").
3. Seja `r` := hist.classPlayingRole("Resource").
4. Let `m` := "resourceClass"+Enter+"ˆ"+r.
5. Compile o método `m` na classe `rm`.

Figura 5.12: Algoritmo para criar o corpo do método-gancho `resourceClass` no GREN

A maioria dos métodos-gancho do GREN é criada utilizando esse mesmo raciocínio, ora retornando nomes de classes ora variáveis booleanas que denotam se um padrão ou participante foi utilizado durante a instanciação. Outros métodos mais elaborados podem ser necessários para tratar de atributos incluídos nas novas classes, por exemplo, o GREN permite atributos ligados a uma lista vinda de tabela e atributos multi-valorados. A arquitetura do GREN-Wizard mostrou-se adequada para lidar com todos os métodos necessários, não havendo necessidade do engenheiro de aplicações programar manualmente nenhum de seus métodos-gancho.

O GREN-Wizard gera, de forma automática, um *script* para criação das tabelas MySQL referentes à nova aplicação. Isso foi implementado de forma análoga à implementação do corpo dos métodos-gancho, ou seja, monta-se uma variável do tipo texto contendo os comandos SQL para criação de tabelas correspondentes às classes da nova aplicação, de acordo com o histórico de padrões aplicados e com os novos atributos incluídos nas classes.

A Figura 5.13 apresenta, de forma simplificada, o algoritmo para criação desse *script*. Note que o método `colunasParaClasse(umaClasse): aString`, mencionado no passo 4(c)i, retorna

uma variável do tipo texto contendo as colunas específicas da classe passada como parâmetro. O *script* gerado pode ser executado por meio de um botão da GUI do GREN-Wizard, ativado pelo engenheiro de aplicações. Portanto, após executar o GREN-Wizard, a nova aplicação pode ser executada sem programação adicional.

1. Seja `script` := variável texto que conterá o *script* para gerar todas as tabelas MySQL.
2. Seja `dbName` := nome da base de dados relativa à nova aplicação.
3. `script` := 'use ' + `dbName` + ';' + Enter
4. Para todas as classes da aplicação criadas durante a instanciação faça:
   (a) Seja `c` := classe criada durante a instanciação.
   (b) Verifique se os objetos de `c` devem ser persistidos.
   (c) Caso positivo então:
      i. `script` := `script` + colunasParaTabela(`c`).
5. `script` := `script`+ enter + 'quit'.
6. Retorne `script`.

Figura 5.13: Algoritmo para gerar o script de criação de tabelas MySQL no GREN-Wizard

5.5.4 O processo de Construção do GREN-Wizard

O GREN-Wizard foi construído em duas etapas: na primeira, foi feito um protótipo no qual cada padrão da GRN possuía uma classe GUI com um formulário específico, embora o módulo de especificação do domínio já seguisse, desde o início, a arquitetura mostrada na seção 5.5.3. Esse protótipo foi implementado apenas para os dois primeiros padrões da GRN e gerava aplicações pequenas utilizando apenas esses padrões. O módulo de geração de código era bem específico, com um único programa para criar todos os métodos a serem sobrepostos nas novas classes. Esse protótipo foi implementado com os objetivos de melhor identificar a funcionalidade do *Wizard*, já que não havia nenhuma ferramenta similar que pudesse ser tomada como base, e de familiarizar-se com as características de reflexão da linguagem de programação Smalltalk, averiguando a possibilidade de gerar o código das classes em tempo de execução.

Na segunda etapa, o GREN-Wizard foi remodelado para facilitar a geração de código, pois seria muito difícil evoluir a primeira versão para inclusão dos demais padrões da GRN. Assim, foi definida a arquitetura da Figura 5.5, na qual as informações sobre o uso dos padrões na modelagem da aplicação específica pudessem ficar armazenadas, facilitando a sistematização de um algoritmo para gerar o código das classes.

No entanto, a versão atual do GREN-Wizard ainda deixa a desejar em termos de facilidade de adaptação a outros pares "linguagem de padrões e framework". Ele mantém algumas características que o tornam atrelado à GRN e ao GREN, conforme descrito nas seções anteriores e que, portanto, precisam passar por mais um refinamento para obter-se um *Wizard* genérico. Embora a implementação dessa nova versão seja sugerida como trabalho futuro (ver seção 7.5), é fornecido, na seção 5.8, um processo para construção de um *Wizard* genérico, que pode ser adaptado a casos específicos de pares "linguagem de padrões e framework". Esse processo deve ser utilizado para guiar a implementação da próxima versão do GREN-Wizard.

5.6 O Processo de Utilização do *Wizard*

O processo de utilização de um *Wizard* na instanciação de aplicações específicas (Braga e Masiero, 2002e) é ilustrado na Figura 8.14. Os passos 4b.1 e 4b.4 são idênticos aos passos 4a.1 e 4a.4, respectivamente, do processo de instanciação manual mostrado na seção 4.4. Isso se justifica pelo fato do *Wizard* automatizar apenas a etapa de implementação da aplicação específica, ou seja, deve-se primeiro realizar a análise do sistema, depois utilizar o *Wizard* para obter as classes da aplicação, que devem ser complementadas para incluir funcionalidades não cobertas pelo framework e, finalmente, devem ser devidamente testadas. Assim, nesta seção descrevemos apenas os passos 4b.2 e 4b.3.

Com base no modelo de análise da aplicação específica, juntamente com o histórico dos padrões e variantes utilizados, obtidos após a execução do passo 4b.1, no passo 4b.2 o *Wizard* é alimentado com informações necessárias para gerar, automaticamente, o código das classes da aplicação específica. A interação do engenheiro de aplicações com a GUI do *Wizard* deve seguir as recomendações do manual do usuário do *Wizard* e são, portanto, específicas do mesmo. No passo 3b.1.4 da seção 5.8 apresentam-se os requisitos desejáveis para a GUI do *Wizard*, de forma a facilitar essa interação com o engenheiro de aplicações. O resultado desse passo é o código fonte das classes do sistema.

Esse código fonte gerado pelo *Wizard* pode, então, sofrer alterações para atender a requisitos não cobertos pela linguagem de padrões e pelo framework (passo 4b.3). Esse passo é opcional, visto que algumas aplicações podem ser totalmente implementadas automaticamente. Caso seja necessário executar esse passo, a documentação técnica e o código do framework devem ser enten-

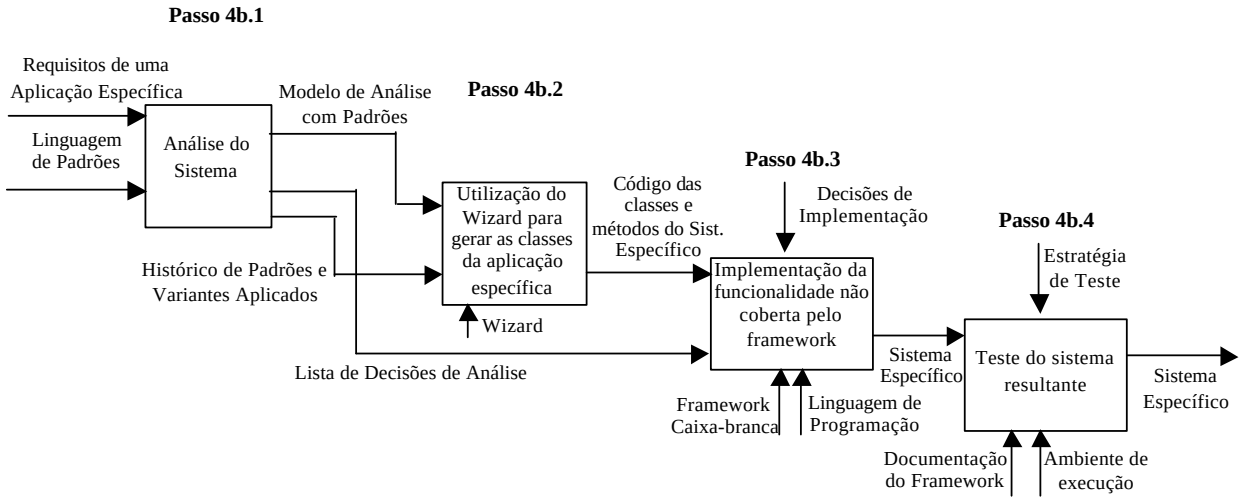


Figura 5.14: Processo para utilização de um *Wizard* na geração de uma aplicação específica

didos para permitir que essas mudanças sejas feitas. Dependendo da linguagem de programação e do framework em particular, pode ser necessário efetuar os procedimentos de compilação do código fonte para obtenção do código executável e criar a base de dados para persistência dos dados. O resultado é o código específico da aplicação, que deve então passar por testes antes de ser entregue ao usuário final.

5.7 Exemplo – Utilização do GREN-Wizard para geração de um Sistema de Controle de Reparo de Buracos

Nesta seção utiliza-se o GREN-Wizard para instanciar o exemplo do Sistema de Acompanhamento e Reparo de Buracos (SARB), descrito na seção 3.5.6 e instanciado na seção 4.5 usando o processo manual.

No passo 4b.1 produz-se o modelo de análise do SARB, que pode ser visto na Figura 3.7, e o histórico de padrões e variantes aplicados, mostrado na Tabela 3.1.

No passo 4b.2 o GREN-Wizard é alimentado com as informações resultantes do passo 4b.1, conforme ilustrado nas Figuras 5.15, 5.16 e 5.17. A interação com o GREN-Wizard é possível consultando seu manual de instanciação[5] ou por meio de treinamento específico.

Após incluir todas as informações sobre o histórico de padrões e variantes usados na modelagem do SARB, o botão “Selecionar Relatórios” é ativado para escolha dos relatórios que farão parte do menu do SARB. A Figura 5.18 mostra os relatórios escolhidos.

[5]BRAGA, R. T. V. GREN-Wizard User Guide. ICMC-USP - São Carlos, documento de trabalho, 15 p., 2002

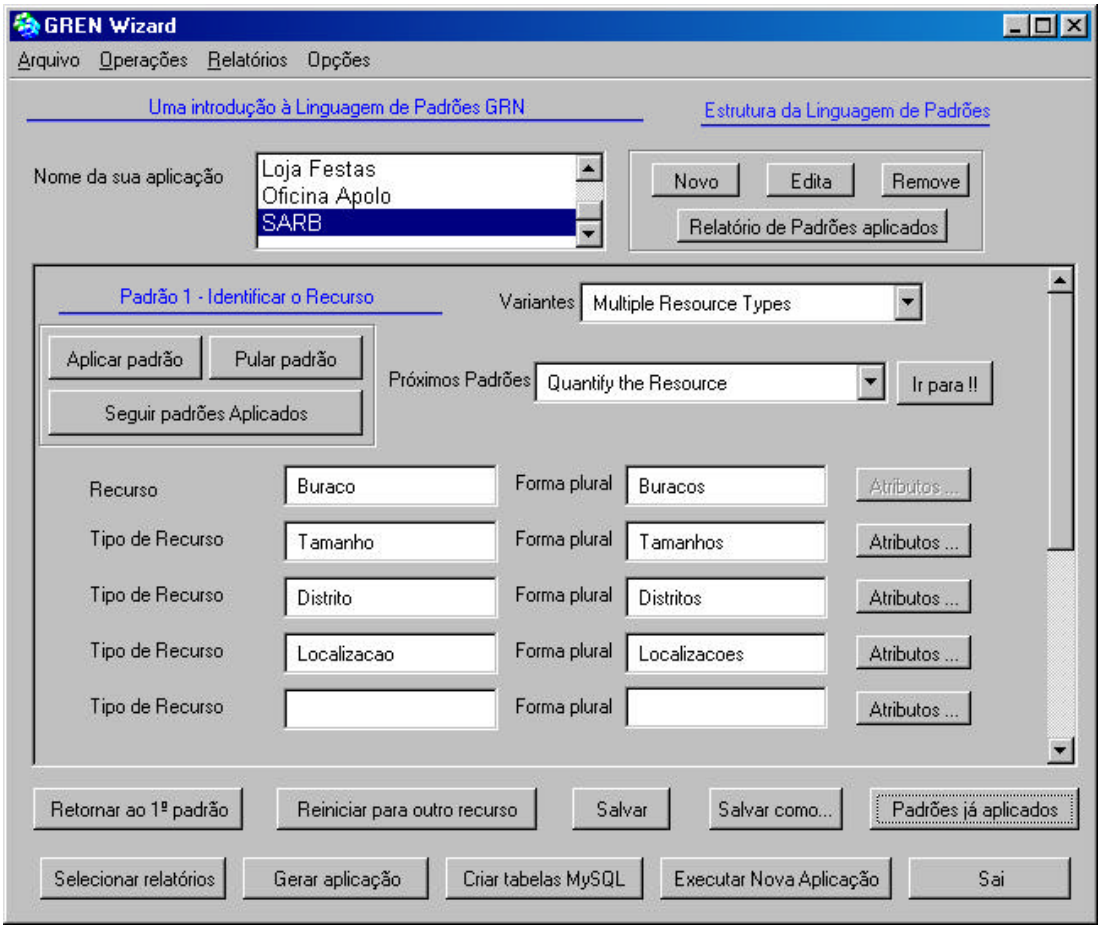


Figura 5.15: GUI do GREN-Wizard após alimentação com informações sobre o SARB (padrão 1)

O sistema resultante, gerado automaticamente após a ativação do módulo de geração de código, possui a GUI idêntica ao sistema instanciado manualmente (Figura 4.11 e Figura 4.12 da seção 4.5). O procedimento para teste do sistema resultante é análogo ao mostrado na seção 4.5.4.

5.8 Desenvolvimento de um *Wizard* Genérico

A construção do GREN-Wizard possibilitou acumular experiências e reunir ensinamentos, tanto a respeito da funcionalidade desse tipo de ferramenta de instanciação, quanto sobre o processo de construção em si. Com base nisso propõe-se, nesta seção, um processo para desenvolvimento de um *Wizard* genérico, que pode ser adaptado conforme os requisitos específicos de um par "framework e linguagem de padrões", permitindo a geração de aplicações usando esse framework e sua linguagem de padrões.

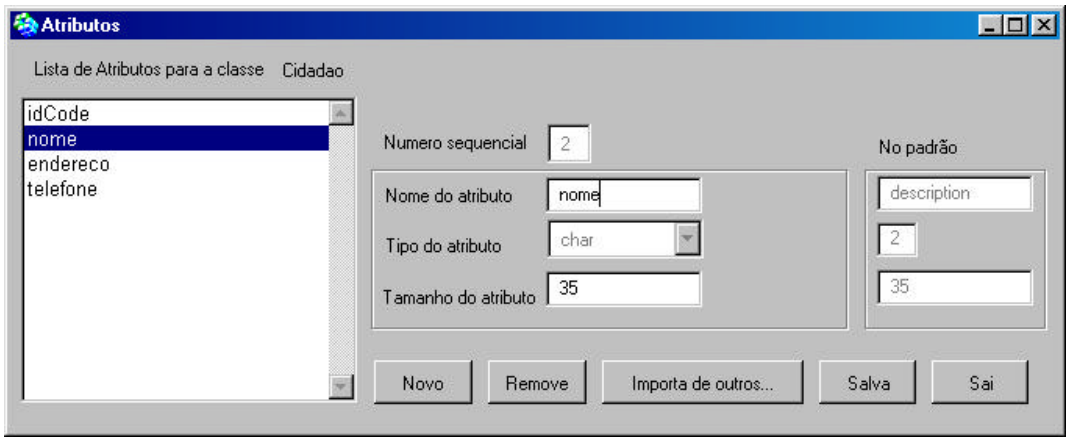


Figura 5.16: GUI do GREN-Wizard após alimentação com informações sobre o SARB (atributos incluídos)

O *Wizard* genérico pode ser considerado sob dois pontos de vista: como um framework que pode ser adaptado para produzir aplicações específicas (nesse caso, *Wizards* específicos) ou como um modelo de objetos adaptativo (do inglês *adaptive object model* (Yoder et al., 2001)), isto é, como um sistema que representa classes, atributos, relacionamentos e comportamento como meta-dados, de forma que o sistema é um modelo baseado em instâncias ao invés de classes. Para mudar o comportamento do sistema, basta que sejam mudados os meta-dados.

Sob o ponto de vista da autora desta tese, é mais correto considerar o *Wizard* genérico como um framework, e documentá-lo de forma a facilitar sua instanciação para aplicações específicas. Apesar dele encaixar-se na definição de modelos de objetos adaptativos, algumas características fazem com que seja difícil considerá-lo como tal, como por exemplo a necessidade de escrever código para geração dos métodos-gancho.

O processo proposto para desenvolvimento de um *Wizard* genérico tem dois passos, 3b.1 e 3b.2, conforme pode ser visto na Figura 8.11: no primeiro constrói-se o *Wizard* genérico e no segundo ele é adaptado para um par específico "framework e linguagem de padrões". O passo 3b.2 pode ser repetido quantas vezes for necessário, produzindo *Wizards* para instanciação de frameworks específicos.

O passo 3b.1 tem como resultado uma ferramenta genérica, que deverá ser adaptada, no passo 3b.2, para cada caso particular (linguagem de padrões e seu framework associado). Traduzindo esses passos para a terminologia de frameworks, temos que o passo 3b.1 trata da construção do framework e o passo 3b.2 faz a instanciação do framework para produzir aplicações específicas. A Figura 8.12 mostra os passos envolvidos no passo 3b.1, detalhados a seguir.

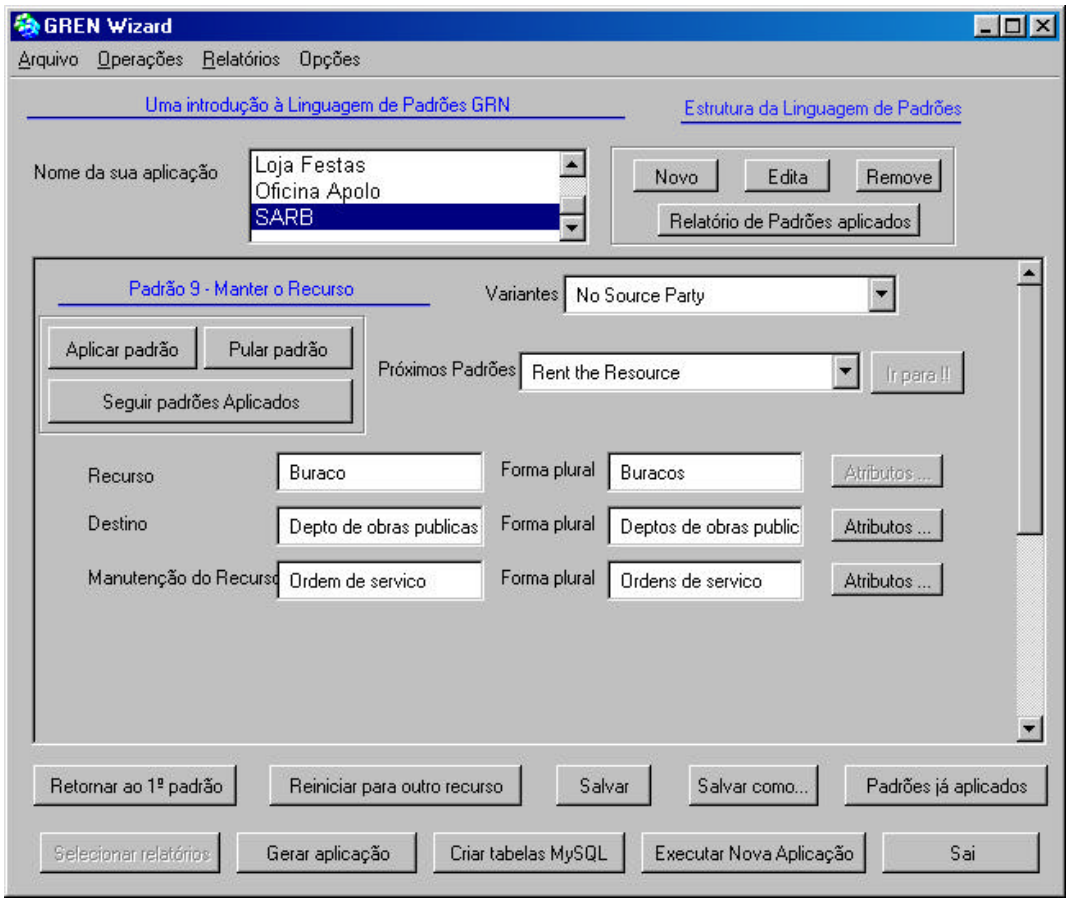


Figura 5.17: GUI do GREN-Wizard após alimentação com informações sobre o SARB (padrão 9)

5.8.1 Construção de um *Wizard* genérico

Passo 3b.1.1. Criar a representação para a linguagem de padrões

O primeiro passo na construção do *Wizard* genérico é criar a representação da linguagem de padrões. Informações sobre os padrões e variantes que a compõem, sobre o relacionamento entre os padrões (inclusive sobre a ordem na qual são aplicados) e as classes pertencentes a cada padrão/variante, bem como seus atributos, métodos e operações, devem ser armazenados em uma base de dados, de forma que possam ser posteriormente recuperados para diversas finalidades. Por exemplo: a) na interface gráfica com o usuário do *Wizard* são mostrados os participantes de cada padrão e o usuário informa os papéis desempenhados pelas classes da aplicação; e b) ao gerar o código da aplicação, os atributos das classes do padrão são confrontados com os atributos da classe concreta, para identificar quais os atributos adicionados e que merecem tratamento diferenciado.

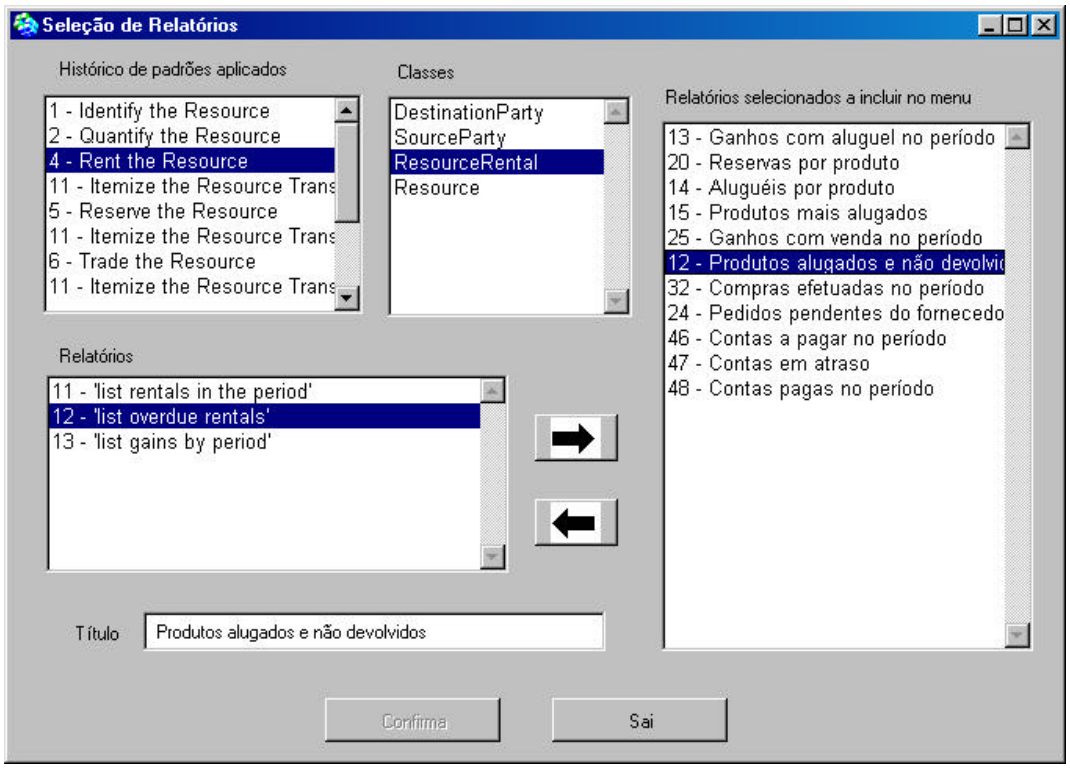


Figura 5.18: GUI do GREN-Wizard após alimentação com informações sobre o SARB (seleção de relatórios)

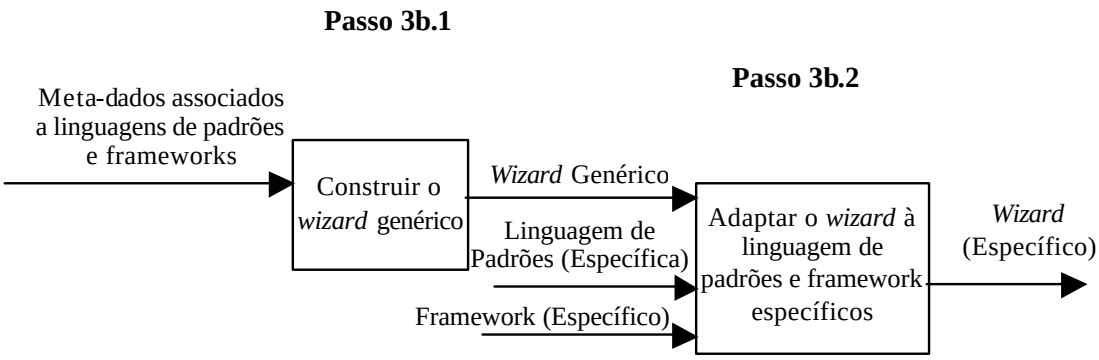


Figura 5.19: Processo para desenvolvimento de *Wizards* para instanciação de frameworks baseados em linguagens de padrões

A Figura 5.21 mostra o modelo de classes associado a essa base de dados, no qual cada classe representa um aspecto importante da linguagem de padrões como, por exemplo, padrões, variantes, classes e atributos. Esse modelo foi obtido generalizando-se o modelo da Figura 5.6, excluindo-se as classes ou atributos que são específicos da linguagem de padrões GRN e do GREN-Wizard.

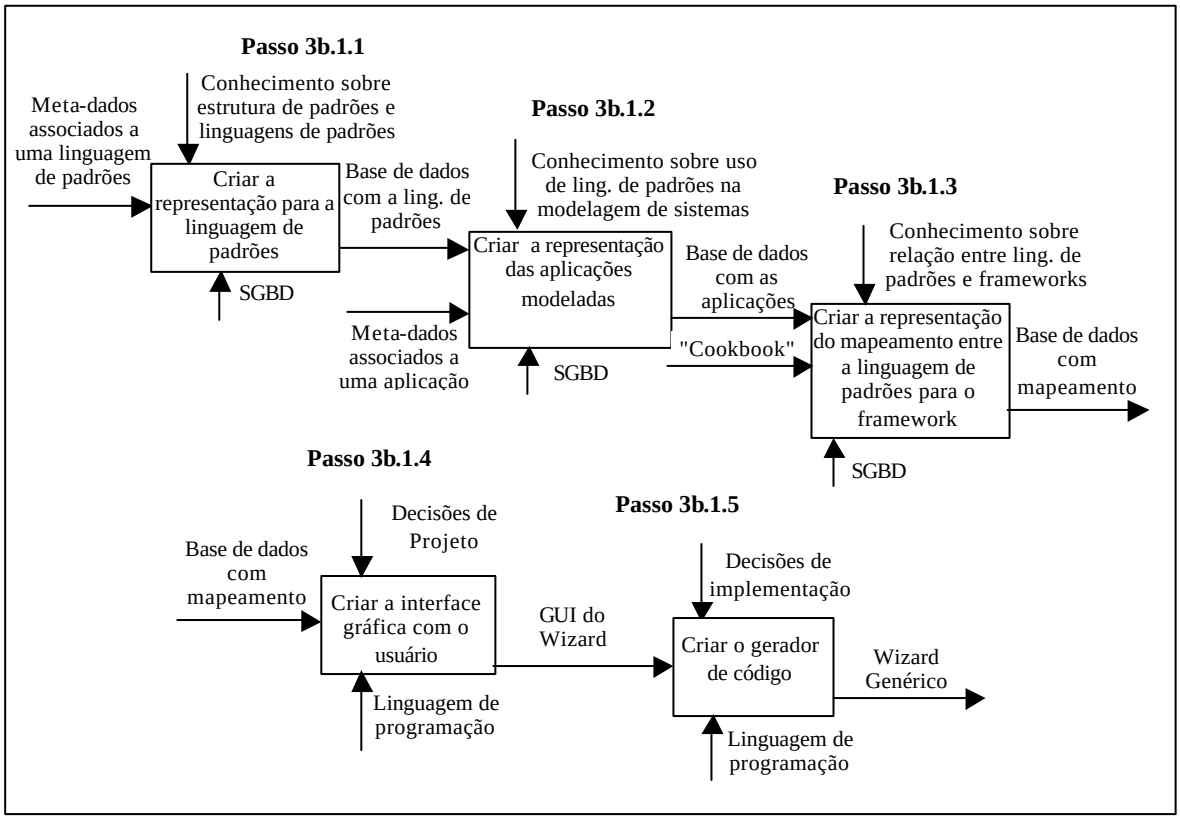


Figura 5.20: Processo para construção de um *Wizard* genérico para instanciação de frameworks baseados em linguagens de padrões

Portanto, a descrição desse modelo, que pode ser encontrada na seção 5.5, não é aqui repetida para evitar redundância de texto.

Para dar suporte a esse mecanismo de representação da linguagem de padrões, pode-se criar um sistema específico para registrar as informações sobre a linguagem de padrões, que ficaria disponível ao desenvolvedor de frameworks. Alternativamente, pode-se entrar com os dados diretamente na base de dados, utilizando um sistema gerenciador de base de dados (SGBD).

Passo 3b.1.2. Criar a representação das aplicações modeladas

O segundo passo para construir o *Wizard* é permitir que as aplicações modeladas com a linguagem de padrões sejam representadas apropriadamente. Toda aplicação gerada a partir do framework deve ser armazenada em uma base de dados, com informações sobre os padrões e variantes aplicados para modelá-la (bem como em que ordem foram aplicados), os papéis desempenhados por cada classe nos padrões e os atributos adicionados às classes dos padrões. Essas informações

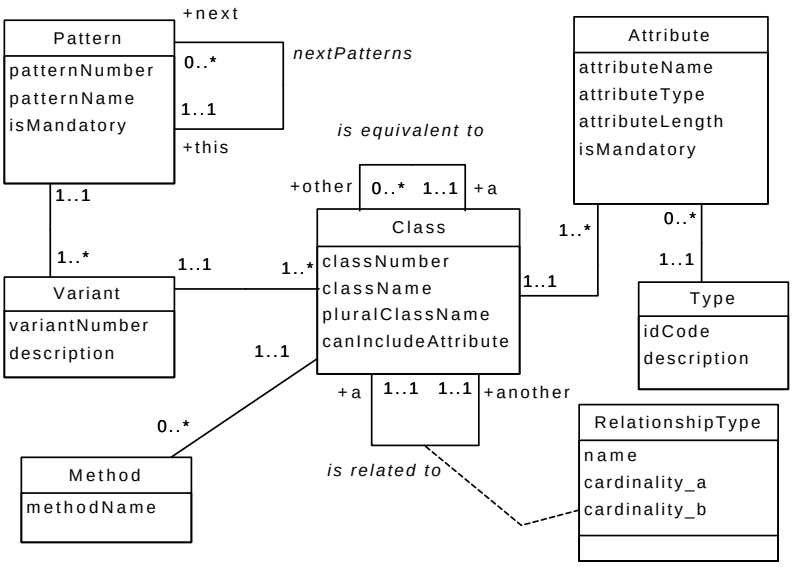


Figura 5.21: Meta-modelo para uma Linguagem de Padrões

serão utilizadas para gerar o código da nova aplicação e também para futuro reuso, já que outras aplicações similares poderão beneficiar-se do histórico de uso armazenado na base de dados.

A Figura 5.22 mostra um meta-modelo para apoiar esse mecanismo, generalizado a partir do modelo do GREN-Wizard (Figura 5.8).

Passo 3b.1.3. Criar a representação do mapeamento entre a linguagem de padrões para o framework

O terceiro passo na construção do *Wizard* é fornecer meios para mapear, possivelmente de forma automática, cada classe participante dos padrões e variantes da linguagem de padrões, para as respectivas classes do framework que a implementam. Isso inclui a especificação dos métodos do framework que devem ser sobrepostos de acordo com o uso dos padrões.

Este passo é o que apresenta maior complexidade, devido à dificuldade de generalizar a dependência existente entre a linguagem de padrões e o framework. Algumas características são válidas para quaisquer pares "linguagem de padrões e framework", mas existem inúmeras particularidades que tornam esse mapeamento difícil. Assim, o *Wizard* deverá ser adaptado caso a caso para tratar as peculiaridades específicas.

Dentre as características comuns encontradas em frameworks desenvolvidos com base em uma linguagem de padrões pode-se citar:

1. *Classes dos padrões possuem uma ou mais classes no framework correspondente*. Por exemplo, na Figura 5.23, a classe *ClasseP3*, que no padrão possuía o comportamento da classe

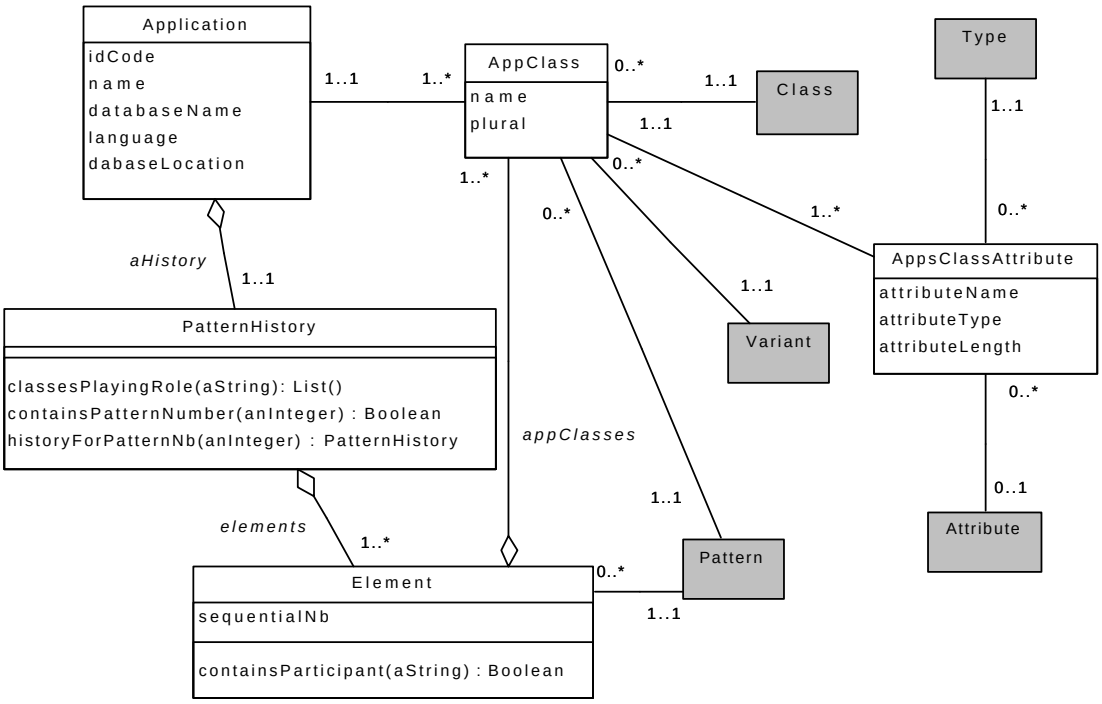


Figura 5.22: Meta-modelo para aplicações geradas usando a Linguagem de Padrões

ClasseP1, no framework possui também o comportamento da nova classe *ClasseF5*. Assim, ao instanciar o framework caixa branca, como o reuso é feito por meio de herança, pode ser necessário criar uma ou mais classes. Esse mapeamento pode ser implementado por intermédio de uma tabela que mostre, para cada variante do padrão, as classes do framework que devem ser especializadas durante a instanciação. Desse modo, o *Wizard* será capaz de automatizar a criação dessas classes.

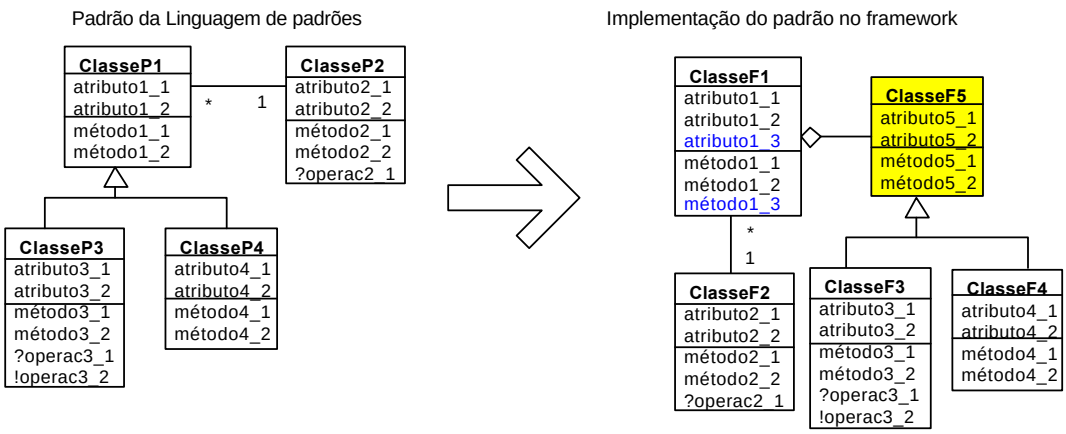


Figura 5.23: Ilustração da correspondência entre uma padrão e sua implementação no framework

2. *Classes abstratas do framework possuem métodos-gancho que devem ser sobrepostos por suas classes descendentes*. Esse mapeamento pode ser implementado por intermédio de uma tabela contendo tais métodos, com indicação dos critérios usados para sobrepô-lo. É possível que o *Wizard* gere o código desses métodos de forma automática, mas isso deve ser tratado caso a caso. No mínimo, mantendo esse mapeamento em uma tabela, pode-se fornecer ao desenvolvedor uma lista com os métodos a serem sobrepostos.

3. *Classes da aplicação podem possuir atributos adicionais, que devem ser devidamente tratados*. Por exemplo, as classes especializadas a partir do framework herdam os atributos das super-classes e, opcionalmente, incluem novos atributos, que devem ter, pelo menos, métodos construtores, de atribuição de valor e de leitura de valor. Já que é possível identificar quais atributos foram incluídos, pois o meta-modelo da Figura 5.22 permite que um atributo de uma classe da aplicação não esteja ligado a um atributo do padrão, então pode-se construir uma tabela contendo os métodos a serem gerados pelo *Wizard* para os novos atributos.

4. *Parte da funcionalidade da linguagem de padrões pode não ter implementação correspondente no framework*. Muitas vezes o framework pode implementar parcialmente algumas das funcionalidades cobertas pela linguagem de padrões. Assim, durante o mapeamento pode-se criar mecanismos para detectar se a aplicação de um padrão/variante possui implementação correspondente no framework. Caso negativo, durante a fase de consistência de informações pode-se emitir uma mensagem de alerta, impedindo que tal uso de padrão seja feito, já que não será possível para o framework apoiar a implementação.

Passo 3b.1.4. Criação da interface gráfica com o usuário

O *Wizard* construído segundo o processo proposto nesta tese deve possuir uma interface gráfica com o usuário (GUI), baseada na linguagem de padrões associada ao framework. Por conseguinte, pode-se enumerar requisitos para essa GUI, de forma que o *Wizard* possua todos os elementos básicos necessários e, ao mesmo tempo, possa ser adaptado para cada caso particular. Os seguintes requisitos fazem parte da funcionalidade comum que espera-se encontrar em um *Wizard* para instanciação de frameworks com base em linguagem de padrões:

1. A GUI do *Wizard* deve permitir ao desenvolvedor de aplicações preencher todas as informações sobre padrões aplicados, bem como possíveis variantes e sub-padrões. Os dados sobre a estrutura da linguagem de padrões devem ser carregados a partir da base de dados criada conforme explicado na seção 5.8.1. Os dados específicos da aplicação instanciada são armazenados na base de dados criada segundo as diretrizes contidas na seção 5.8.1.

2. A GUI deve permitir a aplicação dos padrões de forma iterativa, ou seja, um padrão pode ser aplicado mais do que uma vez, desde que respeite a ordem de aplicação dos padrões imposta pela linguagem de padrões, de acordo com o que for estabelecido pelo relacionamento entre os padrões, denotado no meta-modelo da Figura 5.21 por *nextPatterns*.

3. A GUI deve permitir que o histórico de uso da linguagem de padrões seja armazenado em base de dados, o que pode ser feito por meio de um botão ou opção no menu que, quando acionado, armazene os dados fornecidos pelo usuário na base de dados referente ao meta-modelo da Figura 5.22. A informação armazenada na base de dados pode ser utilizada futuramente para obter informações sobre o uso da linguagem de padrões, permitindo o reuso dessas informações na construção de aplicações similares. Deste modo, é também desejável que se permita criar uma aplicação a partir de outra criada anteriormente, o que pode ser implementado, por exemplo, por meio de uma opção "Salvar como", na qual fornece-se o nome da nova aplicação a ser criada com base em uma já existente.

4. A GUI deve permitir a visualização (por exemplo, por meio de um relatório na tela ou impresso) do histórico de uso da linguagem de padrões na modelagem de uma aplicação específica. Isso permitirá ao desenvolvedor verificar os caminhos seguidos durante o uso da linguagem de padrões, em termos de padrões aplicados e papéis desempenhados pelas classes da aplicação específica.

5. A GUI deve permitir que o desenvolvedor acrescente atributos a uma ou mais classes participantes dos padrões. Em geral, as classes dos padrões possuem apenas os atributos comuns a todas as aplicações do domínio. Portanto, novos atributos podem ser incluídos de acordo com os requisitos da aplicação específica. Deve ser possível determinar quais classes podem ou não receber novos atributos, pois o framework pode possuir limitações quanto a isso. Além disso, como uma mesma classe pode desempenhar papéis diferentes em diversos padrões durante o uso da linguagem de padrões, deve-se limitar os locais nos quais se pode acrescentar os atributos, evitando que sejam acrescentados atributos que, num passo mais à frente, possam pertencer ao próprio padrão. Outra funcionalidade desejável é a de remoção de atributos opcionais dos padrões. Para permitir tais funcionalidades, a base de dados referente ao meta-modelo da Figura 5.21 contém os atributos `canIncludeAttribute` (classe *Class*) e `isMandatory` (classe *Attribute*).

6. A GUI pode implementar um algoritmo para checar se o usuário preencheu adequadamente as telas, de acordo com a obrigatoriedade das classes e relacionamento entre padrões. Caso alguma inconsistência seja encontrada, deve-se emitir um aviso para que o desenvolvedor modifique os padrões já incluídos ou aplique novos padrões.

7. A GUI pode permitir ao desenvolvedor selecionar as operações que farão parte do menu da aplicação. Por exemplo, em sistemas de informação é comum que uma série de relatórios possam ser emitidos ativando-se uma opção do menu. Assim pode-se, durante o uso do *Wizard*, determinar os relatórios desejados dentre os oferecidos pelo framework.

8. Finalmente, a GUI pode oferecer facilidades de ajuda ao usuário, que poderá obter informação sobre os padrões aplicados e sobre a linguagem de padrões como um todo.

Passo 3b.1.5. Criação do gerador de código

A geração do código fonte referente às classes da aplicação instanciada pode ser automatizada pelo *Wizard* construído segundo a abordagem proposta. Para isso, utilizam-se as informações existentes sobre o mapeamento entre a linguagem de padrões e o framework, que permitem saber quais classes devem ser criadas e quais métodos devem ser sobrespostos. Assim como na criação da interface gráfica com o usuário, muitas são as particularidades envolvidas em um framework, o que certamente fará com que a geração de código precise ser adaptada caso por caso. No entanto, as diretrizes mostradas a seguir fornecem os princípios gerais a serem seguidos durante a criação do gerador de código. Deve-se ressaltar que tais princípios aplicam-se apenas a frameworks caixa branca desenvolvidos com base em uma linguagem de padrões, segundo a abordagem proposta nesta tese.

1. Para identificar as classes do framework que devem ser especializadas durante a instanciação, ou seja, quem serão as super-classes correspondentes a cada nova classe da aplicação, o *Wizard* deve possuir um mecanismo que permita percorrer todas as classes da aplicação específica. Para cada uma delas, deve-se buscar, na tabela de mapeamento da linguagem de padrões para o framework (produzida segundo instruções apresentadas na seção 5.8.1), as super-classes do framework a especializar. Isso pode ser feito com base no histórico de padrões aplicados, que está representado, no meta-modelo da Figura 5.21, pela classe *PatternHistory*. Dependendo das facilidades oferecidas pela linguagem de programação na qual o *Wizard* está implementado, essas classes podem ser automaticamente criadas. Caso isso não seja possível pode-se, pelo menos, fornecer ao desenvolvedor uma lista das classes a serem especializadas.

2. Para cada uma das novas classes criadas, identifique quais são os novos atributos acrescentados e que devem, portanto, ser incluídos na declaração das classes na forma de variáveis de instância. Métodos para tratar tais atributos podem ser implementados na fase de adaptação do *Wizard*.

3. Para cada uma das novas classes criadas, identifique quais são os métodos a serem sobrepostos. Para tal, deve-se tomar todas as classes pertencentes à hierarquia da nova classe e verificar quais são seus métodos-gancho. Caso a linguagem de programação permita, esses métodos podem ser automaticamente gerados. Uma forma de implementar essa funcionalidade é por meio de uma tabela contendo as classes abstratas do framework, seus métodos-gancho, a pré-condição para sobreposição do método e o nome de um método do gerador de código que construa o corpo do método e compile-o. Caso não seja possível gerar esses métodos automaticamente, pode-se fornecer ao desenvolvedor uma lista dos métodos a serem sobrepostos.

4. Dependendo do framework em particular, é possível adicionar funções para criar, automaticamente, a base de dados para persistir os objetos. Por exemplo, se o framework utiliza uma base de dados relacional na persistência dos objetos, então um *"script"* pode ser produzido para criar as tabelas referentes às novas classes criadas. O mapeamento entre as classes dos padrões e as tabelas a serem criadas pode ser implementado de forma a facilitar a criação dos *"scripts"*.

5.8.2 Adaptação de um *Wizard* genérico

O passo 3b.2 cuida da adaptação do *Wizard* construído no passo 3b.1, de acordo com cada par específico "framework e linguagem de padrões". As condições mínimas necessárias para fazer tal adaptação são: existência de uma linguagem de padrões composta de padrões de análise para os quais existe um diagrama de classes ilustrando a solução e seus variantes; um framework que implementa os padrões da linguagem de padrões; e a documentação do framework, que mapeia os padrões da linguagem de padrões para as classes do framework.

Diversas atividades, descritas a seguir, são necessárias para adaptar o *Wizard* ao framework específico, conforme ilustrado na Figura 8.13.

Passo 3b.2.1. Registrar informações sobre a linguagem de padrões

Usando a estrutura fornecida pelo *Wizard*, deve-se inserir todas as meta-informações sobre a linguagem de padrões específica, tais como:

- nome de todos os padrões;
- nome de todos os variantes de cada um dos padrões;
- classes participantes de todos os variantes de cada um dos padrões, bem como seus atributos (obrigatórios e opcionais);

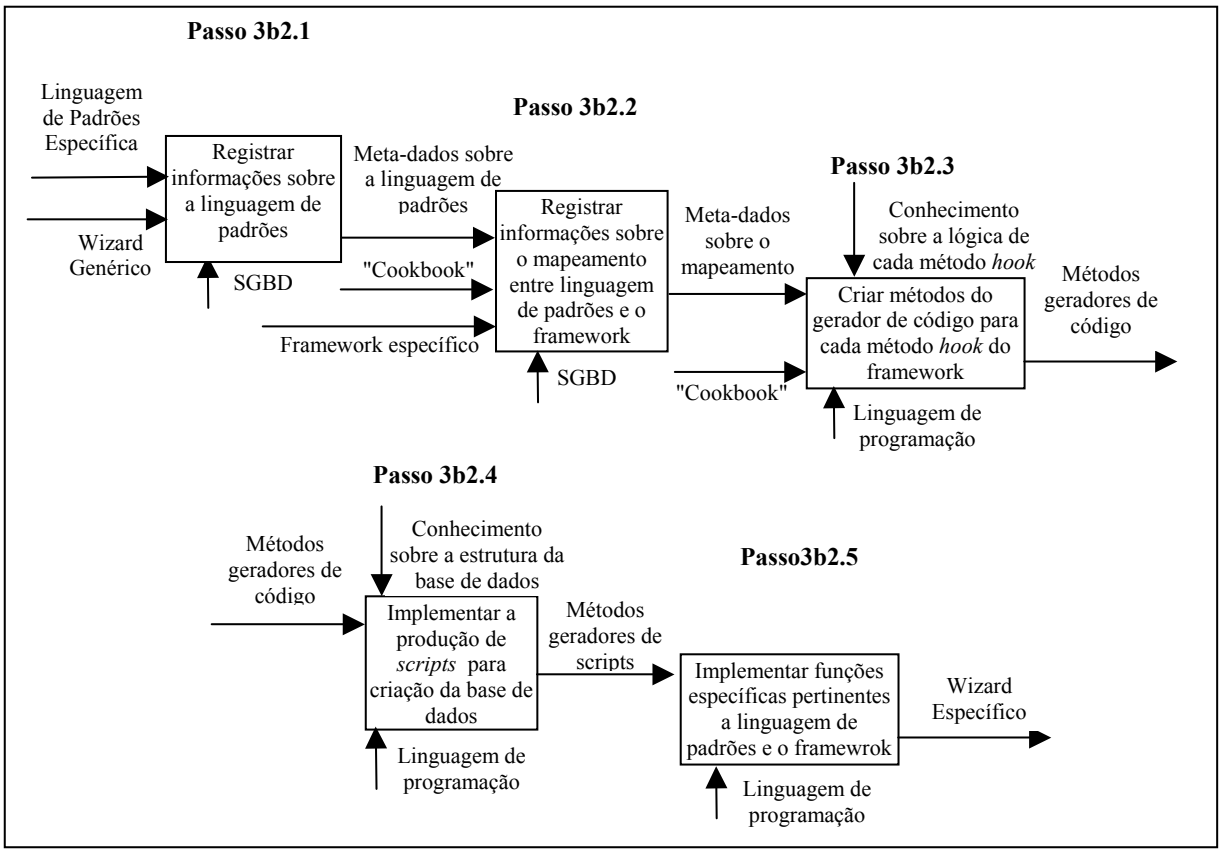


Figura 5.24: Processo para adaptação do *Wizard* para um framework e linguagem de padrões específicos

- relacionamento entre os padrões (ordem na qual podem ser aplicados e restrições de aplicação); e
- operações de entrada e saída que podem ser incluídas nos menus do sistema.

Conforme mencionado na seção 5.8.1, para cadastrar essas informações na base de dados pode-se utilizar o próprio sistema gerenciador da base de dados, ou um sistema de suporte a essa atividade, caso tenha sido implementado durante a construção do *Wizard* genérico. Exemplos desse passo são mostrados na seção 5.5.3.

Passo 3b.2.2. Registrar informações sobre o mapeamento entre a linguagem de padrões e o framework específico

Para cada padrão/variante da linguagem de padrões, deve-se mapear suas classes participantes às classes do framework que as implementam. A principal fonte de informação para realização dessa

atividade é o manual de instanciação do framework, ou *cookbook*, produzido durante a fase de documentação do framework (ver seção 4.2.3). Tendo sido construído usando o processo proposto, esse manual contém tabelas que mapeiam os padrões, em especial as classes participantes de seus variantes, às classes do framework que as implementam. Por conseguinte, pode-se cadastrar esse mapeamento por meio de tabelas de uma base de dados, conforme proposto na seção 5.8.1. Para tal, as tabelas do *cookbook* devem ser armazenadas nas respectivas tabelas do *Wizard*. Se necessário, pode-se recorrer ao código fonte do framework para esclarecer possíveis dúvidas quanto ao mapeamento. Um exemplo completo sobre esse passo pode ser encontrado na seção 5.5.3.

Passo 3b.2.3. Criar os métodos do gerador de código para cada método-gancho do framework

Uma das tabelas do *Wizard* contém os métodos-gancho pertencentes a classes abstratas do framework, e que precisam ser sobrepostos. Para utilizar os mecanismos propostos na seção 5.8.1 para geração automática do código desses métodos, deve-se implementar métodos especiais no *Wizard* que criem o corpo desses métodos e os incluam na classe apropriada da aplicação sendo instanciada.

Deve-se ressaltar que essa atividade é opcional, visto que nem sempre o *Wizard* deve gerar o código mas, dependendo do caso, pode apenas fornecer uma lista dos métodos a serem sobrepostos. Nesse caso, a implementação é de responsabilidade do engenheiro de aplicações.

Para criar um método-gancho, deve-se montar o código desejado para o método, por exemplo por meio de um variável texto (*string*) contendo os comandos que fazem parte do corpo do método, e depois usar as características de reflexão da linguagem de programação para incluir esse método na classe apropriada. Existem pelo menos duas formas de fazer isso: pode-se criar métodos específicos no gerador de código para criar o corpo de cada método-gancho e depois compilar e anexar esse código à classe, ou armazenar em uma tabela na base de dados do *Wizard*, o código que deve ser executado para gerar o corpo do método, e então percorrer essa tabela sistematicamente para gerar todos os métodos-gancho. Exemplos concretos da primeira forma sugerida para esse procedimento são fornecidos na seção 5.5.3.

Passo 3b.2.4. Implementar a geração de *scripts* para criação automática da base de dados

Dependendo do framework, pode ser necessário criar uma base de dados para persistir os objetos, seja ela orientada a objetos ou relacional. É desejável que esta tarefa seja automatizada pelo *Wizard*, por ser uma tarefa trabalhosa e passível de erros. No caso de base de dados relacional, pode-se montar um *script* por meio de uma variável do tipo texto (*string*), que contém os comandos

para criar a base de dados e suas tabelas, com a definição das colunas de cada tabela. Esse *script* pode ser executado posteriormente pelo desenvolvedor de aplicações, ou pode-se fornecer um botão ou opção do menu que o execute.

É necessário observar que o mapeamento da linguagem de padrões para o framework, presente no *cookbook*, pode fornecer diretrizes sobre quais tabelas devem ser criadas e sobre a estrutura de tais tabelas (colunas, índices, chaves, etc.). Assim, a criação dos scripts pode ser feita com base em tabelas que reflitam o mapeamento da linguagem de padrões para as tabelas utilizadas na persistência dos objetos. Um exemplo é fornecido na seção 5.5.3 ilustrando a geração de scripts para criação de tabelas em base de dados MySQL.

Passo 3b.2.5. Implementar funções específicas pertinentes à linguagem de padrões e ao framework

Esta atividade é totalmente dependente de cada caso em particular e, portanto, não pode ser aqui generalizada. Na seção 5.5 pode-se encontrar algumas situações que ocorreram especificamente no GREN-Wizard, exemplificando a necessidade de tratar desses casos individualmente.

5.9 Considerações Finais

Neste Capítulo foram propostos processos para construção e instanciação de um *Wizard* para facilitar a instanciação de frameworks usando linguagens de padrões. Como frameworks construídos usando a abordagem proposta nesta tese possuem uma arquitetura influenciada pela linguagem de padrões, então a criação da ferramenta para automatizar sua instanciação fica facilitada. Ao invés de saber os detalhes de implementação do framework, usuários do *Wizard* precisam apenas saber usar a linguagem de padrões.

O *Wizard* resultante guia o usuário da linguagem de padrões, garantindo que os caminhos certos foram seguidos e fazendo as devidas consistências para assegurar que os padrões aplicados são coerentes entre si. Portanto, o usuário do framework tem condições de saber exatamente onde começar e onde terminar a instanciação. Além do mais, o enfoque da instanciação é na funcionalidade exigida, com uma noção clara de quais requisitos são atendidos por cada padrão.

Wizards desenvolvidos com a abordagem proposta podem ser facilmente reutilizados por outros pares “linguagem de padrões e framework”. A estrutura genérica proposta pode ser considerada tanto como um framework quanto como um modelo de objetos adaptativo. Para adaptá-lo para outros casos, basta preencher a base de dados com informações sobre a linguagem de padrões específica e sobre o mapeamento entre a linguagem de padrões e o framework, e implementar as partes do gerador de código responsáveis pela geração dos métodos-gancho. O restante da funcionalidade independe da linguagem de padrões e do framework.

No próximo capítulo é feita a avaliação do processo geral proposto, por meio de experimentos e relatos de uso dos diversos produtos e processos obtidos durante a pesquisa.

CAPÍTULO

6

Avaliação da Proposta

6.1 Considerações Iniciais

Neste Capítulo, avalia-se a abordagem proposta, por meio de experimentos e relatos de uso dos diversos produtos e processos obtidos ao longo do trabalho. Os experimentos estão documentados usando a estrutura proposta por Wholin et al. (2000): definição do experimento, planejamento do experimento, operação do experimento e análise e interpretação dos resultados. Os diversos usos dos processos e produtos desta tese são relatados de maneira informal, com enfoque maior nos resultados obtidos e lições aprendidas.

Deve-se ressaltar que não foi feita uma avaliação geral de todo o processo, tendo sido a avaliação executada por partes. A realização de uma avaliação geral demandaria muito tempo (a autora estima que por volta de um a dois anos), o que ultrapassaria os limites de tempo estabelecidos para a pesquisa. No entanto, acredita-se que essa avaliação geral poderá ser feita de modo a aperfeiçoar os processos obtidos. Foi realizada a análise estatística para os dois primeiros experimentos executados, que possuem um número suficiente de conjuntos de dados, enquanto para os dois últimos foi feita apenas a análise informal e discussão dos resultados.

O Capítulo está organizado da seguinte forma: na seção 6.2 é avaliado o processo de construção de linguagens de padrões para um domínio específico. Na seção 6.3 avalia-se a utilidade de uma linguagem de padrões na modelagem de sistemas. Na seção 6.4 avaliam-se as vantagens de uma linguagem de padrões na construção de um framework correspondente. Na seção 6.5 avalia-se a

utilidade de uma linguagem de padrões durante o processo de instanciação do framework associado. Na seção 6.6 avaliam-se as vantagens de usar uma linguagem de padrões na construção de um *Wizard* para apoiar a instanciação do framework associado. Na seção 6.7 avalia-se a utilidade de usar uma linguagem de padrões na instanciação de um framework usando um *wizard*. Na seção 6.8 avalia-se o uso de um *wizard* no teste do framework associado. Na seção 6.9 é avaliado o framework GREN na construção de sistemas do domínio, com sugestões de aperfeiçoamento e possíveis extensões. Na seção 6.10 avalia-se o GREN-Wizard em relação ao seu desempenho e limitações. Na seção 6.11 são feitas as considerações finais sobre as avaliações realizadas.

6.2 Avaliação do processo de construção de linguagens de padrões

O processo proposto na seção 3.3 para construção de linguagens de padrões para um domínio específico foi utilizado duas vezes: a primeira pela autora desta tese, para construção da linguagem de padrões GRN e a segunda como parte de um trabalho de mestrado, no qual foi elaborada a linguagem de padrões LV, para controle de sistemas de leilão virtual (Ré et al., 2001).

O processo não foi seguido exatamente da mesma forma nos dois casos, já que ele só pôde ser definitivamente estabelecido após a construção dessas duas linguagens de padrões. A experiência obtida na construção da GRN foi documentada e deu origem à primeira versão do processo. O desenvolvimento da linguagem de padrões LV tomou como base essa primeira versão do processo, que foi refinado buscando melhor definir algumas etapas. Por exemplo, a etapa de criação de um grafo para modelar a interação entre os padrões, que na GRN havia sido feita somente no fim do processo, foi colocada nas etapas iniciais, pois descobriu-se a importância de, ao escrever os padrões, já se ter uma idéia sobre sua ordem de aplicação. O processo apresentado no Capítulo 3 resultou do refinamento do processo obtido após a criação da LV.

A execução do processo na criação de mais uma linguagem de padrões deverá ocorrer em breve, dentro de uma pesquisa de mestrado já iniciada. Assim, espera-se consolidar ainda mais o processo, que poderá tornar-se um método para construção de linguagens de padrões.

Por meio do processo proposto é possível construir linguagens de padrões para outros domínios afins, usando como ponto de partida o conhecimento sobre o domínio e obtendo uma linguagem de padrões que apóia a análise de sistemas nesse domínio. A construção da linguagem é demorada, exigindo diversas iterações até que se obtenha o produto final, mas o esforço é compensado pela facilidade de reuso do conhecimento sobre o domínio por meio dos diversos padrões da linguagem.

6.3 Avaliação da utilidade de linguagens de padrões na modelagem de sistemas

6.3.1 Visão Geral

Para avaliar a utilidade de usar uma linguagem de padrões na modelagem de um sistema no mesmo domínio, foi utilizada a linguagem de padrões GRN, para a qual foram realizados três experimentos, descritos nas sub-seções 6.3.2 a 6.3.4, além de alguns usos informais da mesma na modelagem de sistemas, descritos na sub-seção 6.3.5.

Os três experimentos, detalhados a seguir, consistiram na análise de dois sistemas diferentes, porém com grau de dificuldade similar[1], para os quais foram fornecidos os documentos de requisitos (ver Apêndices A e B). Em um dos sistemas os alunos utilizaram apenas seus conhecimentos sobre orientação a objetos, com o apoio da notação UML (Rational, 2000), e um processo *ad hoc* para modelagem. No outro sistema, os alunos utilizaram a GRN e seu processo de uso para realizar a análise. O objetivo foi comparar o tempo gasto e o número de erros cometidos usando as duas técnicas.

6.3.2 E-GRN-1: Avaliação da GRN na modelagem de sistemas

Definição do Experimento

Objeto de Estudo: Linguagem de padrões GRN

Propósito: Avaliar a utilização da GRN na modelagem de sistemas de gestão de recursos de negócios

Foco qualitativo: Facilidade de modelagem do sistema

Perspectiva: A perspectiva é em relação a desenvolvedores de sistemas no domínio da GRN.

Contexto: o experimento foi realizado por trinta e cinco estudantes de graduação como sujeitos, divididos em grupos, tendo como material básico a linguagem de padrões GRN, um processo de uso da mesma e treinamento prévio de cerca de treze horas/aula. Foi realizado em agosto de 2001 e os alunos pertenciam ao oitavo período do curso “Bacharelado em Ciência de Computação” do ICMC-Universidade de São Paulo.

[1] considerou-se o grau de dificuldade por meio do número de requisitos do sistema e número de classes do sistema final

Planejamento do Experimento

Seleção do Contexto: O experimento foi conduzido de forma independente pelos diversos grupos, não houve comunicação entre eles, mas tiveram liberdade para estipular seus próprios horários para realização das tarefas, desde que anotassem na planilha fornecida os horários exatos de início e fim das atividades. Os estudos foram executados pelos alunos fora de sala de aula e sem supervisão alguma. Nenhum dos estudantes conhecia a linguagem de padrões GRN. O problema utilizado no experimento foi de um sistema real, embora pequeno. Assim, o estudo é válido em um contexto específico do domínio de Engenharia de Software.

Definição da Hipótese

Hipótese nula: São duas as hipóteses: HN1 – o tempo gasto na modelagem de um sistema usando a linguagem de padrões GRN é similar ou maior do que usando um processo *ad hoc* e a notação orientada a objetos UML e HN2 – o número de erros cometidos na modelagem de um sistema usando a linguagem de padrões GRN é similar ou maior do que usando um processo *ad hoc* e a notação orientada a objetos UML.

Hipótese alternativa: São duas as hipóteses: HA1 – o tempo gasto na modelagem de um sistema usando a linguagem de padrões GRN é menor do que usando um processo *ad hoc* e a notação orientada a objetos UML e HA2 – o número de erros cometidos na modelagem de um sistema usando a linguagem de padrões GRN é menor do que usando um processo *ad hoc* e a notação orientada a objetos UML.

Seleção das variáveis:

Variáveis Independentes: *Metodologia utilizada*: Na primeira etapa, um processo *ad hoc* e a notação UML foram utilizados na modelagem, sendo que metade dos grupos modelou um sistema de hotel e a outra metade um sistema de locadora de carros. Na segunda etapa, a linguagem de padrões GRN foi utilizada na modelagem, invertendo-se o sistema alvo; *Experiência e área de interesse do aluno*: a experiência dos alunos com análise orientada a objetos e a área de computação com a qual eles se identificam são fatores que influenciam a facilidade de modelagem.

Variáveis Dependentes: *tempo total* gasto na modelagem do sistema exemplo e *número de erros* encontrados nos modelos, que foram corrigidos com base em um gabarito da solução. Tomou-se nota dos erros cometidos pelos alunos, usando a classificação mostrada na Tabela 6.1. Considera-se que quanto maior o tempo gasto na modelagem e quanto mais erros cometidos, maior será o custo de desenvolvimento.

Tabela 6.1: Tipos de erros considerados no Estudo de Caso com a GRN

Sigla	Descrição do Erro
#ent	Número de entidades (classes) criadas no modelo. O aluno pode ter esquecido alguma entidade ou pode ter adicionado entidades desnecessariamente. Caso o aluno omita uma entidade, deve-se contar também os atributos, métodos, relacionamentos e cardinalidades omitidos por conseqüência.
#atr	Atributos de cada entidade. O aluno pode ter deixado de incluir alguns atributos, pode ter inserido atributos redundantes ou desnecessários ou pode ter inserido atributos na entidade incorreta
#met	Métodos e operações de cada entidade. O aluno pode ter deixado de incluir alguns métodos/operações, pode tê-los inserido de forma redundante ou desnecessária ou pode tê-los inserido na entidade incorreta.
#rel	Relacionamentos entre as entidades, que podem ter sido esquecidos, podem ter sido colocados nos locais errados ou podem ter seu tipo incorreto (por exemplo, agregação ao invés de especialização). Caso o aluno omita um relacionamento, deve-se contar também as cardinalidades omitidas por conseqüência.
#car	Cardinalidade dos relacionamentos. O aluno pode ter omitido ou informado a cardinalidade incorreta de um relacionamento.

Seleção dos sujeitos: A técnica de escolha foi a amostragem por conveniência (as pessoas mais próximas e mais convenientes são selecionadas como sujeitos), visto que são alunos matriculados em disciplina de graduação. A divisão dos trinta e cinco alunos em doze grupos não foi aleatória, pois os próprios alunos montaram os grupos como desejaram. A distribuição dos sistemas de hotel e locadora de carros foi aleatória na primeira etapa e invertida na segunda etapa. Os alunos não tiveram chance de decidir se iriam ou não participar, ou seja, o experimento fazia parte de um projeto obrigatório da disciplina. Assim, não se pode garantir que os resultados seriam os mesmos se os participantes fossem outros, por exemplo, profissionais escalados de maneira *ad hoc* ou voluntários.

Projeto do Experimento: A Tabela 6.2 mostra a divisão dos grupos e sistemas nas duas etapas do experimento.

Tabela 6.2: Projeto do E-GRN-1

Grupo	Etapa 1: Modelagem com a UML	Etapa 2: Modelagem com a GRN
1	Hotel	Locadora de Carros
2	Locadora de Carros	Hotel
3	Hotel	Locadora de Carros
4	Locadora de Carros	Hotel
5	Hotel	Locadora de Carros
6	Locadora de Carros	Hotel
7	Hotel	Locadora de Carros
8	Locadora de Carros	Hotel
9	Hotel	Locadora de Carros
10	Locadora de Carros	Hotel
11	Hotel	Locadora de Carros
12	Locadora de Carros	Hotel

Instrumentação: O material fornecido aos sujeitos para realização do experimento foi composto de: Diretrizes para realização do experimento; Documento de Requisitos dos dois sistemas; linguagem de padrões GRN; processo de uso da linguagem de padrões GRN; Formulários para coleta de dados do experimento, que podem ser encontrados no Apêndice D (F1: formulário a ser preenchido pelo grupo, contendo o tempo total gasto, em pessoas hora, e um questionário a respeito das dificuldades encontradas para execução do experimento e F2:

formulário a ser preenchido individualmente, respondendo um questionário para categorizar o aluno de acordo com alguns pontos-chave para melhor entendimento e análise dos resultados, tais como a experiência profissional, disciplinas cursadas, etc.).

Avaliação da Validade: *Validade da conclusão*: o experimento foi elaborado de forma que os resultados comprovem ou não a hipótese, sem influência de outros fatores externos. *Validade interna*: o experimento tem condições de ser repetido com os mesmos participantes e objetos. Um fator que facilita essa repetição é a ausência de comunicação e competição entre os participantes. *Validade externa*: é provável que o resultado não possa ser generalizado fora do escopo deste estudo, já que a escolha foi não aleatória e não reflete o mercado de trabalho. *Validade da construção*: um questionário foi aplicado aos participantes, no qual foi possível saber a experiência anterior dos mesmos no domínio do objeto de estudo. Assim, foi possível medir a influência desses fatores nos resultados apurados. O fato dos alunos estarem realizando um projeto da disciplina, inclusive valendo nota, poderia ser agravante, já que eles poderiam manipular as informações sobre os tempos gastos para melhorar a nota. No entanto, este fato foi desconsiderado, pois os alunos foram avisados sobre a importância da precisão dessa informação para o experimento, tendo sido garantido a eles que o tempo gasto não traria influência na nota final.

Operação do Experimento

Preparação: após estabelecido contato com os participantes, foi preparada a instrumentação necessária: a linguagem de padrões GRN foi disponibilizada na Web, o treinamento foi elaborado por meio de transparências sobre o conteúdo e os questionários foram impressos. Foi garantido anonimato aos participantes.

Participantes: Participaram do experimento trinta e cinco alunos de graduação do curso "Bacharelado em Ciências de Computação" do ICMC-USP. As Tabelas 6.3 e 6.4 mostram o perfil desses estudantes em relação à área de interesse e experiência no domínio tratado, respectivamente.

Tabela 6.3: Área de interesse dos alunos do E-GRN-1

Área de Interesse	Número de alunos	Percentual
Redes/Sist Distribuídos	4	11%
Banco de Dados	6	17%
Inteligência artificial	6	17%
Engenharia Soft/Sist. Inform.	3	9%
Computação gráfica	6	17%
Hipermídia	9	26%
Hardware	1	3%
Total de alunos que respondeu	**35**	**100%**

Tabela 6.4: Experiência dos alunos do E-GRN-1

Experiência no domínio tratado	Número de alunos	Percentual
Sem experiência	5	14%
Realizou projetos durante disciplinas de graduação/pós graduação usando análise estruturada	9	26%
Realizou projetos durante disciplinas de graduação/pós graduação usando análise orientada a objetos	20	57%
Desenvolveu, profissionalmente, até 3 projetos nesse domínio	1	3%
Total de alunos que respondeu	**35**	**100%**

Execução: A execução do experimento deu-se em duas etapas: na primeira etapa foi realizado o treinamento de uma hora a respeito da UML, pois os participantes já possuíam conhecimento sobre orientação a objetos. Foi então distribuída a tarefa aos participantes, de modelar um sistema usando a UML. O grupo deveria entregar um diagrama de classes para tal sistema, contendo as classes (com atributos, métodos e operações) e os relacionamentos entre as classes (juntamente com as respectivas cardinalidades). Foram entregues aos participantes os formulários F1 e F2. Após duas semanas, os alunos entregaram a modelagem referente à primeira etapa. Foi então realizado o treinamento de doze horas sobre a linguagem de padrões GRN, após o qual os participantes receberam a tarefa de modelar outro sistema usando a GRN como auxiliar, e segundo os requisitos presentes no documento fornecido. O grupo deveria entregar um diagrama de classes do sistema, também usando a notação UML, contendo as classes (com atributos, métodos e operações) e os relacionamentos entre as classes (juntamente com as respectivas cardinalidades). No diagrama de classes deveria ser anotado, para cada classe participante, o papel por ela desempenhado no padrão aplicado. Cada grupo recebeu o formulário F1 para anotar os tempos e dificuldades referentes à segunda etapa. Os alunos também entregaram os modelos resultantes da segunda etapa após duas semanas.

Validação dos Dados: Os formulários distribuídos aos participantes foram conferidos para verificar se foram preenchidos corretamente. Algumas perguntas foram feitas aos participantes para assegurar que seguiram as recomendações sugeridas. Todos eles participaram do experimento de maneira responsável e portanto nenhum dos participantes foi descartado.

Análise e Interpretação dos Resultados

A Tabela 6.5 mostra os resultados obtidos na primeira etapa do E-GRN-1, que consistiu na análise do sistema usando um processo *ad hoc* e a notação UML. A Tabela 6.6 mostra os resultados obtidos na segunda etapa do E-GRN-1, que consistiu na análise do sistema usando a GRN. São mostrados os tempos gastos, erros cometidos (segundo os critérios mostrados na Tabela 6.1) e notas finais obtidas pelos diversos grupos. A nota final foi obtida sem considerar o tempo gasto, mas apenas o número de erros cometidos.

Tabela 6.5: Resultados do E-GRN-1 – Abordagem *Ad hoc*

Grupo	Pessoas/hora	#ent	#atr	#met	#rel	#car	Total de erros	Nota
Sistema Hotel								
1	3,00	1,00	26,00	15,00	10,00	19,00	71	3,75
3	15,00	1,00	20,00	7,00	9,00	16,00	53	4,25
5	3,50	0,00	14,00	2,00	0,00	0,00	16	8,75
7	13,00	1,00	20,00	6,00	8,00	13,00	48	5,75
9	3,80	1,00	17,00	19,00	5,00	26,00	68	4,25
11	7,50	0,00	15,00	15,00	4,00	10,00	44	6,50
Média	**7,63**	**0,67**	**18,67**	**10,67**	**6,00**	**14,00**	**50**	**5,54**
Sistema Locadora de Carros								
2	6,00	1,00	14,00	3,00	4,00	8,00	30	7,50
4	3,00	1,00	7,00	2,00	2,00	8,00	20	8,25
6	4,00	3,00	29,00	17,00	12,00	25,00	86	2,00
8	4,50	1,00	21,00	6,00	6,00	14,00	48	5,75
10	5,50	1,00	14,00	4,00	6,00	10,00	35	6,75
12	4,00	13,00	24,00	19,00	9,00	20,00	85	1,00
Média	**4,50**	**3,33**	**18,17**	**8,50**	**6,50**	**14,17**	**50,67**	**5,21**
Média Geral	**6,07**	**2,00**	**18,42**	**9,58**	**6,25**	**14,08**	**50,33**	**5,38**

Tabela 6.6: Resultados do E-GRN-1 – Abordagem GRN

Grupo	Pessoas/hora	#ent	#atr	#met	#rel	#car	Total de erros	Nota
Sistema Locadora de Carros								
1	6,00	3,00	14,00	8,00	4,00	26,00	55	5,00
3	12,00	0,00	11,00	10,00	1,00	3,00	25	7,50
5	3,80	1,00	8,00	2,00	2,00	3,00	16	8,50
7	12,00	2,00	19,00	4,00	4,00	26,00	55	5,50
9	10,00	1,00	19,00	5,00	4,00	8,00	37	6,75
11	3,00	3,00	18,00	11,00	3,00	8,00	43	5,00
Média	**7,80**	**1,67**	**14,83**	**6,67**	**3,00**	**12,33**	**38,5**	**6,38**
Sistema Hotel								
2	7,50	1,00	17,00	5,00	2,00	8,00	33	7,00
4	12,00	2,00	6,00	2,00	0,00	1,00	11	8,50
6	9,00	2,00	12,00	0,00	4,00	6,00	24	7,50
8	12,00	1,00	10,00	1,00	0,00	1,00	13	8,75
10	6,70	0,00	10,00	2,00	4,00	6,00	22	8,25
12	5,50	1,00	8,00	11,00	4,00	2,00	26	7,00
Média	**8,78**	**1,17**	**10,50**	**3,50**	**2,33**	**4,00**	**21,5**	**7,83**
Média Geral	**8,29**	**1,42**	**12,67**	**5,08**	**2,67**	**8,17**	**30**	**7,10**

Comparando-se as médias apresentadas na Tabela 6.5, observa-se que a nota média dos grupos ímpares é aproximadamente igual à dos grupos pares, o que ajuda a confirmar (embora não prove) a similaridade no grau de dificuldade dos dois sistemas, já que o perfil dos estudantes é bastante semelhante e os grupos foram divididos de forma homogênea. Comparando-se as médias gerais das Tabelas 6.5 e 6.6, percebe-se um aumento tanto na nota final dos grupos quanto no número de horas gastas na modelagem. Por outro lado, nota-se uma diminuição do número global de erros cometidos.

As Figuras 6.1 e 6.2 ilustram graficamente algumas das informações das Tabelas 6.5 e 6.6. Em particular, foram considerados nessas figuras o tempo (em pessoas/hora) gasto para modelagem do sistema e o número de erros cometidos, que constituem as hipóteses HA1 e HA2.

A análise estatística dos resultados foi feita utilizando dois tipos de teste: o teste não paramétrico da soma dos postos de Wilcoxon para comparação de duas amostras independentes e o

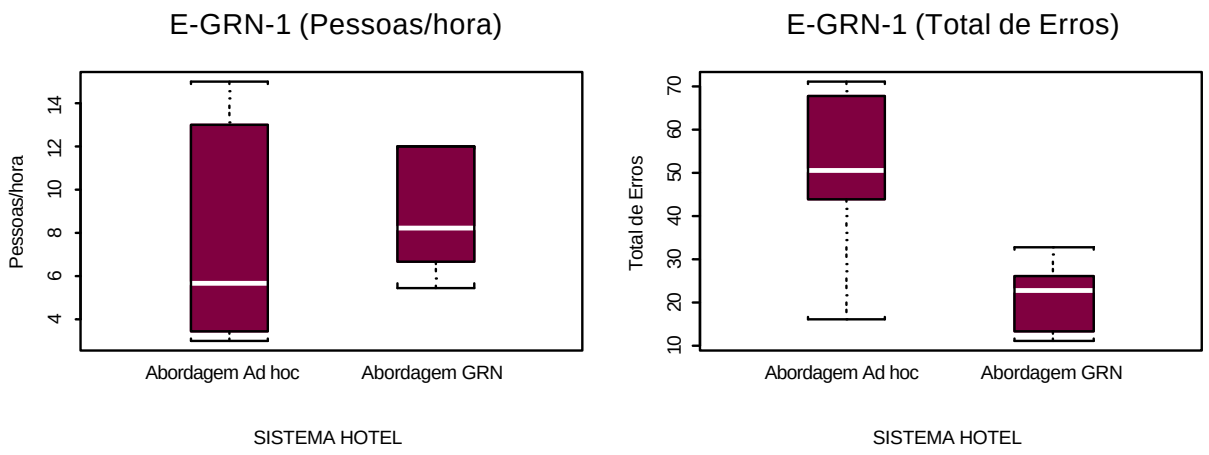


Figura 6.1: Representação gráfica dos resultados do E-GRN-1 nas duas abordagens - Sistema Hotel

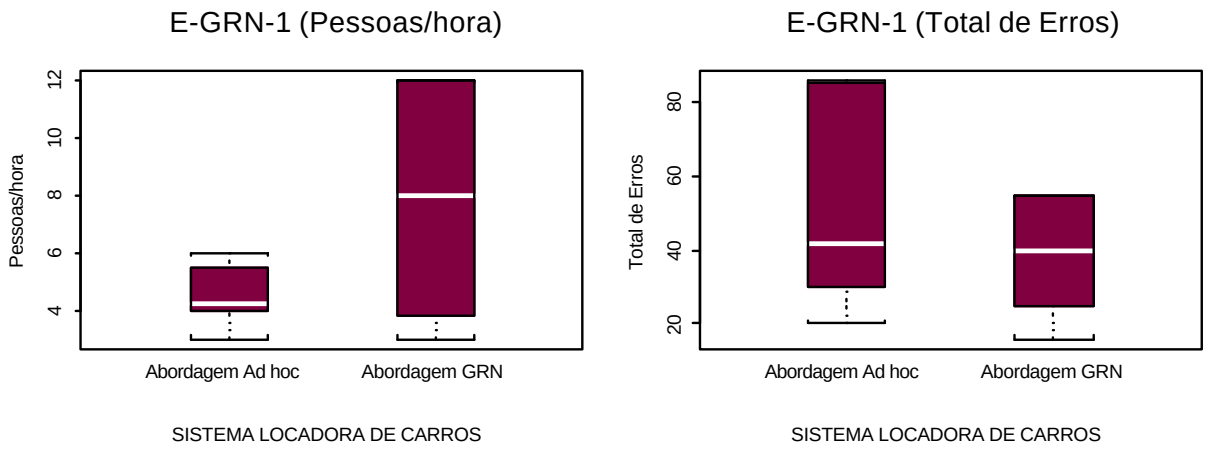


Figura 6.2: Representação gráfica dos resultados do E-GRN-1 nas duas abordagens - Sistema Locadora de Carros

teste não paramétrico de Mann Whitney para comparações de duas amostras independentes (Conover, 1980; Hollander e Wolfe, 1973). Da forma como o experimento foi planejado, dois conjuntos independentes de amostras foram obtidos: Conjunto 1 (Sistema Hotel) = Abordagem *Ad hoc* realizada pelos grupos ímpares X Abordagem GRN realizada pelos grupos pares e Conjunto 2 (Sistema Locadora de Carros) = Abordagem *Ad hoc* realizada pelos grupos pares X Abordagem GRN realizada pelos grupos ímpares.

Considerou-se estatisticamente significativas as comparações cujo valor de p (p-valor) fosse menor do que 0,05. A Tabela 6.7 mostra os resultados obtidos, correspondentes às quatro compara-

ções mostradas nas Figuras 6.1 e 6.2. Os testes foram inconclusivos para o número de pessoas/hora em ambos os sistemas (HA1) e para o número de erros cometidos no sistema de locadora de carros (HA2). Para o número de erros cometidos no sistema de hotel, os testes foram conclusivos, isto é, pode-se dizer que os alunos que usaram a abordagem GRN na modelagem do sistema de hotel cometeram menos erros do que os alunos que utilizaram a abordagem *ad hoc* na análise desse mesmo sistema (hipótese HA2).

Tabela 6.7: Resultados Estatísticos para E-GRN-1

Conjunto de dados	Critério	P-valor – Wilcoxon	P-valor – Mann Whitney	Resultado
Conjunto 1 (Sistema Hotel)	Pessoas-hora	0,4821	0,5887	inconclusivo
	Total de erros	0,0306	0,0261	conclusivo
Conjunto 2 (SistemaLocadora de Carros)	Pessoas-hora	0,2946	0,3095	inconclusivo
	Total de erros	0,6884	0,6991	inconclusivo

Discussão

Além da análise quantitativa dos resultados mostrada acima, uma análise qualitativa pode ser realizada com base no depoimento de dois grupos de alunos que realizaram o experimento. Segundo um dos grupos, a análise do sistema foi mais fácil e completa usando a GRN, embora tenha sido empregado mais tempo devido à inexperiência com o uso da GRN. Outro grupo disse que a análise com a GRN fez com que o número de detalhes a serem cuidados crescesse significativamente, aumentando o número de horas gastas na modelagem. Ao mesmo tempo, isso deu ao grupo o sentimento de que o modelo produzido na análise estaria falho, caso tivesse utilizado apenas a abordagem tradicional com UML, ou seja, provavelmente eles teriam se esquecido de muitos detalhes importantes na análise.

Esses dois depoimentos reforçam o resultado obtido, justificando o aumento do tempo gasto na modelagem e da nota final. O aumento do tempo gasto não implica necessariamente em queda de produtividade, já que o aumento na nota reflete a melhoria de qualidade do modelo obtido, pela diminuição do número de erros cometidos. Assim, embora em um tempo maior, obtém-se um modelo mais próximo do modelo final a ser projetado e implementado. Ainda segundo o depoimento de alguns alunos, esse tempo pode ser diminuído à medida que eles ganhem experiência no uso da GRN.

6.3.3 E-GRN-2: Avaliação da GRN na modelagem de sistemas

Definição do Experimento

Objeto de Estudo: Linguagem de padrões GRN.

Propósito: Avaliar a utilização da GRN na modelagem de sistemas de gestão de recursos de negócios.

Foco qualitativo: facilidade de modelagem do sistema.

Perspectiva: A perspectiva é em relação a desenvolvedores de sistemas no domínio da GRN.

Contexto: o experimento foi realizado por nove estudantes de pós-graduação como sujeitos, tendo como material básico a linguagem de padrões GRN, um processo de uso da mesma e treinamento prévio de cerca de seis horas/aula. Foi realizado em outubro de 2001 e os alunos cursavam a disciplina "Tópicos em Engenharia de Software", da área específica de Engenharia de Software da Universidade Federal de São Carlos.

Planejamento do Experimento

Seleção do Contexto: idem ao contexto do E-GRN-1.

Definição da Hipótese: idem ao E-GRN-1.

Seleção das variáveis:

Variáveis Independentes: Na primeira etapa, um processo *ad hoc* e a UML foram utilizados na modelagem do sistema de hotel; na segunda etapa, a linguagem de padrões GRN foi utilizada na modelagem do sistema locadora de carros.

Variáveis Dependentes: idem ao E-GRN-1.

Seleção dos sujeitos: A técnica de escolha foi a amostragem por conveniência (as pessoas mais próximas e mais convenientes são selecionadas como sujeitos), visto que são alunos matriculados em disciplina de pós-graduação. Escolheu-se, aleatoriamente, o sistema de hotel para ser modelado com o processo *ad hoc*/UML e o sistema de locadora de carros para ser modelado com a GRN. Os alunos realizaram o projeto individualmente, não havendo grupos de alunos. Eles não puderam decidir se iriam ou não participar, pois o experimento fazia parte de um projeto obrigatório da disciplina. Assim, não se pode assegurar que os resultados seriam os mesmos se os participantes fossem outros, por exemplo, profissionais escalados de maneira *ad hoc* ou voluntários.

Projeto do Experimento: A Tabela 6.8 mostra a divisão dos grupos e sistemas nas duas etapas do experimento.

Instrumentação: idem ao E-GRN-1.

Avaliação da Validade: idem ao E-GRN-1.

Tabela 6.8: Projeto do E-GRN-2

Aluno	Etapa 1: Modelagem com a UML	Etapa 2: Modelagem com a GRN
1	Hotel	Locadora de Carros
2	Hotel	Locadora de Carros
3	Hotel	Locadora de Carros
4	Hotel	Locadora de Carros
5	Hotel	Locadora de Carros
6	Hotel	Locadora de Carros
7	Hotel	Locadora de Carros
8	Hotel	Locadora de Carros
9	Hotel	Locadora de Carros

Operação do Experimento

Preparação: idem ao E-GRN-1.

Participantes: Participaram do experimento nove alunos de pós-graduação, cujo perfil é resumido nas Tabelas 6.9 e 6.10.

Tabela 6.9: Área de interesse dos alunos do E-GRN-2

Área de Interesse	Número de alunos	Percentual
Redes/Sist Distribuídos	0	0%
Banco de Dados	1	11%
Banco de Dados/Engenharia de Software	2	22%
Inteligência artificial	0	0%
Engenharia Soft/Sist. Inform.	6	67%
Computação gráfica	0	0%
Hipermídia	0	0%
Hardware	0	0%
Total de alunos que respondeu	**9**	**100%**

Tabela 6.10: Experiência dos alunos do E-GRN-2

Experiência no domínio tratado	Número de alunos	Percentual
Sem experiência	0	0%
Realizou projetos durante disciplinas de graduação/pós graduação usando análise estruturada	0	0%
Realizou projetos durante disciplinas de graduação/pós graduação usando análise orientada a objetos	4	44%
Realizou projetos durante disciplinas de graduação/pós graduação usando análise estruturada e análise orientada a objetos	4	4%
Desenvolveu, profissionalmente, até 3 projetos nesse domínio	0	0%
Desenvolveu, profissionalmente, mais de 4 projetos nesse domínio	1	11%
Total de alunos que respondeu	**9**	**100%**

Execução: similar ao E-GRN-1, exceto quanto à organização do alunos, que trabalharam individualmente neste experimento.

Validação dos Dados: idem ao E-GRN-1.

Análise e Interpretação dos Resultados

A Tabela 6.11 mostra os resultados obtidos na primeira etapa do E-GRN-2 (análise do sistema usando um processo *ad hoc* e a notação UML) enquanto a Tabela 6.12 mostra os resultados obti-

dos na segunda etapa do E-GRN-2 (análise do sistema usando a GRN). A observação direta dos resultados indica que o tempo médio de modelagem foi ligeiramente superior na abordagem GRN, em contrapartida ao número de erros, que foi menor na abordagem GRN.

Tabela 6.11: Resultados do E-GRN-2 – Abordagem *Ad hoc*

Aluno	Pessoas/hora	#ent	#atr	#met	#rel	#car	Total de erros	Nota
Sistema Hotel								
1	6,40	3,00	12,00	2,00	6,00	11,00	34	5,75
2	3,00	0,00	14,00	5,00	3,00	4,00	26	7,75
3	2,50	0,00	24,00	7,00	11,00	10,00	52	5,50
4	1,00	2,00	23,00	7,00	7,00	13,00	52	5,00
5	3,00	1,00	5,00	4,00	2,00	3,00	15	8,50
6	3,00	3,00	30,00	7,00	8,00	14,00	62	4,50
7	3,00	0,00	12,00	0,00	2,00	4,00	18	8,50
8	3,50	0,00	3,00	0,00	1,00	6,00	10	9,00
9	2,20	2,00	16,00	4,00	5,00	6,00	33	6,50
Média	**3,07**	**1,22**	**15,44**	**4,00**	**5,00**	**7,89**	**33,56**	**6,78**

Tabela 6.12: Resultados do E-GRN-2 – Abordagem GRN

Aluno	Pessoas/hora	#ent	#atr	#met	#rel	#car	Total de erros	Nota
Sistema Locadora de Carros								
1	6,00	0,00	18,00	0,00	2,00	4,00	24	8,00
2	3,70	2,00	13,00	2,00	4,00	5,00	26	7,25
3	5,00	3,00	10,00	3,00	6,00	4,00	26	7,00
4	2,00	1,00	12,00	5,00	4,00	4,00	26	7,50
5	3,50	1,00	7,00	2,00	2,00	2,00	14	8,50
6	3,00	3,00	11,00	4,00	2,00	4,00	24	7,00
7	2,00	2,00	8,00	7,00	5,00	4,00	26	7,25
8	3,30	2,00	21,00	8,00	3,00	6,00	40	6,00
9	3,70	1,00	11,00	10,00	2,00	4,00	28	7,25
Média	**3,58**	**1,67**	**12,33**	**4,56**	**3,33**	**4,11**	**26**	**7,31**

A Figura 6.3 ilustra o tempo gasto (em pessoas/hora) e o número de erros cometidos nas duas abordagens para o E-GRN-2 (abordagem *ad hoc* = Sistema de Hotel e abordagem GRN = Sistema de Locadora de Videos).

A análise estatística dos resultados foi feita utilizando o teste não paramétrico de Wilcoxon (pareado) para comparação de duas amostras co-relacionadas (Conover, 1980; Hollander e Wolfe, 1973), já que o planejamento desse experimento permite apenas a comparação entre o sistema de hotel *versus* o sistema de locadora de carros, para os mesmos alunos. Os resultados foram inconclusivos, tanto para o número de pessoas-hora (p-valor = 0,2349) quanto para o número de erros cometidos (p-valor = 0,2591).

Discussão

Alguns depoimentos de alunos que participaram do experimento podem ajudar na análise qualitativa dos resultados. Um dos alunos alegou que a GRN ajuda a modelar aspectos para os quais não se possui conhecimento sobre o domínio. Ele cita como exemplo o padrão PAGAR PELA TRANSAÇÃO DO RECURSO, que o ajudou na modelagem dos aspectos sobre pagamentos, para

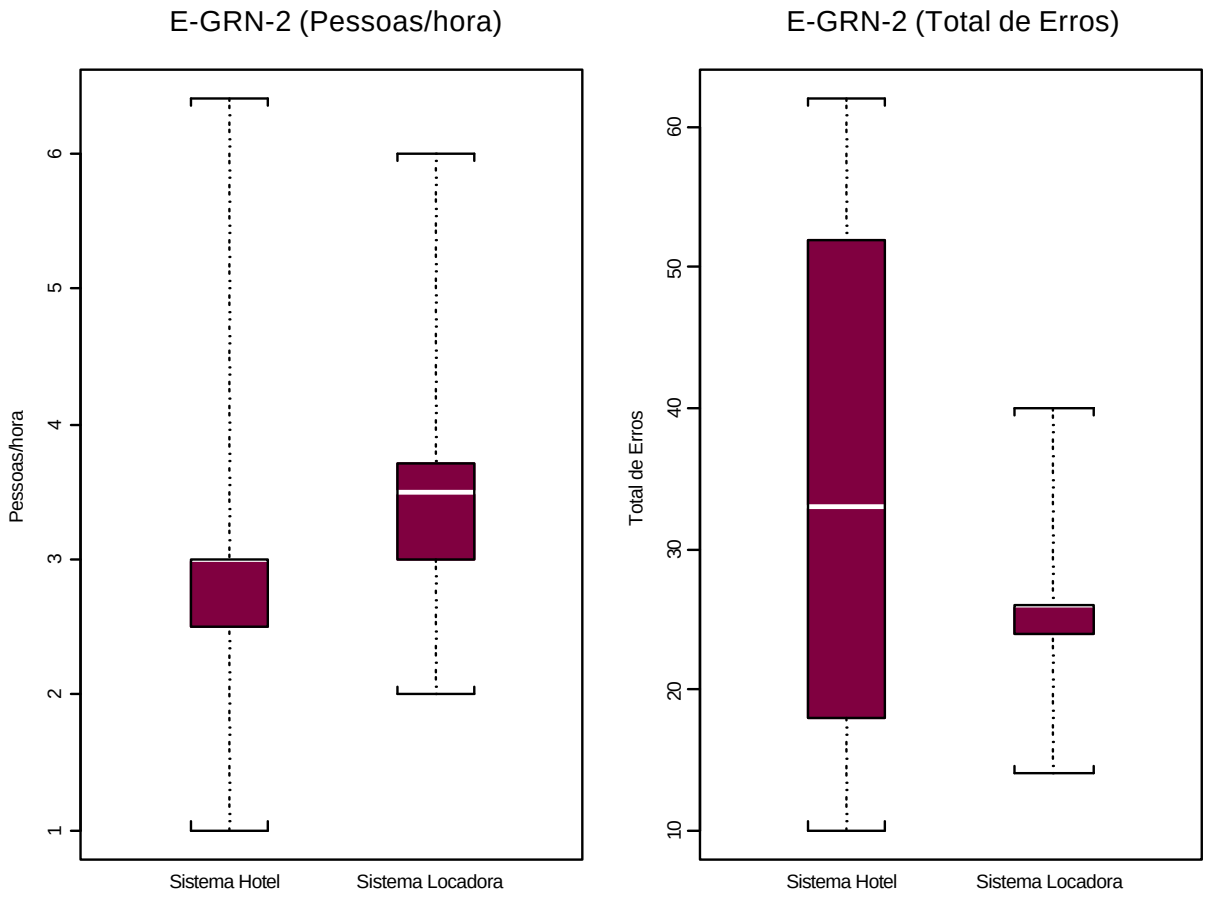


Figura 6.3: Representação gráfica dos resultados do E-GRN-2 nas duas abordagens

os quais ele não possuía entendimento suficiente. Assim como no E-GRN-1, neste experimento também houve o relato de alunos com dificuldade de aplicar a GRN na primeira vez, afirmando que novas aplicações certamente seriam mais produtivas.

Essa dificuldade dos alunos pode ser justificada, pelo menos em parte, pelo pouco tempo dedicado ao treinamento, que foi de cerca de seis horas, em comparação ao tempo de treinamento do E-GRN-1, que havia sido de treze horas. Alguns erros foram cometidos por alguns alunos na aplicação dos padrões, ocasionando a inclusão de classes indesejadas ou a omissão de classes necessárias para a correta modelagem da aplicação. Assim, uma lição aprendida nesse experimento é de que deve-se garantir que os participantes tenham entendido bem a GRN e que saibam utilizá-la. Por exemplo, o treinamento deve incluir pelo menos um exercício de modelagem usando a GRN. É interessante notar, na Figura 6.3, uma concentração maior do número de erros cometidos usando a abordagem GRN. Isso confirma que, usando a linguagem de padrões, os alunos têm tendência

de errar uniformemente, isto é, os mesmos tipos de erros são cometidos pela maioria dos alunos, o que poderia ser melhorado com mais experiência e mais tempo de treinamento.

Uma outra observação feita por um aluno foi com referência à segurança proporcionada pela GRN na orientação da modelagem, principalmente com relação à definição das classes, atributos e relacionamentos. Vários alunos apresentaram dificuldade na modelagem dos relacionamentos entre classes durante a primeira etapa do projeto e, com o uso da GRN, tiveram redução de mais de 50% no número de erros desse tipo. Apenas um dos alunos participantes disse sentir insegurança após o término da modelagem, questionando a respeito do uso correto do processo. As questões apresentadas foram: Será que algum detalhe ficou perdido durante a modelagem? Foram aplicados os padrões corretos? Deixei de aplicar algum padrão? Identifiquei os recursos de negócio corretamente? O próprio aluno disse que essa insegurança deve ter sido causada pela falta de prática de uso da linguagem.

6.3.4 E-GRN-3: Avaliação da GRN na modelagem de sistemas

Definição do Experimento

Objeto de Estudo: Linguagem de padrões GRN.

Propósito: Avaliar a utilização da GRN na modelagem de sistemas de gestão de recursos de negócios.

Foco qualitativo: Facilidade de modelagem do sistema.

Perspectiva: A perspectiva é em relação a desenvolvedores de sistemas no domínio da GRN.

Contexto: o experimento foi realizado em novembro de 2001 por vinte estudantes como sujeitos, divididos em grupos, tendo como material básico a linguagem de padrões GRN, um processo de uso da mesma e treinamento prévio de cerca de três horas/aula. O tempo reduzido foi devido às circunstâncias nas quais o experimento foi realizado: dispunha-se apenas de um sábado, no qual poderiam ser utilizados os períodos da manhã e da tarde no experimento. Participaram do experimento alunos do curso de especialização “Tecnologia da Informação”, ministrado pela UNESP-Campus de Presidente Prudente, em parceria com o SENAC-Presidente Prudente, SP. É importante salientar que os alunos já atuavam como profissionais de informática, conforme pode ser visto nas tabelas com o perfil dos participantes.

Planejamento do Experimento

Seleção do Contexto: idem ao contexto do E-GRN-1.

Definição da Hipótese: idem ao E-GRN-1.

Seleção das variáveis:

Variáveis Independentes: idem ao E-GRN-1.

Variáveis Dependentes: idem ao E-GRN-1.

Seleção dos sujeitos: idem ao E-GRN-1, mas com vinte alunos de curso de especialização, divididos em seis grupos de três ou quatro alunos cada.

Projeto do Experimento: A Tabela 6.13 mostra a divisão dos grupos e sistemas nas duas etapas do experimento.

Tabela 6.13: Projeto do E-GRN-3

Grupo	Etapa 1: Modelagem com a UML	Etapa 2: Modelagem com a GRN
1	Hotel	Locadora de Carros
2	Locadora de Carros	Hotel
3	Hotel	Locadora de Carros
4	Locadora de Carros	Hotel
5	Hotel	Locadora de Carros
6	Locadora de Carros	Hotel

Instrumentação: idem ao E-GRN-1.

Avaliação da Validade: idem ao E-GRN-1.

Operação do Experimento

Preparação: idem ao E-GRN-1.

Participantes: Participaram do experimento vinte alunos de especialização divididos em seis grupos mas, devido à desistência de alguns grupos (G3 e G4) na entrega da segunda parte do projeto, restaram treze alunos, cujo perfil é resumido nas Tabelas 6.14, 6.15 e 6.16.

Tabela 6.14: Área de interesse dos alunos do E-GRN-3

Área de Interesse	Número de alunos	Percentual
Redes/Sist Distribuídos	2	15%
Banco de Dados	2	15%
Banco de Dados/Engenharia de Software	1	8%
Inteligência artificial	2	15%
Engenharia Soft/Sist. Inform.	6	46%
Computação gráfica	0	0%
Hipermídia	0	0%
Hardware	0	0%
Total de alunos que respondeu	**13**	**100%**

Tabela 6.15: Experiência dos alunos do E-GRN-3

Experiência no domínio tratado	Número de alunos	Percentual
Sem experiência	1	8%
Realizou projetos durante disciplinas de graduação/pós graduação usando análise estruturada	1	8%
Realizou projetos durante disciplinas de graduação/pós graduação usando análise orientada a objetos	6	46%
Desenvolveu, profissionalmente, até 3 projetos nesse domínio	3	23%
Desenvolveu, profissionalmente, mais de 4 projetos nesse domínio	2	15%
Total de alunos que respondeu	**13**	**100%**

Tabela 6.16: Cargos desempenhados pelos alunos do E-GRN-3

Cargo	G1	G2	G5	G6	Total	%
Administrador de rede	1				1	8%
Analista de sistemas	1	1			2	15%
Professor	2	1	3		6	46%
Programador				2	2	15%
Encarregado administrativo		1			1	8%
Assistente de diretoria				1	1	8%
Total	4	3	3	3	13	100%

Execução: similar ao E-GRN-1, exceto quanto à organização dos alunos, que trabalharam em grupos de três a quatro pessoas, e quanto ao prazo de entrega dos modelos obtidos. Neste experimento, a primeira etapa foi feita no período da manhã, sendo subdividida em duas partes: treinamento de uma hora a respeito da UML e reunião dos grupos para modelagem do sistema. Ainda antes do almoço o modelo resultante foi entregue. O treinamento para a segunda etapa foi executado no mesmo dia, no período da tarde, e a segunda tarefa foi atribuída aos alunos para que entregassem no prazo de duas semanas.

Validação dos Dados: Dois grupos não entregaram a segunda parte do projeto, que consistia da modelagem usando a GRN. Portanto, esses grupos foram descartados e dos vinte alunos que iniciaram o projeto, restaram treze alunos com resultados a serem avaliados. Os formulários distribuídos aos demais participantes foram conferidos para verificar se foram preenchidos corretamente. Algumas perguntas foram feitas para assegurar que seguiram as recomendações sugeridas.

Análise e Interpretação dos Resultados

A Tabela 6.17 mostra os resultados obtidos na primeira etapa do E-GRN-3 (análise do sistema usando um processo *ad hoc* e a notação UML), enquanto a Tabela 6.18 mostra os resultados obtidos na segunda etapa do E-GRN-3 (análise do sistema usando a GRN). A observação direta dos resultados indica um maior tempo médio de modelagem na abordagem GRN e um menor número médio de erros cometidos. A análise estatística não foi feita devido ao pequeno número de conjuntos de dados neste experimento.

Tabela 6.17: Resultados do E-GRN-3 – Abordagem *Ad hoc*

Grupo	Pessoas/hora	#ent	#atr	#met	#rel	#car	Total de erros	Nota
Sistema Hotel								
1	7,00	1,00	21,00	11,00	3,00	4,00	40	6,00
5	6,00	1,00	13,00	15,00	6,00	16,00	51	4,75
Média	**6,50**	**1,00**	**17,00**	**13,00**	**4,50**	**10,00**	**45,5**	**5,38**
Sistema Locadora de Carros								
2	5,70	1,00	4,00	2,00	2,00	7,00	16	8,50
6	8,00	1,00	17,00	11,00	5,00	8,00	42	5,75
Média	**6,85**	**1,00**	**10,50**	**6,50**	**3,50**	**7,50**	**29**	**7,13**
Média Geral	**6,68**	**1,00**	**13,75**	**9,75**	**4,00**	**8,75**	**37,25**	**6,25**

Tabela 6.18: Resultados do E-GRN-3 – Abordagem GRN

Grupo	Pessoas/hora	#ent	#atr	#met	#rel	#car	Total de erros	Nota
Sistema Locadora de Carros								
1	8,00	1,00	8,00	9,00	3,00	5,00	26	7,50
5	6,00	2,00	9,00	3,00	4,00	8,00	26	7,75
Média	**7,00**	**1,50**	**8,50**	**6,00**	**3,50**	**6,50**	**26**	**7,63**
Sistema Hotel								
2	8,40	1,00	17,00	0,00	2,00	3,00	23	8,25
6	10,50	9,00	28,00	6,00	10,00	9,00	62	3,50
Média	**9,45**	**5,00**	**22,50**	**3,00**	**6,00**	**6,00**	**42,5**	**5,88**
Média Geral	**8,23**	**3,25**	**15,50**	**4,50**	**4,75**	**6,25**	**34,25**	**6,75**

Discussão

Em relação ao E-GRN-1 e E-GRN-2, o E-GRN-3 foi o experimento no qual o uso da GRN apresentou maior dificuldade e menor vantagem. O tempo gasto na modelagem foi semelhante para os três experimentos, levando em conta o número de participantes (os grupos do E-GRN-1 e E-GRN-3 possuíam de três a quatro alunos e o E-GRN-2 foi feito individualmente), pois o valor mostrado na tabela deve ser dividido pelo número de pessoas que executou a modelagem. Porém, o número de erros cometidos pelos alunos foi visivelmente maior no E-GRN-3. Novamente, como explicado na avaliação do E-GRN-2, pode-se justificar essa dificuldade dos alunos pelo pouco tempo dedicado ao treinamento, que foi de cerca de três horas, em comparação ao tempo de treinamento do E-GRN-1, que havia sido de treze horas. Embora esse fato indique a necessidade do correto entendimento da GRN para que se obtenha êxito na sua aplicação, foram feitos outros usos da GRN sem treinamento prévio, visando obter mais evidências que confirmem essa necessidade.

Outro ponto que pode ser discutido em relação aos três experimentos é sobre a utilidade da GRN ser influenciada pela experiência de seus usuários. Parece haver uma tendência de que ela seja mais útil para desenvolvedores inexperientes do que para os que já têm alguma prática na modelagem e desenvolvimento de sistemas. Isso pode ser justificado pelo fato do desenvolvedor experiente já possuir em mente os padrões com que trabalhou em projetos anteriores, mesmo que não tenha consciência disso, ou seja, os padrões são estruturas que ele aprendeu intuitivamente a usar ao longo dos anos e fazem parte de sua experiência pessoal. Assim, ele prefere fazer a modelagem diretamente do que usando padrões aos quais não está habituado, precisando de tempo e treinamento para se adaptar aos novos padrões. Um resultado que ajuda a confirmar essa hipótese

é o fato de ter aumentado o número de entidades erradas na modelagem com a GRN, tanto no E-GRN-2 quanto no E-GRN-3, que tinham participantes com mais experiência. A análise dos modelos produzidos indica que esses erros ocorreram porque os alunos incluiram entidades que não faziam parte dos requisitos, por estarem presentes nos padrões, ou deixaram de incluir entidades por não terem aplicado o padrão correto.

6.3.5 Outras avaliações

Outras avaliações são descritas a seguir, com o objetivo de mostrar vantagens e desvantagens do uso da GRN na modelagem de sistemas, sem intenção de comparação com outras técnicas mas, sim, de enumerar algumas lições aprendidas durante sua realização. Essas avaliações referem-se a relatos de alunos que usaram a GRN para modelar alguns sistemas, mas sem haver rigor nesse uso. Dentre os sistemas modelados estão um sistema para biblioteca de universidade, um sistema para oficina de conserto de aparelhos eletrônicos e um sistema para controle de clínica veterinária.

Esses alunos estudaram a GRN e seu processo de uso, sem nenhum treinamento adicional, e a aplicaram para modelar um ou mais sistemas. O objetivo principal foi de obter informações de como seria o uso da GRN sem treinamento dos participantes, já que em todos os demais experimentos/usos havia sido dado um treinamento prévio, variando de três a treze horas, com melhor desempenho dos participantes com maior tempo de treinamento.

A aluna que modelou os sistemas de biblioteca e oficina de conserto de aparelhos eletrônicos afirmou que, como possuía conhecimento no domínio ao qual a linguagem GRN está inserida, seu aprendizado foi rápido e sem maiores dificuldades. Ela prontamente identificou os requisitos que seriam ou não atendidos pela GRN e, conseqüentemente, pelo GREN, e separou os requisitos não atendidos em dois grupos: requisitos específicos do sistema modelado e requisitos generalizáveis. Os requisitos específicos foram analisados e deram origem a novas classes, atributos ou relacionamentos no modelo do sistema. O mesmo ocorreu com os requisitos generalizáveis, mas eles foram documentados para serem incorporados à próxima versão da GRN e do GREN para que, nos seus próximos usos, já façam parte dos requisitos cobertos.

Outros dois alunos utilizaram a GRN, individualmente, para modelar um sistema de controle de clínica veterinária. Em ambos os casos foram relatados problemas para interpretar corretamente os padrões. Um dos alunos encontrou dificuldade para modelar as consultas que os veterinários realizam nos animais, pois a GRN oferece duas maneiras de modelar essa situação: a consulta pode ser uma aplicação do padrão LOCAR O RECURSO (no caso o tempo do veterinário é o recurso "locado") ou pode ser uma aplicação do padrão MANTER O RECURSO (o animal é o recurso "consertado"). Esse aluno acabou utilizando ambos os padrões e precisou de ajuda para decidir o que fazer para encerrar a modelagem. A outra aluna não considerou a possibilidade de utilizar o padrão MANTER O RECURSO, mas apenas o padrão LOCAR O RECURSO. Assim, ela não

conseguiu cobrir toda a funcionalidade do sistema e também precisou recorrer ao suporte para prosseguir a modelagem.

Esses dois casos de uso da GRN foram úteis para detectar alguns problemas na descrição da interação entre os padrões. Assim, o texto da GRN passará por uma revisão buscando melhorar a documentação dos padrões LOCAR O RECURSO e MANTER O RECURSO. Serão incluídos exemplos de sistemas que podem ser modelados usando padrões diferentes e serão mostradas as vantagens e desvantagens de modelá-los de uma ou de outra maneira.

Essas correções a serem feitas na GRN objetivam possibilitar seu uso sem treinamento, que é uma das propriedades desejáveis de uma linguagem de padrões: ela deve ser auto-explicativa, de forma que um usuário possa, após estudá-la, aplicá-la sem maiores dificuldades. Afinal, a comunidade de padrões possui como filosofia a disponibilização pública de todo e qualquer padrão ou linguagem de padrões, para que outros desenvolvedores possam aproveitar-se do conhecimento que eles proporcionam sobre a experiência na solução dos problemas de desenvolvimento de software. Assim, deve ser possível aplicar padrões somente com as informações neles contidas, sem a necessidade de recorrer ao apoio de outras fontes de informação. Esse é o motivo pelo qual, nas Conferências sobre padrões de software, os artigos sobre padrões e linguagens de padrões são discutidos em sessões de mais de uma hora, nas quais o autor deve permanecer calado quase que todo o tempo. É uma forma de permitir que ele observe a visão de outras pessoas a respeito do que está documentado no padrão e possa modificar o que for necessário, para melhor entendimento do padrão.

6.4 Avaliação da utilidade de uma linguagem de padrões na construção de um framework correspondente

6.4.1 Visão Geral

A avaliação das vantagens e desvantagens do uso de uma linguagem de padrões para guiar a construção de um framework correspondente é feita por meio da avaliação do uso da GRN na construção do GREN (sub-seção 6.4.2) e da avaliação do uso da LV (Ré et al., 2001) na construção do Qd+ (Ré et al., 2002; Ré e Masiero, 2002) (sub-seção 6.4.3). Novos usos poderiam ser conduzidos para melhor validar esse processo e, por isso, os resultados aqui apresentados estão vinculados aos exemplos e poderiam ser diferentes para outros pares “linguagem de padrões e framework”. Entretanto, a opinião da doutoranda é que os resultados obtidos são intuitivos e que o processo proposto têm o potencial de ser aplicado a outros domínios específicos no mesmo nível do domínio de gestão de recursos de negócios, como por exemplo seguradoras, hospitais, escolas, órgãos governamentais e bancos.

6.4.2 A GRN na construção do GREN

Conforme já relatado na seção 4.3, a GRN foi utilizada ao longo do processo de construção do GREN, desde a identificação dos pontos variáveis, passando pelo projeto da hierarquia de classes, até a implementação, documentação e validação do framework. A seguir, faz-se um resumo dos principais pontos nos quais a GRN mostrou-se útil no processo de construção do GREN:

- Na fase de identificação dos pontos variáveis, em um total de quarenta pontos variáveis, trinta e seis foram identificados a partir da GRN.
- Na fase de projeto da hierarquia de classes do GREN, foram utilizados os diagramas de classes dos quinze padrões da GRN, que juntamente com o conjunto de pontos variáveis obtido na fase anterior, serviram de base para o projeto das classes abstratas que garantiram a flexibilidade desejada ao GREN.
- Na fase de implementação, a GRN foi utilizada para implementar o GREN de forma gradual, seguindo seqüencialmente seus quinze padrões. Completando a fase de implementação, foi elaborada a documentação do GREN, totalmente baseada na GRN, visando facilitar sua instanciação para aplicações específicas.
- Na fase de validação do GREN a GRN foi utilizada para guiar a instanciação de três aplicações específicas, escolhidas de forma a exercitar todos os padrões da GRN.

Concluindo, pode-se dizer que a GRN foi muito útil na construção do GREN, permitindo a generalização desse processo, conforme proposto na seção 4.2, para que outros desenvolvedores possam tirar proveito da experiência adquirida.

6.4.3 A LV na construção do Qd+

O processo de construção de um framework com base em uma linguagem de padrões, descrito na seção 4.2, foi também utilizado na construção do framework Qd+ (Ré et al., 2002; Ré e Masiero, 2002), com base na linguagem de padrões LV (Ré et al., 2001), ambos para o domínio de gestão de leilões virtuais. Esse trabalho foi executado durante uma pesquisa de mestrado que ocorreu paralelamente a esta pesquisa de doutorado.

Da mesma forma que na construção do GREN, na construção do Qd+ a LV foi utilizada para apoiar todo o processo, desde a identificação dos pontos variáveis até a implementação e validação do framework. Todos os dezessete pontos variáveis do Qd+ foram encontrados na LV. O projeto e implementação do Qd+ teve algumas particularidades devido à plataforma na qual os sistemas

resultantes devem executar, que é a Web. Assim, outros recursos foram necessários para permitir sua implementação, como os pacotes *VisualWave* e *Smalltalk Server Pages*.

A construção do Qd+ com base na LV indicou que o processo proposto nesta tese é viável de ser executado em outros domínios. Assim como no GREN, a documentação do Qd+ também faz o mapeamento dos padrões da LV para as classes do framework a serem especializadas para produzir aplicações específicas, o que facilita a instanciação de tais aplicações, como descrito na seção 6.5.3.

6.5 Avaliação da utilidade de uma linguagem de padrões na instanciação do framework associado

6.5.1 Visão Geral

As vantagens de utilizar uma linguagem de padrões para instanciar o framework a ela associado foram identificadas por meio de um experimento feito com a GRN e o GREN (sub-seção 6.5.2), pelo uso da LV na instanciação do Qd+ (sub-seção 6.5.3) e por meio de diversos outros usos da GRN na instanciação do GREN (seção 6.5.4). Deve-se ressaltar que os resultados são parciais, pois referem-se a exemplos específicos e que, portanto, podem ser diferentes se forem considerados outros pares “framework e linguagem de padrões”.

6.5.2 E-GRN-GREN: Avaliação da GRN na instanciação do GREN

Definição do Estudo de caso

Objeto de Estudo: Linguagem de Padrões GRN e Framework GREN

Propósito: Avaliar a utilização do framework GREN em relação a duas abordagens possíveis: instanciação usando a linguagem de padrões GRN, denominada “abordagem GRN”, e instanciação *Ad hoc*, denominada “abordagem *ad hoc*”. A abordagem *ad hoc* consiste na utilização de exemplos como base para o processo de instanciação, apoiada pela documentação do framework. Foi escolhida com base em um estudo feito por Shull et al. (2000), que concluiu ser esta a abordagem que produz melhores resultados de produtividade na instanciação de frameworks.

Foco qualitativo: facilidade de uso do framework GREN em relação às duas abordagens.

Perspectiva: A perspectiva é em relação ao usuário do framework, ou seja, o desenvolvedor de aplicações que utiliza o framework durante o desenvolvimento.

Contexto: o estudo de caso foi realizado por estudantes de pós-graduação (mestrado e doutorado) como sujeitos, tendo como material básico manuais de uso do framework pelas duas abordagens, o framework em si, e algum treinamento prévio na linguagem de programação do framework (Smalltalk VisualWorks).

Planejamento do Estudo de Caso

Seleção do Contexto: O estudo de caso foi conduzido de forma independente pelos estudantes, não houve comunicação entre eles, mas eles tiveram liberdade para estipular seus próprios horários para realização das tarefas, desde que anotassem na planilha fornecida os horários exatos de início e fim das atividades. Nenhum dos estudantes conhecia o framework GREN, a linguagem de padrões GRN e a linguagem de programação Smalltalk. O problema utilizado no estudo de caso foi de um sistema prático (embora pequeno) para uma clínica veterinária, cujos requisitos podem ser encontrados no Apêndice C. Assim, o estudo é válido em um contexto específico no domínio de sistemas de informação.

Definição da Hipótese:

Hipótese nula: O tempo total gasto para instanciar o GREN usando a abordagem GRN é maior ou igual ao tempo gasto usando a abordagem *ad hoc*.

Hipótese alternativa: O tempo total gasto para instanciar o GREN usando a abordagem GRN é menor do que o tempo gasto usando a abordagem *ad hoc*.

Seleção das variáveis:

Variáveis Independentes: *Abordagem para Instanciação do Framework GREN*: alguns estudantes utilizaram a abordagem GRN e outros utilizaram a abordagem *ad hoc*; *Experiência do estudante*: alguns já programaram com linguagem orientada a objetos, outros não.

Variáveis Dependentes: Tempo total gasto na implementação do sistema exemplo, número total de classes e métodos criados na versão final do sistema, tempo gasto na fase de depuração e testes da nova aplicação.

Seleção dos sujeitos: A técnica de escolha foi a amostragem por conveniência (as pessoas mais próximas e mais convenientes são selecionadas como sujeitos), visto que são alunos matriculados em disciplina de pós-graduação específica: Seminários Avançados em Engenharia de Software. A divisão dos dois grupos não foi aleatória: dois dos cinco alunos foram escolhidos para utilizar a abordagem GRN porque trabalhariam, no futuro, com o GREN/GRN em trabalho de mestrado/doutorado. Os demais alunos foram escolhidos para utilizar a abordagem *ad hoc*.

Projeto do Estudo de caso: A Tabela 6.19 mostra a distribuição dos alunos entre as duas abordagens estudadas.

Tabela 6.19: Projeto do E-GRN-4

Aluno	Abordagem Ad-hoc	Abordagem GRN
1	X	
2		X
3	X	
4		X
5	X	

Instrumentação: O material fornecido aos sujeitos para realização do estudo de caso foi composto de: diretrizes para realização do estudo de caso; documento de requisitos do sistema da clínica veterinária; manual de instanciação usando a abordagem GRN ou manual de instanciação usando a abordagem *ad hoc*[2]; e formulários para coleta de dados do estudo de caso (tais formulários podem ser encontrados no Apêndice E).

Avaliação da Validade: idem ao E-GRN-1.

Operação do Estudo de Caso

Preparação: após estabelecido contato com os participantes, foi preparada a instrumentação necessária: foram escritos os manuais de instanciação[3], foram organizados os arquivos com o código-fonte dos exemplos e foram impressos os questionários com diferentes tipos de questões, dependendo da abordagem utilizada. Foi garantido anonimato aos participantes.

Participantes: Participaram do experimento cinco alunos de mestrado e doutorado da disciplina de Seminários Avançados em Engenharia de Software do ICMC-USP. Quanto ao perfil desses alunos, eles são da área de Engenharia de Software, todos conhecem pelo menos uma linguagem de programação orientada a objetos, embora alguns não possuam experiência prática de desenvolvimento usando essa linguagem. Apenas um aluno afirma possuir prática profissional no desenvolvimento de sistemas e outro aluno executa manutenção em sistemas no domínio.

Execução: O experimento foi realizado pelos alunos individualmente, fora de sala de aula. A linguagem de padrões GRN foi enviada aos alunos que usariam a abordagem GRN. Foram fornecidos disquetes/CDs com o material necessário para executar o ambiente VisualWorks

[2]BRAGA, R. T. V.; MASIERO, P. C. Manual de Instanciação do Framework GREN pela abordagem Ad Hoc. ICMC- USP - São Carlos, Documento de Trabalho, 67 p., 2002

[3]O manual de instanciação pela abordagem GRN já existia e foi apenas revisado, enquanto o manual de instanciação pela abordagem *ad hoc* foi escrito especialmente para este experimento

e para instalação do GREN. Foram distribuídos os formulários para preenchimento de tempos e anotação de dificuldades encontradas. Foi garantida liberdade aos participantes para adaptar o processo fornecido na medida em que fossem encontradas melhores formas de executar o projeto. Também foi estabelecida uma forma de comunicação com os monitores do experimento, por intermédio de correio eletrônico ou consultas pessoais, para que os alunos pudessem fazer perguntas e esclarecer dúvidas.

Validação dos Dados: Os formulários foram preenchidos corretamente e a aplicação final gerada pelos participantes funcionou adequadamente. Os alunos seguiram as recomendações sugeridas. Portanto, nenhum dos sujeitos foi descartado.

Análise e Interpretação dos Resultados

A Tabela 6.20 mostra os resultados obtidos no E-GRN-GREN, mostrando os tempos totais gastos no desenvolvimento, tempo parcial gasto na modelagem, tempo gasto no teste e depuração do sistema, número de classes e métodos criados. Os alunos que utilizaram a abordagem GRN gastaram mais tempo no desenvolvimento do sistema e tiveram mais dificuldade com relação à fase de implementação no VisualWorks. Como a amostragem é muito pequena, não foi realizada a análise estatística dos resultados.

Tabela 6.20: Resultados do E-GRN-GREN

Sistema Veterinária					
Aluno	**Tempo total (horas)**	**Tempo modelagem**	**Tempo testes**	**#Classes**	**#Métodos**
Abordagem Ad-hoc					
1	67,00	13,00	20	34,00	227,00
2	50,00	5,00	16	54,00	299,00
3	26,00	13,00	8	43,00	310,00
Média	**47,67**	**10,33**	**14,67**	**43,67**	**278,67**
Abordagem GRN					
4	78,00	11,00	32	39,00	260,00
5	76,00	5,00	16	37,00	297,00
Média	**77,00**	**8,00**	**18**	**38,00**	**278,50**

Discussão

A falta de treinamento dos participantes ajuda a justificar esses resultados. Enquanto a abordagem *ad hoc* é bastante intuitiva, a abordagem GRN requer treinamento, principalmente quanto ao processo de instanciação do GREN utilizando seu *cookbook*. O maior tempo gasto pelos alunos (em ambas as abordagens) foi na fase de implementação, por não terem conhecimento suficiente para depurar o programa e corrigir os erros que surgiram. Os alunos que utilizaram a abordagem *ad hoc* recorreram ao código-fonte do exemplo para tentar sanar as dúvidas, o que definitivamente os ajudou na maioria das vezes. Já os alunos que utilizaram a abordagem GRN dispenderam bastante tempo tentando encontrar a solução, ora relendo o manual de instanciação, ora recorrendo

ao apoio da monitora. Assim, a abordagem *ad hoc* mostrou-se mais eficiente na depuração de erros por usuários que conhecem pouco o framework e a linguagem de programação na qual ele está escrito. Outros problemas relatados nos formulários individuais distribuídos aos participantes, indicam situações que podem ter influenciado os resultados, como por exemplo: dois alunos (entre os que levaram mais tempo na instanciação) relataram que "normalmente são lerdos para implementar sistemas e aprender novas técnicas" e um aluno relatou ter gasto sete horas somente na instalação do VisualWorks, devido a problemas técnicos (a média de tempo necessária aos demais alunos para instalação foi de uma hora e quinze minutos).

Se forem planejados outros experimentos de desenvolvimento usando o GREN de forma manual, deverá ser providenciado o treinamento dos participantes, de preferência com instanciação completa de um exemplo, desde a modelagem até a programação no ambiente VisualWorks. Isso talvez não ocorra, visto que a existência do GREN-Wizard dispensa a etapa de implementação do código. No entanto, fica como uma lição aprendida para o caso de outros frameworks a serem desenvolvidos com base em uma linguagem de padrões.

O E-GRN-GREN levantou outros pontos interessantes que merecem ser discutidos. Em primeiro lugar, percebeu-se que o GREN pode ser instanciado com sucesso usando exemplos de aplicações desenvolvidas com base nele, ao invés de utilizar o processo com base na GRN. Até então, todos os usos do GREN haviam sido feitos com base na GRN, já que tanto a documentação quanto o processo de instanciação do GREN estavam totalmente voltados para ela. Após criar a documentação e o processo que permitem a instanciação do GREN por meio de exemplos, percebeu-se que esta é uma alternativa viável, principalmente quando o sistema alvo possui funcionalidade muito parecida com um sistema já desenvolvido.

No entanto, nota-se que o uso por exemplos somente se justifica na ausência de um *Wizard* para auxílio à instanciação, já que esse último certamente propicia maior rendimento, tanto no caso de sistemas que possuem outros similares já instanciados anteriormente, quanto no caso de sistemas a serem instanciados pela primeira vez.

O segundo ponto interessante refere-se à modelagem dos sistemas pelos alunos que utilizaram a abordagem *ad hoc*. Embora o processo apresentado no manual de instanciação recomendasse que a análise do sistema fosse feita antes de iniciar o estudo dos exemplos, dois dos três alunos consideraram mais fácil iniciar pelos modelos de classes do exemplo e conduzir sua análise com base neles, identificando similaridades e substituindo classes do exemplo pelas classes da nova aplicação. Essa técnica mostrou-se eficiente, principalmente porque os exemplos continham subsistemas intuitivamente equivalentes ao sistema de veterinária. Outro resultado interessante foi que os três alunos escolheram os mesmos exemplos para basearem seu desenvolvimento: o primeiro exemplo, da oficina de veículos, foi escolhido para modelar a consulta feita aos animais da clínica

veterinária e o segundo exemplo, da loja de venda e aluguel de produtos para festas, foi utilizado para modelar a venda de produtos na loja da clínica.

O terceiro aspecto que veio à tona durante esse experimento foi quanto ao aprendizado que se consegue a respeito do GREN e da linguagem de programação específica (Smalltalk) após a instanciação. No caso da abordagem *ad hoc* alguns alunos relataram que não precisaram saber muito sobre Smalltalk e sobre o GREN para fazer a instanciação, já que utilizaram cópia de código e apenas modificaram os conteúdos dos métodos de acordo com as classes do sistema específico. Outros alunos disseram que aprenderam bastante, pois precisaram navegar na hierarquia de classes do GREN para entender o que deveria ser feito durante a instanciação. Isso mostra que houve diversos tipos de uso da abordagem *ad hoc*: alguns a seguiram estritamente e outros preferiram tomar outros caminhos para otimizar o processo. Já no caso da abordagem GRN, nesse e em outros usos feitos, foi relatado pelos alunos que não é necessário conhecer o GREN nem tampouco a linguagem Smalltalk, pois a instanciação é feita seguindo-se um roteiro sistemático que leva ao código-fonte final. Ao mesmo tempo, alguns alunos reclamam que terminaram a instanciação "sem conhecer o GREN". Na verdade, o aprendizado sobre o framework e a linguagem de programação na qual foi desenvolvido — conseguidos durante a instanciação — pode ou não ser importante, proporcionalmente a quanto o framework abrange da funcionalidade do sistema alvo desenvolvido. Se parte da funcionalidade não for coberta pelo framework, então é importante que o desenvolvedor conheça o projeto e implementação do framework para facilitar a inclusão dos requisitos não cobertos. Se esse conhecimento não tiver sido adquirido durante a instanciação, deverá ser conseguido de outra maneira, como por exemplo, mediante treinamento adicional ou estudo da documentação disponível. Se, no entanto, o framework cobrir grande parte da funcionalidade desejada, não é tão importante que o desenvolvedor o conheça em detalhes.

Alunos de ambas as abordagens relataram dificuldade na depuração do sistema, devido a pouca experiência com Smalltalk e com o ambiente VisualWorks, indicando a necessidade de maior treinamento em futuros casos. O treinamento dado a esses alunos antes do experimento consistiu em cerca de seis horas de aula sobre Smalltalk e alguns exercícios para familiarização com o ambiente VisualWorks. Em futuros casos seria necessário fazer um treinamento mais específico que incluísse a instanciação de um exemplo usando o GREN.

Finalmente, o E-GRN-GREN ajudou a confirmar um dos resultados obtidos por Shull et al. (2000): participantes que usam a abordagem *ad hoc*, que é baseada em exemplos, possuem uma forte tendência de incluir em suas aplicações funcionalidades não previstas nos requisitos. Dois dos três alunos que usaram essa abordagem incluíram no sistema final o sub-sistema de pedido e compra de produtos da veterinária, que não consta dos requisitos, conforme pode ser observado no Apêndice C (pode-se também notar que o número de classes criadas por esses alunos foi superior ao dos alunos que fizeram a instanciação usando a abordagem GRN). Eles alegaram que esta é

uma funcionalidade desejável e que, como tal funcionalidade estava pronta no sistema exemplo e portanto, fácil de reusar, eles resolveram incluí-la. Além disso, a terceira aluna só não implementou esse sub-sistema porque teve dúvidas durante a análise e pediu ajuda ao suporte, que acabou tendo influência em sua decisão de não incluir funcionalidade fora dos requisitos fornecidos.

6.5.3 A LV na instanciação do Qd+

A linguagem de padrões LV apóia a instanciação de aplicações de leilões virtuais usando o framework Qd+. Foi realizado o uso do Qd+, pelo próprio mestrando que o implementou, para desenvolver um sistema de leilões virtuais com funcionalidade similar à do *site* iBazar[4], que oferece o serviço de leilões virtuais na Web. Da mesma forma que no GREN, a linguagem de padrões LV foi utilizada para obter o modelo da aplicação concreta e, depois, foi feito o mapeamento das classes dos padrões e variantes para as classes correspondentes no Qd+. A implementação exigiu uma etapa adicional, de programação da GUI da aplicação, já que devem ser criados os arquivos com código HTML para visualização dos formulários. Além disso, para executar aplicações Web no Visualworks é necessário realizar algumas atividades e configurar alguns componentes para que o servidor Web funcione corretamente[5].

Portanto, o processo aqui proposto pôde ser utilizado para outro par "linguagem de padrões e framework" no domínio de gestão de leilões virtuais, o que indica sua adequação para domínios afins.

6.5.4 Outros Casos de Uso da GRN na instanciação do GREN

Diversos casos de uso da GRN na instanciação do GREN foram realizados para identificar vantagens e desvantagens. Os três primeiros usos, já citados na seção 6.4.2, tiveram como objetivo testar e aprimorar a documentação do GREN, pela própria autora desta tese, para averiguar se essa documentação seria suficiente para que outros usuários pudessem utilizar a GRN na instanciação do GREN. Os sistemas utilizados foram: uma video-locadora, com vinte e seis classes e um mil e duzentas LOC; uma loja de venda e aluguel de produtos para festas, com trinta classes e cerca de um mil e trezentas LOC; e uma oficina mecânica de veículos, com vinte e três classes e cerca de um mil e cem LOC. Os estudos cumpriram seu propósito, já que diversos erros foram encontrados, tanto na documentação do processo de uso da GRN para instanciar o GREN quanto no próprio código do GREN, e puderam ser corrigidos antes que o framework fosse utilizado por usuários externos.

[4] `http://www.ibazar.com.br`

[5] Maiores informações sobre os detalhes de instanciação do Qd+ podem ser encontrados na dissertação de mestrado correspondente (Ré e Masiero, 2002).

Os usos subseqüentes do GREN foram feitos para verificar a viabilidade de uso do processo proposto no manual de instanciação (ou *cookbook*) por pessoas que não conheciam o projeto do GREN. Inicialmente ele foi utilizado para implementar o sistema de hotel e o sistema de locadora de carros (já utilizado em experimentos anteriores, descritos na seção 6.3) pelos mesmos alunos que participaram do E-GRN-1. Partindo do resultado do E-GRN-1, que foi o modelo de análise do sistema, devidamente corrigido de acordo com o gabarito proposto, metade dos grupos realizou a implementação do sistema de hotel e, a outra metade, do sistema de locadora de carros. Os alunos utilizaram o *cookbook* do GREN para guiar a instanciação, no qual é explícita a utilização da GRN durante todo o processo. O estudo de caso cumpriu seu propósito, visto que os alunos puderam efetuar a instanciação de maneira sistemática, sem necessidade de conhecimento da estrutura interna do GREN. As dificuldades relatadas pelos alunos foram anotadas e uma nova versão do *cookbook* foi produzida após o caso de uso.

Em outro caso de uso, uma aluna de doutorado, também do ICMC-USP, usou o GREN para instanciar um sistema real de Controle de Estoque, cujos requisitos não se adequavam totalmente ao domínio da GRN. Nesse caso, a GRN foi útil para avaliar a adequação do GREN para implementar a aplicação específica. Como a GRN não conseguiu dar cobertura a um número suficiente de requisitos do sistema de controle de estoque em mãos, concluiu-se que o uso do GREN não traria muitos ganhos, porque muita programação adicional seria necessária para implementar as funções não cobertas por ele. Mesmo assim, a aluna efetuou a instanciação referente aos requisitos cobertos, com base no *cookbook* do GREN, instanciando apenas os padrões referentes a esses requisitos. A aluna estima que foram cobertos cerca de 65% dos requisitos. Trinta e cinco classes foram criadas, num total de aproximadamente duas mil e duzentas LOC. Considera-se esse resultado satisfatório comparado à implementação do sistema partindo do nada. No entanto, como será necessário programar grande parte do código, outras alternativas de reuso poderiam ser estudadas para tentar alcançar o máximo de reuso possível.

6.6 Avaliação da utilidade de uma linguagem de padrões na construção de um *Wizard*

Desenvolver um *Wizard* para auxiliar na instanciação de um framework que tenha sido construído com base em uma linguagem de padrões, pode trazer uma série de vantagens, principalmente se o projeto desse *Wizard* for genérico, prevendo sua adaptação futura para outros pares "framework e linguagem de padrões", conforme discutido na seção 5.8. Embora o processo proposto naquela seção ainda não tenha sido utilizado na prática, ele foi derivado a partir da experiência de construção do GREN-Wizard com base na GRN, conforme descrito na seção 5.5. A generalização

foi feita com base em um caso concreto e cujos problemas certamente são representativos da classe de problemas encontrados no desenvolvimento de uma ferramenta desse tipo.

Algumas vantagens de uma linguagem de padrões na construção de um *Wizard* são:

- o módulo de especificação do domínio pode ser reaproveitado de outros desenvolvimentos de *Wizards*, já que os elementos dos padrões que devem ser representados são sempre os mesmos, por exemplo, padrões, variantes, classes, atributos, relacionamentos e ordem de aplicação dos padrões. No caso de já existir um *Wizard* genérico esse reaproveitamento é ainda mais direto.

- o projeto da GUI do *Wizard* fica mais fácil, pois deve obedecer aos padrões da linguagem. No caso de já existir um *Wizard* genérico, a GUI pode ser totalmente reaproveitada, pois seu projeto depende apenas da correta adaptação da base de dados com os meta-dados sobre a linguagem de padrões. Mesmo no caso de construção de um *Wizard* específico, tem-se em mãos o projeto arquitetural da GUI, ou seja, a estrutura geral pode ser reaproveitada, cuidando apenas das telas que mostram cada padrão individual.

- o módulo gerador de código possui algumas funções que não variam de uma linguagem de padrões para outra e que, portanto, podem ser reaproveitadas. Mesmo em um *Wizard* específico, pode-se reusar alguns algoritmos genéricos, como os fornecidos nas Figuras 5.10 e 5.11.

Acredita-se que outros desenvolvedores de frameworks possam utilizar o processo proposto para construção de seus frameworks e ferramentas de instanciação, contribuindo para o aperfeiçoamento do processo proposto à medida que outros problemas forem sendo encontrados. Um dos trabalhos futuros sugeridos na seção 7.5 é de estudar o GREN-Wizard e modificar algumas partes de seu projeto para que ele possa ser considerado um *Wizard* genérico. Isso permitirá sua adaptação para funcionamento com o framework Qd+ e a linguagem de padrões LV.

6.7 Avaliação da utilidade de uma linguagem de padrões na instanciação de um framework usando seu *Wizard*

Tendo sido o *Wizard* construído com base em uma linguagem de padrões, é intuitivo que esta seja essencial durante a utilização do mesmo. A avaliação aqui apresentada baseia-se no uso do GREN-Wizard guiado pela GRN, já que não se dispõe de outros *Wizards* construídos seguindo o processo

proposto. Uma nova avaliação poderá ser feita após a realização do trabalho de construção de um *Wizard* genérico com base na modificação do GREN-Wizard.

Diversos usos do GREN-Wizard foram realizados para identificar dificuldades, vantagens e desvantagens. O orientador desta tese foi quem primeiro usou o GREN-Wizard, após um breve treinamento destinado a prepará-lo para uma demonstração da ferramenta em uma Conferência Internacional. Sendo co-autor da GRN, foi dispensável fazer o estudo da mesma. O treinamento, que foi de cerca de uma hora, demonstrou ser suficiente para iniciar o uso do GREN-Wizard. O exemplo escolhido para o treinamento foi o SARB (ver seção 3.5.6), para o qual já se dispunha da modelagem usando a GRN. Depois disso foi escolhido um outro sistema, bem mais simples, para servir de exemplo na demonstração da ferramenta, devido ao curto tempo reservado para tal na Conferência. Algumas dificuldades no uso da GUI foram relatadas e são discutidas na seção 6.10. Quanto ao uso da GRN durante a instanciação, não houve dificuldade.

Outros usos do GREN-Wizard foram realizados[6] pela mesma aluna de doutorado que havia instanciado manualmente algumas aplicações usando o *cookbook* do GREN. Os sistemas instanciados usando o GREN-Wizard foram um sistema de biblioteca e um sistema de oficina de aparelhos eletrônicos, que já haviam sido modelados usando a GRN (ver sub-seção 6.3.5), mas ainda não tinham sido implementados. Não houve treinamento específico para essa aluna, tendo apenas sido fornecido o manual de uso do GREN-Wizard. Aqui, novamente, foram identificados alguns problemas de usabilidade na GUI do GREN-Wizard, que são discutidos na seção 6.10. A aluna também não teve dificuldade no uso da GRN durante a instanciação, tendo relatado que a interação é bastante fácil e intuitiva, ou seja, conhecendo-se a GRN é possível utilizar o GREN-Wizard sem maiores problemas. Ela mencionou que a existência de botões que habilitam e desabilitam guiam o usuário na aplicação da linguagem de padrões. Comparou o uso do GREN-Wizard à instanciação manual, dizendo que o código produzido pelo *Wizard* é semelhante ao código programado manualmente e que pode ser facilmente modificado caso necessário.

Alguns requisitos dos sistemas de biblioteca e oficina eletrônica não puderam ser atendidos pelo GREN e, conseqüentemente, pelo GREN-Wizard. Analisando esses requisitos, percebeu-se que seria desejável incluí-los no GREN, já que são funcionalidades que fazem parte do domínio e podem ser reusadas em outras futuras aplicações do GREN. Assim, a aluna se propôs a fazer uma manutenção, tanto no GREN quanto no GREN-Wizard, para incluir essas funcionalidades, que referem-se basicamente aos tipos de dados dos novos atributos incluídos nas classes dos padrões e à impressão de etiquetas. A versão 1.0 do GREN (e também a correspondente versão do GREN-Wizard) permitia a inclusão de novos atributos nas classes dos padrões, mas apenas quatro tipos de dados eram previstos: número inteiro, número flutuante, data e texto. Ao modelar o sistema de

[6]CAGNIN, M. I.; MALDONADO, J. C.; PENTEADO, R. D.; GERMANO, F. S. Manutenções no framework GREN e no GREN-Wizard motivadas pelo seu uso no processo de reengenharia de sistemas legados. Documento de trabalho, a ser publicado como Relatório Técnico do ICMC-USP, 2002.

biblioteca, a aluna percebeu que seria conveniente que o GREN-Wizard apoiasse a implementação dos seguintes tipos de dados:

Lista a partir de uma tabela: ideal para casos nos quais o valor do atributo pode ser obtido em uma tabela pré-existente do sistema ou criada durante a instanciação. Assim, no sistema de oficina eletrônica, por exemplo, é possível criar um atributo `proprietário` na classe que representa o aparelho eletrônico consertado. Isso equivale a acrescentar um relacionamento do tipo "muitos para um" entre o aparelho e seu proprietário.

Atributo multi-valorado: ideal para casos nos quais o atributo possui diversos valores vindos de uma tabela pré-existente do sistema ou criada durante a instanciação. Isso possibilita que, por exemplo, no sistema de biblioteca, possam ser cadastrados todos os autores de um determinado livro, ou seja, um novo atributo `autores` é acrescentado à classe *Livro*, cujos valores são provenientes da classe *Autor*. Isso equivale à inclusão de um relacionamento do tipo "muitos para muitos" entre *Livro* e *Autor*.

Lista discreta: esse é um caso particular de Lista a partir de uma tabela e que deve ser usado quando os valores são fixos. Por exemplo, o estado civil do leitor da biblioteca pode utilizar uma lista discreta com valores pré-definidos. Isso também equivale a criar um relacionamento do tipo "muitos para um" entre o leitor e seu estado civil.

Deve-se observar que esses novos tipos de atributos poderiam ser implementados manualmente na nova aplicação, bastando para isso um pouco de experiência com Smalltalk, mas com o apoio do GREN facilita-se ainda mais o processo de instanciação. Além disso, o GREN-Wizard também foi adaptado para incluir esses novos tipos de dados.

A Tabela 6.21 apresenta um resumo quantitativo da manutenção realizada, na qual é mostrado o tempo em horas gasto para realizar a manutenção no GREN e no GREN-Wizard, bem como o número de métodos e classes criadas, modificadas, etc. É possível observar que foi empregado um tempo maior na manutenção do GREN em relação ao do GREN-Wizard, apesar de haver menos métodos e classes criadas. A justificativa para isso é que a aluna iniciou a manutenção pelo GREN e não conhecia nem a hierarquia de classes do framework e nem a linguagem Smalltalk. Portanto nessa totalização de tempo está embutido o tempo de aprendizado da hierarquia de classes do GREN e do GREN-Wizard e da linguagem de programação Smalltalk.

Após ter realizado a manutenção no framework GREN e no GREN-Wizard, a aluna instanciou os sistemas usando o GREN-Wizard, tendo obtido vinte e três classes no sistema de biblioteca, com cerca de duas mil e setecentas LOC, e dezesseis classes no sistema de oficina eletrônica, com aproximadamente três mil e setecentas LOC. Observa-se que o número de LOC no sistema de oficina é maior comparado ao sistema de biblioteca, ao contrário do número de classes. Isso se

Tabela 6.21: Manutenção realizada no GREN e GREN-Wizard

Informações quantificadas	Manutenção GREN	Manutenção GREN-Wizard
Classes criadas	4	8
Classes modificadas	0	5
Métodos criados	57	107
Métodos modificados	0	9
Métodos removidos	0	8
Linhas código fonte criadas	265	792
Linhas código fonte adicionadas	0	105
Tempo médio (hs)	49	47,1

deve a dois fatos relativos ao sistema de oficina: o grande número de atributos incluídos nas classes e à implementação efetuada para alguns relatórios específicos.

6.8 Avaliação da utilidade de um *Wizard* no teste do framework associado

Uma outra utilidade de um *Wizard* construído seguindo o processo proposto nesta tese é no teste do framework associado. A única maneira de testar um framework é por meio de sua instanciação para diversas aplicações (Bosch et al., 1999), até que seja tomada a decisão de considerar o framework validado e liberar seu uso para desenvolvimento de aplicações reais. Entretanto, a instanciação manual do framework é trabalhosa, demorada e sujeita a erros. Assim, a existência de um *Wizard* agiliza os testes do framework.

Uma estratégia que pode ser adotada é a definição de uma série de aplicações-teste que exercitem os diversos padrões e variantes da linguagem, e usar o *Wizard* para gerar o código de tais aplicações, que podem então ser testadas. Aplicações ligeiramente diferentes de outras já testadas, por exemplo, com mudança apenas em um dos padrões ou variantes, podem ser facilmente introduzidas na GUI do *Wizard* pelo reuso da especificação de outra aplicação já registrada.

Além do framework em si, pode-se também testar o *Wizard*, já que o mapeamento dos padrões para o framework precisa passar por testes. Mais do que isso, alguns métodos-gancho do framework somente são exercitados em casos específicos de uso dos padrões. Portanto, consegue-se unir o teste do framework ao teste do *Wizard*, garantindo que o *Wizard* foi adequadamente adaptado ao par “framework e linguagem de padrões”.

Cabe observar que o uso de um *Wizard* no teste do framework só é possível se for feita uma adaptação no processo proposto nesta tese (ver Figura 8.1 no Capítulo 3). Nesse processo, o desenvolvimento do *Wizard* só é feito após o término da implementação e validação do framework. Assim, sugere-se que, para tirar proveito do *Wizard* na validação do framework, antecipe-se a etapa de criação do *Wizard*, de forma que os testes possam ser feitos com o apoio do mesmo. Se um *Wizard* genérico tiver sido construído em desenvolvimentos anteriores, então essa etapa pode

ser antecipada sem maiores problemas, pois basta adaptá-lo para o par específico “framework e linguagem de padrões”.

No caso do GREN-Wizard, o GREN já havia sido testado para algumas aplicações e pôde-se complementar esses testes com o GREN-Wizard, tendo sido encontrados erros em diversos lugares diferentes: no código do GREN, no seu *cookbook* e nas tabelas de mapeamento da GRN para o GREN. Concluindo, mesmo que se siga o processo tal qual proposto nesta tese, pode-se ainda utilizar o *Wizard* para completar ainda mais os testes efetuados.

6.9 Avaliação do Framework GREN

Em seções anteriores deste Capítulo, o GREN foi avaliado indiretamente por meio de seus usos e de um experimento (seção 6.5.2). Nesta seção comentam-se alguns pontos a serem aperfeiçoados no GREN, tanto para melhorar sua usabilidade quanto para torná-lo mais abrangente. Esses comentários são baseados em críticas e sugestões feitas pelos alunos que o usaram.

Um dos pontos que precisam ser melhorados no GREN é com relação a algumas classes participantes dos padrões que o GREN exige que sejam implementados, embora pudessem ser opcionais em muitas aplicações do domínio. Exemplos dessas classes são a Taxa de Juros e Taxa de Multa no padrão PAGAR PELA TRANSAÇÃO DO RECURSO. Como o GREN foi implementado com base na GRN, somente foram considerados opcionais os participantes que nela constavam como tal. Assim, ao implementar essas mudanças deverá ser providenciada a alteração correspondente na GRN.

O experimento E-GRN-GREN identificou um problema de projeto do GREN, relacionado ao padrão PAGAR PELA TRANSAÇÃO DO RECURSO. Esse padrão pode ser aplicado a quaisquer transações do sistema e deveria ser possível criar uma única classe para englobar pagamentos de diversas transações. Por exemplo, num sistema de clínica veterinária, os pagamentos associados à consulta do animal poderiam pertencer à mesma classe que representa os pagamentos de vendas no balcão. No entanto, o GREN não permite essa junção, sendo necessário criar duas classes distintas, uma para cada tipo de transação. O problema continua após criar essas classes, porque as subclasses referentes aos tipos de pagamento (por exemplo, dinheiro, cheque, cartão de crédito, etc.) guardam uma referência ao número da transação, e então ocorre uma falha no sistema ao tentar reaver os detalhes da parcela.

O GREN possui um gerador de relatórios bastante simples, mas que atende às necessidades básicas do sistema, ou seja, é possível gerar relatórios referentes às operações de cada padrão. Esse gerador de relatórios também é utilizado para gerar os relatórios produzidos durante o uso do GREN-Wizard. Embora os relatórios gerados apresentem as informações exigidas pelo sistema, a disposição dessas informações poderia ser melhorada por meio do uso de melhores recursos

gráficos. Além disso, uma próxima versão desse gerador de relatórios poderia incluir mais algumas funcionalidades, como por exemplo: durante a visualização do relatório em tela deveria ser possível ver cada página separadamente, no mesmo formato em que será realmente impresso, ao invés de ver o relatório em um único texto com uma barra de rolagem vertical, que é o que ocorre na versão atual.

6.10 Avaliação do GREN-Wizard

O GREN-Wizard foi avaliado em seções anteriores quanto à facilidade de uso na instanciação do GREN. Outros fatores que merecem ser discutidos são o desempenho do GREN-Wizard e suas limitações.

Algumas medidas tomadas durante o uso do GREN-Wizard são mostrados na Tabela 6.22. Elas referem-se à implementação de três sistemas pela autora desta tese: o SARB (ver seção 5.7), um sistema de hotel e um sistema para loja de venda e aluguel de produtos para festas. O número de padrões aplicados foi incluído para dar uma noção do tamanho do sistema. Por exemplo, o sistema de hotel é bem menor do que o sistema de loja para festas. Deve-se observar que essas medidas são influenciadas pela experiência de quem a utilizou, no caso a própria desenvolvedora do GREN-Wizard. Antes de iniciar o preenchimento dos formulários em tela, a GRN já tinha sido aplicada e havia sido construída uma tabela com os padrões aplicados, papéis desempenhados e atributos adicionados, conforme recomendado na seção 3.4. Uma das tarefas que demanda maior tempo no preenchimento dos formulários em tela é a inclusão de novos atributos, pois é necessário especificar seu tipo e tamanho. Além disso, se os atributos forem de tipos especiais, como por exemplo valores vindos de uma lista, deve-se fornecer os detalhes sobre essa lista, o que aumenta o tempo de preenchimento das telas.

Tabela 6.22: Medidas de uso do GREN-Wizard

Medida	SARB	Hotel	Loja Festas
Nro de padrões utilizados	9	5	13
Total de atributos incluídos	7	16	15
Nro de classes criadas	26	21	41
Nro de métodos criados	195	181	266
Nro de tabelas criadas na base de dados	14	13	22
Nro de linhas de código geradas	1400	1300	2000
Tempo gasto na geração do código	20s	23s	37s
Tempo gasto no preenchimento dos formulários em tela	7min30s	10min	15min

O tempo gasto na geração refere-se ao tempo aproximado dispendido pelo módulo de geração de código para criar as classes, métodos e tabelas MySQL. Esse tempo é bastante influenciado pela quantidade e tipo dos atributos adicionados, pois atributos novos exigem a criação de diversos métodos, incluindo os métodos para inseri-los nas classes de interface gráfica com o usuário.

Foram tomadas outras medidas do tempo necessário para preenchimento dos formulários em tela, pelos alunos participantes do E-GRN-GREN. Eles levaram entre trinta e quarenta e cinco minutos para preencher os formulários do GREN-Wizard para instanciá-lo para o sistema de clínica veterinária (em média foram trinta minutos). Esses participantes tiveram treinamento de cerca de meia hora a respeito do GREN-Wizard e depois receberam o manual a ser estudado antes de seu uso.

As medidas de tempo para uso do GREN-Wizard indicam que é possível obter aplicações no domínio de gestão de recursos de negócios em poucos minutos. Deve-se lembrar que nem sempre se consegue modelar toda a funcionalidade do sistema com a GRN e, conseqüentemente, o código gerado deverá ser complementado para atender aos requisitos não cobertos. Ainda assim, aconselha-se a utilização do GREN-Wizard para obtenção da parte do código referente aos requisitos atendidos pela GRN, para depois iniciar-se a implementação dos demais requisitos.

Quanto às limitações do GREN-Wizard, pode-se citar algumas sugestões feitas por pessoas que o utilizaram ou que assistiram a uma demonstração durante a sessão de ferramentas do Simpósio Brasileiro de Engenharia de Software de 2002. A navegação entre os padrões é uma das deficiências do GREN-Wizard, pois não existe um botão para voltar ao padrão aplicado anteriormente. Na implementação atual, caso seja necessário voltar ao padrão anterior, deve-se salvar a especificação corrente, retornar ao primeiro padrão aplicado e então percorrer novamente todos os padrões até chegar ao padrão desejado. Esse requisito poderá ser implementado na próxima versão da ferramenta. Outro ponto a ser melhorado é com relação a participantes opcionais dos padrões. Em alguns casos foram criados variantes do padrão para permitir que um participante fosse opcional, por exemplo, no padrão 4-LOCAR O RECURSO, existe um variante "Sem Origem", no qual o participante "Origem" não é utilizado. Em outros casos simplesmente é permitido que o participante seja deixado em branco durante o preenchimento dos formulários em tela. Na próxima versão da ferramenta pretende-se unificar o tratamento desse caso, dando preferência à segunda alternativa, que é de permitir que o participante fique em branco. Haverá uma indicação na tela de que o participante é opcional, por exemplo, o nome estará em itálico.

Outro tipo de limitação existente no GREN-Wizard é quanto à possibilidade de modificação dos padrões da GRN sem perder as informações presentes na base de dados. Por exemplo, se a definição dos participantes de um padrão for modificada, a base de dados que contém a especificação das aplicações já modeladas com a GRN teria que ser revisada para manter a compatibilidade com a nova versão. Isso seria desejável para garantir o propósito de reutilizar a especificação de aplicações já desenvolvidas quando forem criadas aplicações similares. Para resolver esse problema, será necessário avaliar todas as possibilidades de mudança na definição dos padrões e estudar as conseqüências dessas mudanças nas aplicações modeladas usando a GRN.

6.11 Considerações Finais

Deve-se ressaltar que a avaliação realizada para os diversos processos e produtos está atrelada aos produtos específicos: GRN, GREN e GREN-Wizard. Assim, não se pode generalizar os resultados para outras linguagens de padrões, frameworks e *Wizards*. Por exemplo, nada garante que outras linguagens de padrões de análise, escritas por autores diferentes, com nível distinto de detalhes e usando outro estilo para documentar os padrões, sejam ou não úteis na modelagem de sistemas. Da mesma forma, outros frameworks construídos com base em linguagens de padrões podem estar documentados de forma inadequada quanto ao propósito de permitirem a instanciação tal qual proposto nesta tese.

Em suma, os resultados confirmam o forte relacionamento existente entre linguagens de padrões e frameworks sugerido em diversos trabalhos existentes (Beck e Johnson, 1994; Brugali e Menga, 1999; Johnson, 1992) (ver seção 2.5). Pode-se tirar proveito desse relacionamento, por exemplo utilizando o processo geral proposto nesta tese, para acelerar o processo de desenvolvimento de sistemas num particular domínio.

Os experimentos relatados nas seções 6.3.2, 6.3.3, 6.3.4 e 6.5.2, juntamente com a experiência adquirida na sua realização, podem servir de base para o planejamento de novos experimentos que mostrem com significância o valor dos processos e produtos apresentados nesta tese.

A análise efetuada neste Capítulo forneceu indícios da utilidade dos processos propostos para os vários profissionais da Engenharia de Software, conforme almejado nos objetivos desta tese (ver seção 1.3). O desenvolvedor de frameworks pode se beneficiar dos processos de construção de frameworks e de *Wizards*, conforme relatado nas seções 6.2, 6.4 e 6.6. O desenvolvedor de aplicações experiente pode usar os processos para uso de uma linguagem de padrões, seu framework associado e *wizard* (se houver), para agilizar o desenvolvimento de software, conforme discutido nas seções 6.3, 6.5 e 6.7. O desenvolvedor de aplicações inexperiente pode usar a linguagem de padrões para modelar seu sistema e *wizard* para produzir o código final (também conforme discutido nas seções 6.3, 6.5 e 6.7). Além disso, outros profissionais que podem usar as tecnologias propostas são: o analista de domínio, que pode usar o processo de construção de linguagens de padrões, conforme discutido na seção 6.2; o testador de software, que pode utilizar o *wizard* para acelerar o teste do framework, conforme mencionado na seção 6.8; e o construtor de aplicações para o domínio de gestão de recursos de negócios, que pode utilizar a GRN, o GREN e o GREN-Wizard para modelar e implementar seus sistemas nesse domínio.

No próximo Capítulo resume-se o trabalho efetuado, listando suas principais contribuições, limitações e possíveis trabalhos futuros.

CAPÍTULO

7

Conclusões

7.1 Considerações Iniciais

Este Capítulo resume o trabalho efetuado e lista suas contribuições para a área de Engenharia de Software e Sistemas de Informação, bem como suas limitações. Aponta também os trabalhos futuros que podem ser realizados em continuidade a este trabalho, que poderão gerar outras contribuições.

7.2 Resumo do Trabalho Efetuado

Este trabalho consistiu em duas fases: a primeira de desenvolvimento de produtos concretos e a segunda de generalização dos processos utilizados para obtenção desses produtos (Braga e Masiero, 2001b, 2002b). Na primeira fase foram realizadas três atividades principais: o desenvolvimento de uma linguagem de padrões para o domínio de Gestão de Recursos de Negócios(Braga et al., 1999); a construção de um framework para Gestão de Recursos de Negócios, baseado na linguagem de padrões; e a construção de um *Wizard* para automação da instanciação do framework para aplicações particulares (Braga e Masiero, 2002c).

Na segunda fase, com base na experiência adquirida na fase anterior, foram definidos quatro processos diferentes: o primeiro para construção de linguagens de padrões para um domínio específico; o segundo para a construção de frameworks com base em uma linguagem de padrões

(Braga e Masiero, 2001a, 2002d); o terceiro para construção de *Wizards* para automação da instanciação de frameworks com base em uma linguagem de padrões; e o quarto para instanciação de frameworks usando uma linguagem de padrões (Braga e Masiero, 2002e), sendo dividido em dois sub-processos: o primeiro quando a instanciação é feita de forma manual e o segundo quando o *Wizard* é utilizado para automatizar a instanciação.

Diversos experimentos e usos dos diferentes processos foram realizados, para averiguar a viabilidade de uso desses processos por outros engenheiros de software. Os resultados foram positivos e confirmaram as vantagens em se usar uma linguagem de padrões na construção e instanciação de um framework para o mesmo domínio. Muitas lições foram aprendidas nessa fase, tendo sido documentadas visando à sua utilização em futuros desenvolvimentos que sigam os processos aqui propostos.

7.3 Contribuições

A contribuição principal deste trabalho é a definição de um processo geral para facilitar tanto a construção quanto a utilização de frameworks, que constituem uma técnica poderosa para alavancar o reuso de software. Tal processo foi definido em passos detalhados e possíveis de serem seguidos por engenheiros de software que confrontam-se com o mesmo problema: desenvolver um software reutilizável (framework) que possa ser adaptado a famílias de aplicações dentro de um domínio específico. A principal característica do processo proposto é utilizar uma linguagem de padrões para guiar todo o processo de desenvolvimento e uso do framework. Para obter um sistema específico, o usuário do framework — o engenheiro de software que desenvolve aplicações concretas a partir do framework — precisa basicamente conhecer e saber aplicar a linguagem de padrões.

O processo geral é composto de quatro sub-processos detalhados ao longo da tese. O primeiro sub-processo foi definido para auxiliar desenvolvedores a criar uma linguagem de padrões com base em sua experiência num dado domínio, ou obtendo as informações necessárias por meio de análise de domínio, ou pela engenharia reversa de aplicações em tal domínio. Esse sub-processo tem como resultado uma linguagem de padrões, que pode ser utilizada na modelagem de sistemas para o domínio em questão, cujo processo de utilização também é descrito nesta tese.

O segundo sub-processo mostra como, a partir da linguagem de padrões obtida como resultado do primeiro processo, construir um framework de software orientado a objetos que dê suporte à linguagem de padrões e permita a implementação de aplicações no mesmo domínio dessa linguagem. Esse sub-processo tem como resultado um framework composto de classes concretas e abstratas e um manual do usuário (*cookbook*) para guiar a instanciação do framework para atender aos requisitos de uma aplicação específica no domínio em questão.

O terceiro sub-processo define as características que uma ferramenta de instanciação automática do framework (denominada *Wizard*) deve ter, para permitir que o usuário do framework, tendo como conhecimento básico apenas a linguagem de padrões, possa obter suas aplicações específicas. A estrutura interna proposta para o *Wizard* permite que ele funcione para quaisquer pares "linguagem de padrões e framework caixa branca", pois todas as informações sobre a linguagem de padrões, sobre o framework e sobre o mapeamento dos padrões para as classes do framework ficam armazenadas em uma base de dados e são interpretadas em tempo de execução pelo *Wizard*. Esse sub-processo tem como resultado um *Wizard* cuja interface gráfica com o usuário pode ser utilizada por conhecedores da linguagem de padrões associada ao framework.

O quarto sub-processo mostra como instanciar o framework a partir dos requisitos de uma aplicação específica, utilizando a linguagem de padrões como apoio. São duas alternativas diferentes, dependendo da existência ou não do *Wizard* para automatizar parte do processo. Esse sub-processo tem como resultado a aplicação específica, na forma de classes especializadas a partir de classes abstratas do framework, juntamente com os métodos sobrepostos ou criados durante a instanciação. Dependendo do caso, pode-se ter como resultado, também, um *script* para criação da base de dados a fim de persistir os objetos da nova aplicação.

Contribuições adicionais deste trabalho, valiosas na área de Sistemas de Informação, são os produtos concretos obtidos durante a definição do processo geral proposto. A Linguagem de Padrões para Gestão de Recursos de Negócios (GRN) é um desses produtos. Ela tem sido utilizada com sucesso durante os três últimos anos por diversas categorias de usuários, como alunos de graduação/pós-graduação e profissionais da área de informática, que a aplicaram na modelagem de sistemas do domínio de Gestão de Recursos de Negócios. O framework GREN é o segundo produto obtido. Ele foi utilizado em diversos estudos de caso, conforme descrito no Capítulo 6, e ainda poderá ser objeto de diversos trabalhos futuros. O GREN-Wizard é o terceiro produto obtido. Ele tem sido utilizado para gerar aplicações do domínio e para testar o framework GREN. Assim como o GREN, o GREN-Wizard também poderá ser objeto de trabalhos futuros.

7.4 Limitações do Trabalho Efetuado

Algumas limitações dos produtos obtidos nesta tese decorrem, principalmente, do fato da linguagem de padrões GRN pertencer a um domínio restrito de Sistemas de Informação. Isso faz com que a abordagem aqui proposta necessite sofrer extensões para poder ser aplicada a outros domínios da computação, que possuem problemas não considerados por serem inexistentes no domínio tratado.

A implementação do framework GREN foi feita utilizando-se o modelo Cliente-Servidor e, portanto, aplicações dele derivadas não podem ser executadas em ambiente Web. Sugere-se como trabalho futuro, a extensão do GREN para funcionar em plataforma Web. Além disso, algumas

das classes de interface com o usuário fornecidas pelo GREN, são adequadas apenas para sistemas de pequeno porte, pois o desempenho fica prejudicado para aplicações com muitos objetos. Isso deve-se ao fato de que algumas telas de entrada de dados mostram uma lista de todos os objetos já registrados, o que, no caso de haver muitos objetos, pode levar tempo demais para carregá-los da base de dados relacional para os objetos em memória. Isso pode ser resolvido em trabalhos futuros implementando-se novas classes na camada de interface gráfica com o usuário para casos em que há muitos objetos armazenados na base de dados.

Como a finalidade do desenvolvimento do GREN foi a definição do processo de construção de um framework com base em uma linguagem de padrões, e não se pretendia utilizá-lo efetivamente em aplicações concretas, não houve preocupação em implementar alguns aspectos que são importantes em sistemas do domínio tratado, como por exemplo o controle de acesso para diversos tipos de usuário do sistema, o suporte a diferentes tipos de bases de dados e a distribuição. Essas características também podem ser alvo de trabalhos futuros, inclusive já havendo sido iniciado um trabalho para atender parte desses requisitos.

No GREN-Wizard, alterações feitas na especificação do domínio, por exemplo, na estrutura dos padrões ou no mapeamento para o framework, podem acarretar inconsistência na base de dados com a especificação das aplicações já modeladas. Assim, ao modificar a especificação do domínio, deve-se ter o cuidado de rastrear toda a base de dados de aplicações existentes e efetuar as devidas adaptações para que elas possam ser reusadas futuramente no desenvolvimento de aplicações similares.

Outro ponto não coberto pelo processo proposto é o uso de mais de uma linguagem de padrões para desenvolver um sistema. Por exemplo, uma delas poderia conter padrões de análise e a outra padrões de projeto, mais preocupados em solucionar problemas inerentes ao projeto físico do sistema, tais como persistência dos dados, segurança, etc. Nesse caso, a flexibilidade fornecida pelo framework quanto a esses aspectos também poderia fazer parte da linguagem de padrões e isso guiaria as decisões de projeto.

7.5 Sugestões de trabalhos futuros

Além dos trabalhos sugeridos para vencer as limitações discutidas na seção anterior, esta tese poderá originar diversos outros trabalhos de pesquisa. Trabalhos com pequeno grau de dificuldade são: implementação de uma ferramenta que receba como entrada os padrões e variantes utilizados na instanciação de um sistema específico e produza um diagrama de classes (em UML, por exemplo); implementação de uma ferramenta para armazenar a especificação da linguagem de padrões em base de dados, facilitando o uso do *Wizard* por outras linguagens de padrões e frameworks; e inclusão, na GRN, de modelos que representem o comportamento dinâmico dos padrões, por

exemplo por meio de diagramas de seqüência da UML, já que a versão atual da GRN mostra apenas o comportamento estático dos padrões.

Trabalhos mais elaborados, com potencial para mestrado ou doutorado são:

- a implementação do GREN usando Java, que é uma linguagem orientada a objetos altamente utilizada atualmente;
- extensão da linguagem de padrões GRN e do framework GREN para outros aspectos de gestão de recursos de negócios, por exemplo, contabilidade, finanças, promoções, controle de entregas, etc.;
- extensão do GREN para gestão de recursos na WEB, com base no framework de leilões virtuais Qd+;
- generalização da ferramenta GREN-Wizard para que possa ser utilizada por quaisquer pares "framework e linguagem de padrões", seguida de sua adaptação para o framework Qd+ e a linguagem de padrões LV;
- formalização do relacionamento entre a linguagem de padrões e o framework;
- implementação de um Ambiente de Desenvolvimento de Frameworks e Instanciadores de Frameworks, com base em linguagens de padrões. Dada a linguagem de padrões, o ambiente ajudaria a armazenar sua definição (usando a ferramenta proposta acima), a derivação dos *hot spots* seria monitorada e registrada pela ferramenta e formas de implementação dos *hot spots* seriam sugeridas. Após escolhida a forma de implementação, o desenvolvedor do framework usaria o ambiente para armazenar o mapeamento da linguagem de padrões para o framework. Assim o *Wizard* genérico sugerido nesta tese poderia ser automaticamente usado para outras linguagens de padrões e frameworks; e
- elaboração de uma estratégia para teste de frameworks, na qual a linguagem de padrões poderia oferecer indicações dos casos de teste que devem ser gerados.

7.6 Trabalhos decorrentes desta pesquisa

Um trabalho de mestrado foi concluído (Ré e Masiero, 2002), já mencionado em capítulos anteriores, no qual foi desenvolvida uma linguagem de padrões para gestão de leilões virtuais e o framework associado. Esse trabalho possibilitou o uso de vários dos processos propostos nesta tese, permitindo avaliá-los e melhorá-los.

Foi também concluído um trabalho de mestrado (Coelho, 2002), no qual a arquitetura do GREN foi re-projetada usando componentes de software, visando a comparação da complexidade do framework resultante em relação à sua versão atual.

Encontra-se em andamento um trabalho de doutorado para definição de um processo para reengenharia com base em frameworks, no qual o GREN e a GRN estão sendo utilizados como exemplos para o estudo de caso. Um dos resultados esperados para esse trabalho é que o framework possa servir não somente para engenharia avante, mas também para reengenharia de sistemas legados. Após a engenharia reversa do sistema legado, conduzida com base na linguagem de padrões, o framework é utilizado para efetuar a engenharia avante do sistema.

Iniciou-se um trabalho de mestrado para implementação de aspectos não-funcionais que não foram previstos no GREN, como por exemplo controle de acesso e segurança, usando a tecnologia de aspectos. Nesse trabalho será mantida a linguagem Smalltalk e serão estudadas alternativas para implementar aspectos em Smalltalk, como por exemplo o Apostle[1], o AspectS[2] e o MetaClassTalk[3].

Foi iniciado um trabalho de doutorado no qual estão sendo estudadas novas formas de componentizar o framework GREN, explorando os aspectos funcionais do domínio e propondo novas maneiras de estruturá-los, por exemplo, por meio de aspectos (Elrad et al., 2001; Kiczales, 1996) e agentes de software. Durante esse trabalho poderá ser necessário realizar a migração do GREN para outra linguagem de programação, por exemplo Java, para facilitar o uso de tecnologias ligadas a aspectos.

[1] `http://www.cs.ubc.ca/labs/spl/projects/apostle/`
[2] `http://www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/`
[3] `http://csl.ensm-douai.fr/MetaclassTalk`

Referências

AARSTEN, A.; BRUGALI, D.; MENGA, G. *A CIM framework and pattern language* in M. Fayad, R. Johnson. Domain-Specific Application Frameworks: Frameworks Experience by Industry, John Willey and Sons, p. 21–42, 2000.

ALBRECHT, A. J. *AD/M productivity measurement and estimate validation*. Purchase-NY, USA: IBM Corporate Information Systems, 1984.

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

ALEXANDER, C.; ET AL. *A pattern language*. New York: Oxford University Press, 1977.

APPLETON, B. Patterns and software: Essential concepts and terminology. Disponível na URL: `http://www.enteract.com/~bradappdocpatterns-intro.html`, 1997.

BECK, K.; CUNNINGHAM, W. Using pattern languages for object-oriented programs. Relatório Técnico n. CR-87-43, disponível em 18/11/2002 na URL: `http://c2.com/doc/oopsla87.html`, 1987.

BECK, K.; JOHNSON, R. Patterns generate architectures. In: *European Conference on Object-Oriented Programming*, Bologna, Italy, 1994, p. 139–149.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *The unified modeling language user guide*. Addison-Wesley, 1998.

BOSCH, J.; MOLIN, P.; MATTSSON, M.; BENGTSSON, P.; FAYAD, M. *Framework problem and experiences* in M. Fayad, R. Johnson, D. Schmidt. Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Willey and Sons, p. 55–82, 1999.

BOYD, L. *Business patterns of association objects* in Martin, R.C.; Riehle, D.; Buschmann, F. Pattern Languages of Program Design 3, Addison-Wesley, p. 395–408, 1998.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. A confederation of patterns for resource management. In: *5th Pattern Languages of Programs Conference (PLoP'98)*, Monticello – IL, USA, 1998.

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

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. GRN: A pattern language for business resource management. Submetida ao Journal of the Brazilian Computer Society em agosto de 2002, disponível na URL: `http://www.icmc.usp.br/~rtvb/ext_pat_lang.zip`, 2002.

BRAGA, R. T. V.; MASIERO, P. C. Identification of framework hot spots using pattern languages. In: *15th Brazilian Symposium on Software Engineering (SBES 2001)*, Rio de Janeiro-Brasil, 2001a, p. 145–160.

BRAGA, R. T. V.; MASIERO, P. C. A pattern language-based approach for framework construction and instantiation. In: *1st ASERC Workshop on Software Architectures*, Computing Science Centre University of Alberta, Canadá, 2001b, p. Position paper, 2p.

BRAGA, R. T. V.; MASIERO, P. C. Building a wizard for framework instantiation based on a pattern language. Artigo submetido à Conferência "European Conference for Object-Oriented Programming" (ECOOP 2003), 2002a.

BRAGA, R. T. V.; MASIERO, P. C. Frameworks construction and instantiation using pattern languages. In: *International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications*, ACIS, Foz do Iguazu-Brazil, 2002b, p. 305–310.

BRAGA, R. T. V.; MASIERO, P. C. GREN-Wizard: a tool to instantiate the GREN framework. In: *Caderno de Ferramentas do 16o Simpósio Brasileiro de Engenharia de Software (SBES 2002)*, Gramado-RS, 2002c, p. 408–413.

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

BRAGA, R. T. V.; MASIERO, P. C. The role of pattern languages in the instantiation of object-oriented frameworks. *Lecture Notes on Computer Science*, v. 2426-Advances in Object-Oriented Information Systems, p. 122–131, 2002e.

BROWN, K.; WHITENACK, B. G. *Crossing chasms: A pattern language for Object-RDBMS integration, the static patterns* in Vlissides et al., 1996 Pattern Languages of Program Design 2, Addison-Wesley, p. 227–238, 1996.

BRUGALI, D. *From objects to agents: Software reuse for distributed systems*. Tese de Doutoramento, Department of Informatica and Automatica of the Politecnico of Turine, Itália, 1998.

BRUGALI, D.; MENGA, G. Frameworks and pattern languages: an intriguing relationship. *ACM Computing Surveys*, v. 32, n. 1, p. 2–7, 1999.

BRUGALI, D.; MENGA, G.; AARSTEN, A. The framework life span. *Communications of the ACM*, v. 40, n. 10, p. 65–68, 1997.

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

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

BUTLER, G.; KELLER, R.; MILI, H. A framework for framework documentation. *ACM Computing Surveys*, v. 32, n. 1, 2000.

CAREY, J.; CARLSON, B.; GRASER, T. *SanFrancisco design patterns: Blueprints for business software*. Addison-Wesley, 2000.

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

COAD, P. Object-oriented patterns. *Communications of the ACM*, v. 35, n. 9, p. 152–159, 1992.

COAD, P.; NORTH, D.; MAYFIELD, M. *Object models: Strategies, patterns and applications*. 2 ed. Yourdon Press, 1997.

COELHO, F. M. *Uso de componentes de software no desenvolvimento de frameworks orientados a objetos*. Dissertação de Mestrado, IC-UNICAMP, Campinas – SP, 2002.

COLEMAN, D.; ET AL. *Object oriented development – the Fusion method*. Prentice Hall, 1994.

CONOVER, W. J. *Practical nonparametric statistics – 2a. ed.* Wiley, New York, 1980.

COPLIEN, J. *Advanced C++ programming styles and idioms*. Reading-MA: Addison-Wesley, 1992.

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

CUNNINGHAM, W. *The CHECKS pattern language of information integrity* in J. Coplien and D. Schmidt (eds.) - Pattern Languages of Program Design, Addison-Wesley, p. 145–155, 1995.

CZARNECKI, K.; EISENECKER, U. *Generative programming: Methods, tools, and applications*. Addison-Wesley, 2000.

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

ERIKSSON, H. *UML toolkit*. Wiley Computer Publishing, 1998.

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

FAYAD, M. E.; SCHMIDT, D. C. Object-oriented application frameworks. *Communications of the ACM*, v. 40, n. 10, 1997.

FERNANDEZ, E. Stock manager: An analysis pattern for inventories. In: *7th Pattern Languages of Programs Conference (PLoP'00)*, Monticello – IL, USA, 2000.

FERNANDEZ, E.; YUAN, X. An analysis pattern for reservation and use of reusable entities. In: *6th Pattern Languages of Programs Conference (PLoP'99)*, Monticello – IL, USA, 1999.

FERNANDEZ, E.; YUAN, X.; BREY, S. Analysis patterns for the order and shipment of a product. In: *7th Pattern Languages of Programs Conference (PLoP'00)*, Monticello – IL, USA, 2000.

FONTOURA, M.; PREE, W.; RUMPE, B. *The UML profile for framework architectures*. Addison-Wesley, 2001.

FOWLER, M. *Analysis patterns*. Addison-Wesley, 1997.

FROEHLICH, G.; HOOVER, J.; LIU, L.; SORENSON, P. Hooking into object-oriented application frameworks. In: *International Conference on Software Engineering*, Boston-USA, 1997, p. 491–501.

GALL, H. C.; KLÖSH, R. R.; MITTERMEIR, R. T. Application patterns in re-engineering: Identifying and using reusable concepts. In: *6th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 1996, p. 1099–1106.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design patterns - abstraction and reuse of object-oriented design. *Lecture Notes on Computer Science*, , n. 707, p. 406–431, 1993.

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

GANGOPADHYAY, D.; MITRA, S. Understanding frameworks by exploration of exemplars. In: *International Workshop on C.A.S.E*, IEEE, 1995, p. 90–99.

GIMENES, I. M. S.; TRAVASSOS, G. H. O enfoque de linha de produto para desenvolvimento de software. In: *Anais do XXII Congresso da Sociedade Brasileira de Computação - Tutoriais*, 2002.

GOMAA, H. Reusable software requirements and architectures for families of systems. *Journal of Systems and Software*, v. 28, n. 3, p. 189–202, 1996.

HAY, D. *Data model patterns – conventions of thought*. New York - NY, USA: Dorset House Publishing, 1996.

HOLLANDER, M.; WOLFE, D. A. *Nonparametric statistical methods*. New York, USA: John Wiley, 1973.

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

JOHNSON, R. E. Documenting frameworks using patterns. In: *OOPSLA '92*, 1992, p. 63–76.

JOHNSON, R. E. *Transactions and accounts* in Vlissides et al., 1996 Pattern Languages of Program Design 2, Addison-Wesley, p. 239–249, 1996.

JOHNSON, R. E. Components, frameworks, and patterns. In: *Proceedings of the 1997 Symposium on Symposium on Software Reusability (SST'97)*, 1997a.

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

JOHNSON, R. E.; RUSSO, V. *Reusing object-oriented designs*. Rel. Téc. UIUCDCS 91-1696, University of Illinois, 1991.

JOHNSON, R. E.; WOOLF, B. *Type object* in Martin, R.C.; Riehle, D.; Buschmann, F. Pattern Languages of Program Design 3, Addison-Wesley, p. 47–65, 1998.

JONES, C. *Preliminary table of languages and levels*. Burlington, Mass., USA: Software Productivity Research Inc., 1989.

JONES, C. *Applied software measurement*. New York, USA: McGraw Hill, 1991.

KEEFER, P. D. An object oriented framework for accounting systems. Dissertação (Mestrado), Universidade de Illinois at Urbana-Champaign, EUA, 86 p. (em inglês), 1994.

KERTH, N. L.; CUNNINGHAM, W. Using patterns to improve our architectural vision. *IEEE Software*, p. 53–59, 1997.

KICZALES, G. Aspect-oriented programming. *ACM Computing Surveys*, v. 28, n. 4es, p. 154, 1996.

KRASNER, E. K.; POPE, S. T. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object Oriented Programming*, p. 26–49, 1988.

LARMAN, C. *Applying UML and patterns: An introduction to object-oriented analisys and design*. Prentice-Hall, 1997.

MESZAROS, G.; DOBLE, J. *A pattern language for pattern writing*, cáp. 29 in J. Coplien; D. Schmidt. Pattern Languages of Program Design, Reading-MA, Addison-Wesley, p. 529–574, 1998.

MONDAY, P.; CAREY, J.; DANGLER, M. *SanFrancisco 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.

OMNISPHERE, I. S. C. Omni-Sphere – open application generator. Página com mais informações disponível em 26 de Novembro de 2002 na URL: `http://www.omni-sphere.com/overview/overview.htm`, 2002.

ORTIGOSA, A.; CAMPO, M. Towards agent-oriented assistance for framework instantiation. *ACM SIGPLAN Notices*, v. 35, n. 10, p. 253–263, 2000.

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

PREE, W. *Hot-spot-driven development* in M. Fayad, R. Johnson, D. Schmidt. Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Willey and Sons, p. 379–393, 1999.

PREE, W.; POMBERGER, G.; SCHAPPERT, A.; SOMMERLAD, P. Active guidance of framework development. *Software - Concepts and Tools*, v. 16, n. 3, p. 136–, 1995.

PRESSMAN, R. S. *Engenharia de software, 5a. edição*. McGraw Hill, 843 p., 2002.

PRIETO-DIAZ, R.; ARANGO, G. *Domain analysis and software system modeling*. IEEE Computer Science Press Tutorial, 1991.

RÉ, R.; BRAGA, R. T. V.; MASIERO, P. C. A pattern language for online auctions. In: *8th Pattern Languages of Programs Conference (PLoP'2001)*, Monticello – IL, USA, 2001.

RÉ, R.; BRAGA, R. T. V.; MASIERO, P. C. A process for framework development from reverse engineering of Web- based information systems: Application to the online auction domain. In: *First Seminar on Advanced Research in Electronic Business (EBR 2002)*, Rio de Janeiro, 2002.

RÉ, R.; MASIERO, P. C. *Um processo para construção de frameworks a partir da engenharia reversa de sistemas de informação baseados na Web: Aplicação ao domínio dos leilões virtuais*. Dissertação de Mestrado, ICMC/USP, São Carlos – SP, 2002.

RATIONAL, C. Unified Modeling Language. Disponível na URL: `http://www.rational.com/uml/references`, 2000.

ROBERTS, D.; JOHNSON, R. *Evolving frameworks: A pattern language for developing object-oriented frameworks* in Martin, R.C., Riehle, D. , Buschmann, F. Pattern Languages of Program Design 3, Addison-Wesley, p. 471–486, 1998.

ROSS, D. Structured analysis: A language for communicating ideas. *IEEE Trans. Soft. Eng.*, v. 3, n. 1, 1977.

SCHMID, H. A. Systematic framework design by generalization. *Communications of the ACM*, v. 40, n. 10, p. 48–51, 1997.

SCHMID, H. A. *Framework design by systematic generalization* in M. Fayad, R. Johnson, D. Schmidt. Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Willey and Sons, p. 353–378, 1999.

SCHMID, H. A.; FAYAD, M. E.; (GUEST EDITORS), R. E. J. Software patterns. *Communications of the ACM*, v. 39, n. 10, p. 36–39, 1996.

SHULL, F.; LANUBILE, F.; BASILI, V. Investigating reading techniques for object-oriented framework learning. *IEEE Transactions on Software Engineering*, v. 26, n. 11, p. 1101–1118, 2000.

TALIGENT, I. Leveraging object-oriented frameworks. White-paper, Disponível na URL: `http://www.ibm.com/java/education/oobuilding/index.html`, 1993.

VLISSIDES, J. *Pattern hatching – design patterns applied*. Addison-Wesley, 1998.

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

WERNER, C. M. L.; BRAGA, R. M. M. Desenvolvimento baseado em componentes. In: *Anais do XIV Simpósio Brasileiro de Engenharia de Software - Minicurso*, João Pessoa, PB, 2000, p. 297–329.

WHOLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.; REGNELL, B.; WESSLÉN, A. *Experimentation in software engineering - an introduction*. Kluwer Academic Publishers, 2000.

YASSIN, A.; FAYAD, M. E. *Application frameworks: A survey*, cáp. 29 in M. E. Fayad and R. E. Johnson. Domain-Specific Application Frameworks: Frameworks Experience by Industry, John Wiley & Sons, p. 615–632, 2000.

YODER, J. W.; BALAGUER, F.; JOHNSON, R. E. Architecture and design of adaptive object models. In: *Intriguing Technology Presentation, Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '01)*, ACM SIGPLAN Notices, disponível em 8/11/02 na WWW na URL: `http://www.joeyoder.com/papers/`, 2001.

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.

Apêndices

Apêndice A - Requisitos para o Sistema de Hotel

Sistema para Hotel

Documento de Requisitos

Outubro de 2001

A – VISÃO GERAL DO SISTEMA

O sistema para o Hotel Uirapuru consiste basicamente do gerenciamento das estadias de *hóspedes* do hotel, controlando desde a reserva de *acomodações* até o acompanhamento do período de estadia, considerando os diversos tipos de consumo efetuados pelos hóspedes, tais como frigobar, restaurante, lavanderia e telefonemas. O hotel possui 70 apartamentos simples, 30 apartamentos para casal e 10 suítes de luxo, distribuídos pelos diversos andares do hotel. O sistema deve ainda emitir diversos tipos de relatórios e consultas, possibilitando um melhor gerenciamento das *acomodações* oferecidas.

B – REQUISITOS FUNCIONAIS

B1 – Lançamentos diversos

1. O sistema deve permitir a inclusão, alteração e remoção de *hóspedes do hotel*, contendo os seguintes atributos: nome, endereço, cidade onde mora, estado, país, telefone, email, documento de identificação (RG ou CPF para brasileiros e passaporte para estrangeiros), data de nascimento e nome dos pais.

2. O sistema deve permitir a inclusão, alteração e remoção de *itens de consumo*, classificados por diversas categorias (frigobar, restaurante e lavanderia), com os seguintes atributos: código do item, descrição e preço de venda.

3. O sistema deve permitir a inclusão, alteração e remoção de *funcionários* do hotel, com os seguintes atributos: nome, endereço, cidade, estado, telefone e data de nascimento.

4. O sistema deve permitir a inclusão, alteração e remoção dos *tipos de acomodação* oferecidos pelo hotel, com os seguintes atributos: código do tipo de acomodação, descrição do tipo de acomodação, quantidade de unidades total desse tipo de acomodação existente no hotel, preço da *diária*, número de pessoas adultas e número de crianças que esse tipo de acomodação comporta.

5. O sistema deve permitir a inclusão, alteração e remoção das *acomodações* existentes no hotel, com os seguintes atributos: número da acomodação, andar no qual se encontra e código do tipo de acomodação. Para cada tipo de acomodação, podem existir diversas acomodações com números diferentes e localizadas em diversos andares do hotel.

6. O sistema deve permitir a *reserva de acomodação*. Cada reserva possui os seguintes atributos: data e hora de chegada do hóspede, data e hora de saída do hóspede, identificação do hóspede principal (previamente cadastrado), tipo de acomodação desejada, nomes e idades dos *acompanhantes*, valor da *diária*, taxa de multa a ser cobrada em caso de desistência de última hora (a menos de 12 horas do início previsto de entrada), os dados do cartão de crédito do hóspede e desconto concedido (opcional). A reserva somente deve ser concretizada se houver vagas suficientes para atendê-la. Caso contrário deverá ser mostrada uma mensagem alertando que não há disponibilidade de acomodações para o período indicado. A remoção de reserva somente é permitida sem maiores encargos até 12 horas antes do início previsto para *estadia no hotel*. Após esse período, a remoção da reserva deve alertar o *funcionário* do hotel de que deve ser cobrada

a taxa de multa estabelecida durante a reserva.

7. O sistema deve permitir o processamento da *entrada de hóspede* no hotel, com os seguintes atributos: data e hora de chegada do hóspede, data e hora prevista para saída do hóspede, identificação do hóspede principal (previamente cadastrado), número da acomodação utilizada, nomes e idades dos *acompanhantes*, valor da *diária*, funcionário responsável pelo recebimento do hóspede e desconto concedido (opcional). Se tiver sido feita a reserva prévia da acomodação, então, durante a entrada do hóspede, informa-se seu nome e o sistema recupera automaticamente os dados da reserva, que podem ser alterados, se necessário.

8. O sistema deve permitir a inclusão, alteração e remoção de *consumo do hóspede*. Durante a estadia no hotel, diariamente um funcionário do hotel faz a coleta de informações no frigobar para informar ao sistema os itens consumidos. Além disso, pedidos feitos ao restaurante e serviços requisitados à lavanderia são instantaneamente informados ao sistema pelo funcionário responsável, que deve ter uma senha especial de acesso ao sistema. Cada consumo do hóspede possui os seguintes atributos: data do consumo, nome do funcionário responsável, número da acomodação, código dos itens consumidos, quantidades consumidas e valor unitário.

9. O sistema deve permitir o processamento da *saída de hóspede* do hotel, com os seguintes atributos: número da acomodação utilizada, data e hora de saída do hóspede, número de *diárias* cobradas, valor de cada diária, valor dos gastos com telefonemas, e desconto concedido (opcional). O sistema deve automaticamente totalizar os gastos de consumo do hóspede, que foram previamente cadastrados, mostrando os subtotais por categoria (frigobar, restaurante e lavanderia). O sistema deve também apresentar na tela o total a pagar, que é a soma das diárias, acrescentando-se os consumos e os telefonemas e subtraindo-se o desconto, se houver. Também é desejável que o sistema permita ao hóspede dar entrada ao seu processo de *saída do hotel* a partir da televisão de seu apartamento.

10. O sistema deve permitir as seguintes opções de *pagamento da estadia* no hotel: 1) à vista (em dinheiro, cheque ou cartão de crédito); 2) faturado em 30 dias.

11. O sistema deve permitir a quitação de uma fatura paga pelo hóspede, contendo as seguintes informações: número da fatura, data de vencimento, data de pagamento, valor total pago, juros e multa.

B2 – Impressão de diversos tipos de relatórios e consultas

12. O sistema deve permitir a impressão de uma listagem dos hóspedes que estão no hotel no momento, contendo o nome do hóspede principal, nome dos acompanhantes, data de entrada, data prevista para saída e número da acomodação.

13. O sistema deve permitir a impressão de uma listagem das reservas efetuadas para a data atual, contendo o nome do hóspede principal, telefone para contato, tipo de acomodação e data prevista para saída.

14. O sistema deve permitir a impressão de um comprovante de *saída do hóspede*, contendo o nome do hóspede, documento, datas e horários de entrada e saída, número total de diárias, valor total das diárias, valor total do consumo do hóspede, valor total dos telefonemas, valor do desconto e total a pagar. Nesse mesmo comprovante deve ser mostrada uma lista com os produtos consumidos, contendo a data do consumo, descrição do item de consumo, quantidade consumida, preço unitário e preço total. Ainda nesse comprovante deve constar a forma de pagamento e deve ser reservado um espaço para assinatura do hóspede.

15. O sistema deve permitir ao hóspede imprimir um histórico de suas *estadias no hotel*. Para tal o hóspede deve ter sido previamente cadastrado e deve portar um código de identificação e uma senha. Esse histórico deve conter uma linha para cada estadia do hóspede, contendo as datas de entrada e saída e os totais pagos em cada ocasião.

16. O sistema deve permitir a consulta on-line da ocupação das *acomodações* num certo período. Uma acomodação está ocupada se existem hóspedes utilizando-a no momento. Uma acomodação está disponível se não está ocupada no período e o número de reservas para tal tipo de acomodação no período é inferior ao número total de acomodações existentes para tal tipo. Essa consulta deve mostrar uma linha para cada tipo de acomodação oferecida, constando, em cada uma dessas linhas, o código do tipo de acomodação, a descrição do tipo de acomodação, o número de acomodações existentes, o número de acomodações ocupadas, o número de acomodações reservadas e o número de acomodações disponíveis.

17. O sistema deve permitir a impressão de um relatório resumindo o faturamento do hotel no período (por exemplo, semanal ou quinzenal), contendo, para cada dia do período, um resumo das estadias pagas nesse dia, com sete colunas: diárias, frigobar, restaurante, lavanderia, telefonemas, descontos e total.

18. O sistema deve permitir a impressão diária das faturas a serem enviadas aos hóspedes que optaram pelo faturamento de suas contas. A fatura contém o nome e endereço completo do hóspede, o período de estadia, o total de diárias, o total com demais gastos, o valor do desconto, o total líquido a pagar e a data de vencimento.

19. O sistema deve permitir a impressão de um relatório contendo as faturas em atraso no período (por exemplo, semanal ou quinzenal), contendo, para cada dia do período, o nome do hóspede, a data de vencimento e o valor devido pelo hóspede

C – REQUISITOS NÃO FUNCIONAIS

C1. Confiabilidade

20. O sistema deve ter capacidade para recuperar os dados perdidos da última operação que realizou em caso de falha.

21. O sistema deve fornecer facilidades para a realização de *backups* dos arquivos do sistema.

22. O sistema deve possuir senhas de acesso e identificação para diferentes tipos de usuários: administrador do sistema, funcionários do hotel e clientes que têm acesso ao sistema no hotel (em quiosques especiais).

C2. Eficiência

23. O sistema deve responder a consultas on-line em menos de 5 segundos.

24. O sistema deve iniciar a impressão de relatórios solicitados dentro de no máximo 20 segundos após sua requisição.

C3. Portabilidade

25. O sistema deve ser executado em computadores Pentium 200mHz ou superior, com sistema operacional Windows 98 ou acima.

26. O sistema deve ser capaz de armazenar os dados em base de dados Oracle ou Sybase.

Glossário

Termo	Descrição
Acomodação	Quarto, apartamento, suíte ou outro tipo de cômodo do hotel que sirva para acomodar seus hóspedes durante sua *estadia no hotel*
Acompanhante	Pessoa que ficará hospedada na mesma *acomodação* do *hóspede* principal, porém cuja presença faz com que o valor da *diária* possa ser alterado.
Backup	Cópia de segurança ou cópia de salvaguarda
Consumo do hóspede	Refere-se a um ou mais *itens de consumo* que o hóspede utilizou durante sua *estadia no hotel* e que, portanto, deverá pagar pelos mesmos.
Diária	Valor a ser pago pelo *hóspede* referente a um dia completo de *estadia no hotel*.
Entrada do hóspede	Início de uma nova estadia de um *hóspede* no hotel. Significa que o hóspede chegou ao hotel e está ocupando uma acomodação
Estadia no hotel Funcionário Hóspede Item de consumo	Período no qual um hóspede permanece hospedado no hotel, desde sua *entrada* até sua *saída*. Pessoa que trabalha no hotel e faz a verificação do consumo do hóspede Pessoa que se hospeda no hotel por um determinado período de tempo Produto ou serviço oferecido do hotel para seus *hóspedes*, como por exemplo produtos do frigobar, refeições no restaurante, serviços de lavagem ou passagem de roupas, etc.
Pagamento da estadia	Pagamento que o hóspede deve fazer referente aos serviços prestados pelo hotel durante sua *estadia*. Além da cobrança pelo uso de uma *acomodação*, ele deve pagar pelo *consumo* de itens do frigobar, restaurante, serviços de lavanderia e telefonemas realizados.
Reserva de acomodação	Procedimento pelo qual um *tipo de acomodação* fica reservado para um *hóspede*, garantindo que haverá disponibilidade desse tipo de acomodação quando o hóspede fizer sua *entrada* no hotel.
Saída do hóspede	Fim de uma estadia de um *hóspede* no hotel. Significa que o hóspede deixou o hotel e portanto liberou sua acomodação para que outros hóspedes possam nela hospedar-se
Tipo de acomodação	Categorias nas quais as *acomodações* são divididas, para facilitar a gestão das vagas disponíveis e controlar os preços de diárias.

Apêndice B - Requisitos para o Sistema de Locadora de Carros

Sistema para Locadora de Carros

Documento de Requisitos

Novembro de 2001

A – VISÃO GERAL DO SISTEMA

O sistema para a Locadora de Carros Ubirajara consiste do gerenciamento dos *aluguéis de carros*, bem como do controle de itens que podem ser adicionados ao aluguel, tais como combustível, seguros (danos materiais e pessoais), taxa de retorno, cadeiras de bebê, motoristas, entrega do carro, etc. O sistema deve ainda emitir diversos tipos de relatórios e consultas, possibilitando um melhor gerenciamento dos aluguéis.

B – REQUISITOS FUNCIONAIS

B1 – Lançamentos diversos

1. O sistema deve permitir a inclusão, alteração e remoção de *clientes* da locadora de carros, com os seguintes atributos: nome, endereço, cidade, estado, telefone, email, documento de identificação (CPF para brasileiros e passaporte para estrangeiros), data de nascimento e nome dos pais.

2. O sistema deve permitir a inclusão, alteração e remoção das diversas *categorias de carros*, com os seguintes atributos: código da categoria, descrição da categoria, preço diário de locação, preço semanal de locação, preço mensal de locação e quantidade de carros em estoque para cada categoria.

3. O sistema deve permitir a inclusão, alteração e remoção dos *carros para aluguel* pertencentes à locadora de carros. Cada carro possui os seguintes atributos: placa do carro, código da categoria de carro, fabricante, marca, ano e modelo do carro. Para cada categoria de carro podem existir diversos carros com placas, modelos e anos diferentes.

4. O sistema deve permitir a inclusão, alteração e remoção de *funcionários* da locadora de carros, com os seguintes atributos: nome, endereço, cidade, estado, telefone e data de nascimento.

5. O sistema deve permitir a inclusão, alteração e remoção dos tipos de *serviços adicionais* oferecidos pela locadora de carros. Cada tipo de serviço adicional possui os seguintes atributos: código do tipo de serviço oferecido, descrição do serviço adicional e preço.

6. O sistema deve permitir o processamento da *reserva de carro*, com os seguintes atributos: data e hora de retirada do carro, data e hora previstas para devolução do carro, identificação do cliente, categoria de carro desejada, valor da *diária*, *funcionário* responsável pela reserva, taxa de multa a ser cobrada em caso de desistência de última hora (a menos de 4 horas do início previsto de retirada), os dados do cartão de crédito do cliente e desconto concedido (opcional). A reserva só deve ser permitida se houverem carros do tipo requisitado disponíveis no período indicado. Caso contrário o sistema deve emitir uma mensagem de alerta e a reserva não pode ser confirmada. A remoção de reserva somente é permitida sem maiores encargos até 4 horas antes do início previsto para *aluguel do carro*. Após esse período, a remoção da reserva deve alertar o *funcionário* do hotel de que deve ser cobrada a taxa de multa estabelecida durante a reserva.

7. O sistema deve permitir a *retirada de carro* por um cliente. Cada retirada de carro possui os seguintes atributos: data e hora de retirada do carro, data e hora prevista para devolução do carro, identificação do cliente (previamente cadastrado), *funcionário* responsável pela retirada, placa do carro alugado, quilometragem registrada no momento da retirada, discriminação dos serviços adicionais contratados, valor da *diária* e desconto concedido (opcional). Se tiver sido feita a reserva prévia do carro, então, durante a retirada, informa-se o nome do cliente e os dados da reserva são recuperados automaticamente pelo sistema e alterados pelo *funcionário*, se necessário.

8. O sistema deve permitir a *devolução do carro* por um cliente, com os seguintes atributos: placa do carro alugado, data e hora de devolução do carro, quilometragem registrada no momento da devolução, número de *diárias* cobradas, valor de cada diária, valor dos gastos com combustíveis, e desconto concedido (opcional). O sistema deve automaticamente somar às diárias e combustível, as taxas cadastradas durante a retirada do carro, referentes aos *serviços adicionais* contratados. O sistema deve também apresentar na tela o total a pagar, que é a soma das diárias, acrescentando-se o combustível, serviços adicionais e subtraindo-se o desconto, se houver. Além disso deve-se cobrar uma *taxa de serviço* de 5% sobre os *serviços adicionais*.

9. O sistema deve permitir as seguintes opções de *pagamento do aluguel* do carro: 1) à vista (em dinheiro, cheque ou cartão de crédito); 2) faturado em 30 dias.

10. O sistema deve permitir a quitação de uma fatura paga pelo cliente, contendo as seguintes informações: número da fatura, data de vencimento, data de pagamento, valor total pago, juros e multa.

B2 – Impressão de diversos tipos de relatórios e consultas

11. O sistema deve permitir a impressão de uma listagem dos carros alugados no momento, contendo o nome do cliente, placa, marca e modelo do carro, data de retirada e data prevista para devolução.

12. O sistema deve permitir a impressão de uma listagem das reservas efetuadas para a data atual, contendo o nome do cliente, telefone para contato, categoria de carro e data prevista para saída.

13. O sistema deve permitir a impressão de um comprovante de *retirada do carro*, contendo o nome do *cliente*, documento, datas e horários de retirada e previsão de devolução, valor da diária e *serviços adicionais* utilizados. Nesse mesmo comprovante deve ser mostrada uma lista dos serviços adicionais utilizados, a forma de pagamento escolhida e deve ser reservado um espaço para assinatura do cliente.

14. O sistema deve permitir a impressão de um comprovante de *devolução do carro*, contendo o nome do cliente, documento, datas e horários de retirada e devolução do carro, número de diárias cobradas, valor da diária, valor para reabastecimento do combustível, discriminação dos serviços adicionais utilizados, descontos e taxa sobre serviços. Deve ainda mostrar o total líquido a pagar, bem como a forma de pagamento escolhida, reservando um espaço para assinatura do cliente.

15. O sistema deve permitir ao cliente imprimir um histórico de seus *aluguéis de carro*. Para tal o cliente deve ter sido previamente cadastrado e deve portar um código de identificação e uma senha. Esse histórico deve conter uma linha para cada carro alugado pelo cliente, contendo as datas de retirada e devolução e os totais pagos em cada ocasião.

16. O sistema deve permitir a consulta on-line da ocupação dos carros num certo período. Um carro está ocupado se existe um cliente utilizando-o no momento. Um carro está disponível se não está ocupado no período e o número de reservas para tal categoria de carro no período é inferior ao número total de carros existentes para tal categoria. Essa consulta deve mostrar uma linha para cada categoria de carro oferecida, constando, em cada uma dessas linhas, o código da categoria de carro, a descrição da categoria de carro, o número de carros existentes, o número de carros ocupadas, o número de carros reservados e o número de carros disponíveis.

17. O sistema deve permitir a impressão de um relatório resumindo o faturamento da locadora de carros no período (por exemplo, semanal ou quinzenal), contendo, para cada dia do período, um resumo dos aluguéis pagos nesse dia, com seis colunas: diárias, combustível, serviços adicionais, *taxas de serviço*, descontos e total.

18. O sistema deve permitir a impressão diária das faturas a serem enviadas aos clientes que optaram pelo faturamento de suas contas. A fatura contém o nome e endereço completo do cliente, o período de locação do carro, o total de diárias, o total com demais serviços contratados, o valor do desconto, o total líquido a pagar e a data de vencimento.

19. O sistema deve permitir a impressão de um relatório contendo as faturas em atraso no período (por exemplo, semanal ou quinzenal), contendo, para cada dia do período, o nome do cliente, a data de vencimento e o valor devido pelo cliente.

C – REQUISITOS NÃO FUNCIONAIS

C1. Confiabilidade

20. O sistema deve ter capacidade para recuperar os dados perdidos da última operação que realizou em caso de falha.

21. O sistema deve fornecer facilidades para a realização de *backups* dos arquivos do sistema.

22. O sistema deve possuir senhas de acesso e identificação para diferentes tipos de usuários: administrador do sistema, funcionários da locadora de carros e clientes que têm acesso ao sistema na locadora de carros (em quiosques especiais).

C2. Eficiência

23. O sistema deve responder a consultas on-line em menos de 5 segundos.

24. O sistema deve iniciar a impressão de relatórios solicitados dentro de no máximo 20 segundos após sua requisição.

C3. Portabilidade

25. O sistema deve ser executado em computadores Pentium 200mHz ou superior, com sistema operacional Windows 98 ou acima.

26. O sistema deve ser capaz de armazenar os dados em base de dados Oracle ou Sybase.

Glossário

Termo	Descrição
Aluguel do carro	Ato de emprestar um carro a um *cliente* por um determinado período de tempo, desde a *retirada do carro* até a *entrega do carro*.
Backup	Cópia de segurança ou cópia de salvaguarda
Carro para aluguel	Carro disponível para *aluguel* por um *cliente* por um determinado período de tempo.
Categoria de carro	Categorias nas quais os carros são divididos, para facilitar a gestão das vagas disponíveis e controlar os preços de diárias, por exemplo, carros econômicos, intermediários ou executivos.
Cliente	Pessoa que aluga um carro na locadora por um determinado período de tempo
Devolução do carro	Fim do *aluguel do carro*. Significa que o carro foi devolvido à locadora pelo *cliente*.
Diária	Valor a ser pago pelo *cliente* referente a um dia completo de *aluguel do carro*.
Funcionário	Pessoa que trabalha na locadora de carros e faz a verificação do aluguel e *serviços adicionais* requisitados pelo *cliente*.
Pagamento do aluguel	Pagamento que o *cliente* deve fazer referente aos serviços prestados pela locadora de carros durante o *aluguel do carro*. Além da cobrança pelo uso do carro, ele deve pagar pelos serviços adicionais utilizados.
Reserva de carro	Procedimento pelo qual um *carro para aluguel* fica reservado para um *cliente*, garantindo que haverá disponibilidade desse tipo de carro quando o cliente fizer sua *retirada do carro*.
Retirada do carro	Início de um novo *aluguel de carro* por um *cliente*. Significa que o cliente levou o carro ou o carro será levado até ele pela locadora de veículos.
Serviço adicional	Serviço oferecido pela locadora de carros a um *cliente*, como por exemplo, seguros (danos materiais e pessoais), taxa de retorno (se a locadora precisa buscar o carro em um local estabelecido pelo cliente), cadeira de bebê, motorista, entrega do carro (se a locadora precisa levar o carro até o cliente em um local por ele estabelecido), etc.
Taxa de serviço	Percentual fixo cobrado sobre os *serviços adicionais*, que deve ser pago pelo *cliente* que faz uma *retirada de carro*. Essa taxa é repassada pela locadora às pessoas que executaram tais serviços.

Apêndice C - Requisitos para o Sistema de Clínica Veterinária

Sistema para Clínica Veterinária

Documento de Requisitos

Setembro de 2002

A – VISÃO GERAL DO SISTEMA

O sistema para a Clínica Veterinária Animal & Cia trata do gerenciamento das consultas realizadas em animais domésticos (por exemplo, cães e gatos). A consulta pode ser de rotina, mas pode implicar em diagnósticos que envolvam outros serviços a serem prestados pelo veterinário, como injeções, vacinação, cirurgias, etc. Além disso, o animal pode precisar de medicamentos, que podem ser adquiridos na própria clínica. A clínica possui também diversos produtos para venda, como rações, brinquedos, casas de madeira, shampoos, escovas, bebedouros, etc. Esses produtos podem ser vendidos separadamente, ou integrados a uma consulta. Diversos relatórios devem ser gerados pelo sistema para permitir a gestão adequada da clínica, como o relatório de estoque de medicamentos e produtos, consultas feitas em um determinado animal, relatório de vendas de produtos, etc.

B – REQUISITOS FUNCIONAIS

B1 – Lançamentos diversos

1. O sistema deve permitir a inclusão, alteração e remoção de *animais*, com os seguintes atributos: código do animal, nome do *animal* (opcional), identificação de seu proprietário (que é um *cliente*), data de nascimento (opcional), *espécie do animal*, *raça*, sexo e cor.

2. O sistema deve permitir a inclusão, alteração e remoção de *clientes*, com os seguintes atributos: código, nome, endereço, cidade, estado, telefone, documento, email.

3. O sistema deve permitir a inclusão, alteração e remoção de *espécie de animais*, com os seguintes atributos: código de identificação e espécie do animal.

4. O sistema deve permitir a inclusão, alteração e remoção de *raças*, com os seguintes atributos: código de identificação e descrição da raça.

5. O sistema deve permitir a inclusão, alteração e remoção dos diversos *produtos* vendidos na clínica veterinária, com os seguintes atributos: código do produto, descrição do produto, fabricante, categoria (se é alimento, acessórios, medicamento, etc.), preço de venda e quantidade em estoque.

6. O sistema deve permitir a inclusão, alteração e remoção das diversas *categorias de produtos*, com os seguintes atributos: código e descrição da categoria.

7. O sistema deve permitir a inclusão, alteração e remoção de *veterinários* da Animal & Cia, com os seguintes atributos: nome, endereço, cidade, estado, telefone residencial e telefone celular.

8. O sistema deve permitir a inclusão, alteração e remoção de *tipos de serviços* prestados durante a *consulta*, com os seguintes atributos: código de identificação, descrição do tipo de serviço e preço cobrado por hora.

9. O sistema deve permitir o processamento da *consulta de um animal*, com os seguintes atributos: data e hora da consulta, identificação do *cliente/animal* (previamente cadastrado), identificação do *veterinário* que efetuou a consulta, descrição dos sintomas do animal, identificação dos *serviços* executados como parte da consulta, relação de medicamentos e *produtos* usados na consulta e forma de pagamento.

10. O sistema deve permitir o processamento de *venda de produto*, com os seguintes atributos: data da venda, identificação do cliente (previamente cadastrado), relação de produtos adquiridos, valor total e forma de pagamento.

11. O sistema deve permitir as seguintes opções de *pagamento da consulta*: 1) à vista (em dinheiro ou cheque); 2) em 1 ou mais cheques pré-datados.

B2 – Impressão de diversos tipos de relatórios e consultas

12. O sistema deve permitir a impressão de uma listagem dos produtos/medicamentos em estoque, agrupados por categorias, contendo a descrição do produto, fabricante, preço, e quantidade em estoque.

13. O sistema deve permitir a impressão de um comprovante de *consulta*, contendo o nome do *cliente*, data e horário da consulta, nome do veterinário responsável, serviços prestados, produtos adquiridos e valores totais. Nesse mesmo comprovante deve ser mostrada a forma de pagamento escolhida e deve ser reservado um espaço para assinatura do cliente.

14. O sistema deve permitir a impressão de um relatório de animais, classificados por espécie, contendo: raça, nome do animal, nome do proprietário, data de nascimento, sexo e cor.

15. O sistema deve permitir a impressão de uma consulta online mostrando toadas as *consultas* realizadas em um determinado animal, agrupadas por espécie, contendo o nome do animal, raça, nome do proprietário, data da consulta, veterinário responsável, serviços realizados e valor total.

16. O sistema deve permitir a impressão de um relatório resumindo do faturamento da Animal & Cia. no período (por exemplo, semanal ou quinzenal), contendo, para cada dia do período, um resumo das consultas e vendas nesse dia.

17. O sistema deve permitir a impressão de um relatório contendo os cheques pré-datados a serem depositados no dia contendo o nome do cliente, a data do cheque e valor.

C – REQUISITOS NÃO FUNCIONAIS

C1. Confiabilidade

18. O sistema deve ter capacidade para recuperar os dados perdidos da última operação que realizou em caso de falha.

19. O sistema deve fornecer facilidades para a realização de *backups* dos arquivos do sistema.

20. O sistema deve possuir senhas de acesso e identificação para diferentes tipos de usuários: administrador do sistema, veterinários e clientes que têm acesso ao sistema da clínica veterinária (em quiosques especiais).

C2. Eficiência

21. O sistema deve responder a consultas on-line em menos de 5 segundos.

22. O sistema deve iniciar a impressão de relatórios solicitados dentro de no máximo 20 segundos após sua requisição.

C3. Portabilidade

23. O sistema deve ser executado em computadores Pentium 200mHz ou superior, com sistema operacional Windows 98 ou acima.

24. O sistema deve ser capaz de armazenar os dados em base de dados Oracle ou Sybase.

Glossário

Termo	Descrição
Animal	Animal que é *consultado* na clínica, podendo ser de diversas *espécies* e *raças.*
Backup	Cópia de segurança ou cópia de salvaguarda
Categoria de Produto	Diversas classificações nas quais os produtos vendidos pela clínica podem ser agrupados, por exemplo: medicamentos, alimentos, acessórios, etc.
Cliente	Proprietário de um ou mais *animais* que podem ser consultados na clínica.
Consulta	Evento no qual um *animal* comparece à clínica para ser feito um ou mais *serviços* como consulta para diagnóstico de doença, vacinação, etc., podendo-se também, durante a consulta, vender medicamentos ou outros *produtos* para animais.
Espécie do animal	Espécie na qual os animais são classificados, por exemplo, cães, gatos, etc.
Pagamento da consulta	Pagamento que o proprietário do *animal* deve fazer referente aos serviços prestados pela clínica e produtos adquiridos durante a *consulta.*
Produto	Diversos bens vendidos pela clínica ao proprietário do *animal*, como por exemplo vacinas, remédios anti-piolhos, ração, bebedouro, shampoos, etc.
Raça	Raça específica de uma espécie de animal, por exemplo, um cão pode ser Labrador, Pastor alemão, Coker, Poodle, etc.
Tipo de Serviço	Serviço oferecido pela clínica ao *animal*, como por exemplo, vacinação, aplicação de injeções, banho, tosa, cirurgias, etc.
Veterinário	Profissional contratado pela clínica para efetuar *consulta* em *animais* que comparecem à clínica.

Apêndice D - Formulários usados no E-GRN-1 a 3

COLETA DE TEMPO PARA ANÁLISE DE UM SISTEMA

FICHA PARA CADA GRUPO

Sistema: ( ) Hotel Data: ___/___/_____
( ) Locadora de Carros
( ) Loja de Venda e Aluguel de Produtos para festas

GRUPO nº _____ Quantidade de alunos: _____

Resumo de Coleta de tempo do grupo: _____ Pessoas/Hora

Algumas observações:

1) Fazer os diagramas somente a mão, não esquecendo de acrescentar os atributos e métodos de cada classe, bem como os nomes e cardinalidade dos relacionamentos. Anota-se o tempo gasto na modelagem, incluindo o tempo de leitura dos requisitos. Depois pode-se passar a limpo, mas não anote o tempo gasto para tal.

2) O tempo gasto para preencher a ficha não deve ser incluído nas coletas. Cada pessoa mede seu tempo individualmente, mesmo se estiver trabalhando em grupo. Exemplo, se o grupo se reune por 1 hora, em três pessoas, cada um dos elementos do grupo gastou 1 pessoa/hora e o total do grupo é de 3 pessoas/hora.

3) Responder às perguntas abaixo somente depois de terminar a coleta. Durante a análise, pedir para algum membro do grupo anotar possíveis dificuldades para facilitar responder o questionário.

Questionário:

1) O grupo teve alguma dificuldade em entender os requisitos? Quais?

_____

_____

2) O grupo encontrou algum requisito em duplicidade? Quais?

_____

_____

3) O grupo encontrou requisitos vagos ou inconsistentes, que merecem ser melhor detalhados? Quais?

_____

_____

4) Comentários (usar o verso, se necessário)

_____

_____

_____

_____

COLETA DE TEMPO PARA ANÁLISE DE UM SISTEMA

FICHA INDIVIDUAL

Sistema: ( ) Hotel Data: ___/___/_____
( ) Locadora de Carros
( ) Loja de Venda e Aluguel de Produtos para festas

GRUPO nº _____ Nome do aluno: _____

COLETA DE TEMPOS

Data	Início	Término	Tempo (pessoas/hora)	Comentários

Curso: ( ) Ciência da Computação ( ) Bacharelado em Informática ( ) Processamento de dados
( ) Outros _____

Nível: ( ) Graduação ( ) Mestrado ( ) Doutorado ( ) Especialização

Horário ( ) Diurno ( ) Noturno Período atual: _____ do total de ____ semestres

Em qual das categorias abaixo você melhor se encaixa, enquanto desenvolvedor?
( ) Não tenho experiência em desenvolvimento de sistemas desse tipo (sistemas comerciais)
( ) Desenvolvi alguns projetos desse tipo durante disciplinas de graduação/pós-graduação, usando análise estruturada/essencial
( ) Desenvolvi alguns projetos desse tipo durante disciplinas de graduação/pós-graduação, usando análise orientada a objetos
( ) Desenvolvi, profissionalmente, alguns projetos nesse domínio (até 3)
( ) Desenvolvi, profissionalmente, muitos projetos nesse domínio (mais de 4)
( ) Outros, especifique: _____

Por favor, informe quais disciplinas você já cursou na área de análise de sistemas/engenharia de software

_____

_____

Qual sua maior área de interesse:

( ) Engenharia de Software ( ) Redes/Sistemas Distribuídos ( ) Banco de Dados
( ) Inteligência Artificial ( ) Hipermídia ( ) Arquitetura de computadores
( ) Computação Gráfica ( ) Outros : _____

Observações ou comentários: (favor anotar no verso)

COLETA DE TEMPO PARA ANÁLISE DE UM SISTEMA USANDO A LINGUAGEM DE PADRÕES GRN

FICHA PARA CADA GRUPO

Sistema: ( ) Hotel Data: ___/___/_____
( ) Locadora de Carros
( ) Loja de Venda e Aluguel de Produtos para festas

GRUPO nº _____ Quantidade de alunos: _____

Resumo de Coleta de tempo do grupo: _____ Pessoas/Hora

Algumas observações:

1) Antes de iniciar a análise do sistema, estudar a linguagem de padrões GRN. O tempo gasto nesse estudo pode ser anotado à parte, nas observações, mas não deve ser computado na análise do sistema.
2) Já o tempo gasto para leitura dos requisitos de especificação deve contar.
3) Fazer os diagramas somente a mão, não esquecendo de acrescentar os atributos e métodos de cada classe, bem como os nomes dos relacionamentos.
4) No diagrama de classes, desenhar balões e anotar o papel desempenhado por cada classe em cada padrão, como no exemplo dado em sala de aula.
5) O tempo gasto para preencher a ficha não deve ser incluído nas coletas.

Questionário:

1) O grupo teve alguma dificuldade em entender os requisitos? Quais?

_____

_____

2) O grupo encontrou algum requisito em duplicidade? Quais?

_____

_____

3) O grupo encontrou requisitos vagos ou inconsistentes, que merecem ser melhor detalhados? Quais?

_____

_____

4) Comentários

_____

_____

_____

_____

_____

COLETA DE TEMPO PARA ANÁLISE DE UM SISTEMA USANDO A LINGUAGEM DE PADRÕES GRN

FICHA INDIVIDUAL

Sistema: ( ) Hotel Data: ___/___/_____
( ) Locadora de Carros
( ) Loja de Venda e Aluguel de Produtos para festas

Nome do aluno: _____

COLETA DE TEMPOS

Data	Início	Término	Tempo (pessoas/hora)	Comentários

Observações ou comentários:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Apêndice E - Formulários usados no E-GRN-GREN

PROJETO: INSTANCIAÇÃO DO FRAMEWORK GREN PARA UM SISTEMA DE CONTROLE DE CLÍNICA VETERINÁRIA

INSTRUÇÕES GERAIS

- O projeto será feito individualmente, não devendo haver nenhum tipo de comunicação a respeito do projeto entre os alunos da disciplina.
- Dúvidas devem ser solucionadas diretamente com o professor da disciplina ou com a monitora (Rosana)
- Embora sejam fornecidos processos para instanciação do framework GREN, o projeto pode ser executado seguindo os conhecimentos do aluno, desde que anote como está sendo feita a instanciação do framework.
- Não se espera que toda a funcionalidade descrita no documento de requisitos seja implementada, mas apenas o que for coberto pelo framework GREN.
- O questionário Q1 deve ser entregue hoje.
- O questionário Q2 deve ser entregue no dia 12/11/02.
- Preencher, na ficha F1, os tempos gastos com quaisquer atividades relacionadas ao projeto, anotando separadamente o tempo gasto com cada tipo de atividade, por exemplo, estudo do método, estudo da linguagem (VisualWorks), análise do sistema, implementação de classes, teste, etc. Ao término do projeto, faça uma estimativa do tempo total gasto com cada atividade, e também com as atividades especificadas na página 3 da ficha F1.
- Faça os diagramas de classes à mão, anotando o tempo gasto para tal. Se for passar a limpo em uma ferramenta Case, anote esse tempo em separado.
- O tempo gasto para preencher as fichas não deve ser computado.
- Anote quaisquer dificuldades encontradas em uma folha, para facilitar responder o questionário Q2 no término do estudo de caso.

Q1 - QUESTIONÁRIO INDIVIDUAL PARA PARTICIPANTES DO ESTUDO DE CASO

Nome: _____

Técnica utilizada: ( ) Ad-hoc ( ) Linguagem de Padrões

Sistema: _____ Data: ___/___/_____

Curso: ( ) Ciência da Computação ( ) Bacharelado em Informática ( ) Processamento de dados
( ) Outros _____

Nível: ( ) Graduação ( ) Mestrado ( ) Doutorado ( ) Especialização

Horário ( ) Diurno ( ) Noturno Período atual: _____ do total de ____ semestres

Em qual das categorias abaixo você melhor se encaixa, enquanto desenvolvedor?
( ) Não tenho experiência em desenvolvimento de sistemas desse tipo (sistemas comerciais)
( ) Desenvolvi alguns projetos desse tipo durante disciplinas de graduação/pós-graduação, usando análise estruturada/essencial
( ) Desenvolvi alguns projetos desse tipo durante disciplinas de graduação/pós-graduação, usando análise orientada a objetos
( ) Desenvolvi, profissionalmente, alguns projetos nesse domínio (até 3)
( ) Desenvolvi, profissionalmente, muitos projetos nesse domínio (mais de 4)
( ) Outros, especifique: _____

Por favor, informe quais disciplinas você já cursou na área de análise de sistemas/engenharia de software

_____

_____

_____

_____

Conhece linguagens orientadas a objetos? _____ Quais? _____

Qual sua maior área de interesse:

( ) Engenharia de Software ( ) Redes/Sistemas Distribuídos ( ) Banco de Dados
( ) Inteligência Artificial ( ) Hipermídia ( ) Arquitetura de computadores
( ) Computação Gráfica ( ) Outros : _____

Observações ou comentários: (favor anotar no verso, caso o espaço abaixo seja insuficiente)

_____

_____

Declaro concordar em participar do estudo de caso, considerando que os resultados permanecerão confidenciais.

Local/Data: _____ Assinatura: _____

F1 - COLETA DE TEMPO PARA INSTANCIAÇÃO DO FRAMEWORK Pg. 1/3

Nome: _____

Data	Início	Término	Tempo (pessoas/hora)	Atividades

Data	Início	Término	Tempo (pessoas/hora)	Comentários

Resumo das Atividades:

Atividade	*Total de horas*

Por favor, faça uma estimativa dos seguintes tópicos:

Tópico	**Total**
tempo gasto antes de inicia r a implementação da primeira linha de código	
tempo gasto antes de identificar a primeira classe do framework a ser especializada	
número total de classes e de métodos implementados na versão final do sistema	
tempo gasto na fase de depuração e tes tes da nova aplicação	
tempo total gasto na implementação do sistema exemplo	

Q2 - QUESTIONÁRIO INDIVIDUAL PARA PARTICIPANTES DO ESTUDO DE CASO

Nome: _____

Técnica utilizada: AD-HOC

Quanto ao documento de requisitos do sistema da Veterinária, responda:

1) Você teve alguma dificuldade em entender os requisitos? Quais?

_____

_____

2) Você encontrou algum requisito em duplicidade? Quais?

_____

_____

3) Você encontrou requisitos vagos ou inconsistentes, que merecem ser melhor detalhados? Quais?

_____

_____

4) Comentários adicionais

_____

_____

_____

_____

_____

Quanto à instanciação do framework GREN usando a técnica AD-HOC, responda:

5) Quais as dificuldades encontradas no entendimento da técnica (justificar)

_____

_____

_____

6) Quais as dificuldades para entendimento do framework?

_____

_____

_____

_____

_____

7) Quais as dificuldades para encontrar o exemplo mais apropriado?

_____

_____

_____

_____

8) Quais as dificuldades no ambiente VisualWorks? Especifique, por exemplo, o treinamento foi suficiente para cobrir os aspectos necessários para instanciar o GREN? Problemas de instalação, da linguagem, etc...

_____

_____

_____

_____

Outras dificuldades:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Q2 - QUESTIONÁRIO INDIVIDUAL PARA PARTICIPANTES DO ESTUDO DE CASO

Nome: _____

Técnica utilizada: LINGUAGEM DE PADRÕES

Quanto ao documento de requisitos do sistema da Veterinária, responda:

1) Você teve alguma dificuldade em entender os requisitos? Quais?

_____

_____

2) Você encontrou algum requisito em duplicidade? Quais?

_____

_____

3) Você encontrou requisitos vagos ou inconsistentes, que merecem ser melhor detalhados? Quais?

_____

_____

4) Comentários adicionais

_____

_____

_____

_____

_____

Quanto à instanciação do framework GREN usando a técnica LINGUAGEM DE PADRÕES, responda:

5) Quais as dificuldades encontradas no entendimento da técnica (justificar)

_____

_____

_____

6) Quais as dificuldades para entendimento do framework?

_____

_____

_____

_____

_____

7) Quais as dificuldades para identificar o(s) padrão(ões) mais apropriado(s)?

_____

_____

_____

_____

8) Quais as dificuldades no ambiente VisualWorks? Especifique, por exemplo, o treinamento foi suficiente para cobrir os aspectos necessários para instanciar o GREN? Problemas de instalação, da linguagem, etc...

_____

_____

_____

_____

Outras dificuldades:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Apêndice F – Documentação adicional do GREN e do GREN-Wizard

Hierarquia de Classes do GREN

As classes do VisualWorks aparecem em itálico. As demais são as classes do GREN, divididas em camadas de acordo com a Figura 4.7. Os atributos de cada classe aparecem entre parênteses. Para dar a noção de hierarquia, foram incluídos três traços para cada nível de especialização (por exemplo, InterestRate é especializada de ExactNumberCalculator, que é especializada de AbstractCalculator, e assim por diante).

Camada de Persistência

Object ()
---ConnectionManager ()
---OIDManager ('className' 'lastIdCode')
---PersistentObject ('isChanged' 'isPersisted')

Camada de Negócios

Object()
---DatePeriod('initialDate' 'finishingDate')
---DiscreteList ('values')
---PersistentObject ('isChanged' 'isPersisted')
------AbstractCalculator ('percentageRate')
---------ExactNumberCalculator ('number')
----------InterestRate ()
---------NumberRangeCalculator ('lowerNumber' 'upperNumber')
----------FineRate ()
------BusinessResourceTransaction ('number' 'date' 'observation' 'status' 'totalPrice' 'totalDiscount'
'finalTotal' 'destinationParty' 'sourceParty' 'resource' 'items' 'executor' 'transQuantification')
---------BasicMaintenance ('faultsPresented' 'tasks' 'parts' 'totalTasks' 'totalParts')
----------MaintenanceQuotation ('expirationDate' 'maintenanceNumber')
----------ResourceMaintenance ('exitDate' 'quotationNumber')
---------BasicNegotiation ('freight' 'taxes' 'oldQtty')
----------BasicDelivery ('tradeNumber')
----------PurchaseDelivery ()
----------SaleDelivery ()
----------ResourceTrade ('quotationNumber')
----------BasicPurchase ()
----------BasicSale ()
---------ResourceRental ('finishingDate' 'returnDate' 'reservationNumber' 'fineValue')
---------ResourceReservation ('period')
------BusinessResourceQuotation ('number' 'date' 'observation' 'status' 'sourceParty' 'quotationItems' 'executor')
------Commission ('transaction' 'executor' 'date' 'aValue')
------LocationZoneStCond ('locationZoneIdCode' 'stCond' 'condition')
------MaintenancePart ('transaction' 'part' 'qtty' 'cost')
------MaintenanceTask ('transaction' 'problemToSolve' 'laborDescription' 'hoursSpent' 'cost' 'executor')
------NToNRelationship('obj1' 'obj2')
------Payment ('dueDate' 'paymentDate' 'installmentNumber' 'paymentType' 'value' 'status')
------PaymentStrategy ('transaction' 'installmentNumber')
---------ImmediateReceiving ()
----------Cash ('suppliedValue')
----------MoneyOrder ('bank')
----------ElectronicTransfer ('cardNumber' 'bank')
---------LaterReceiving ()
----------CashOnDelivery ('paymentForm')
----------Check ('accountNumber' 'bank' 'checkNumber')
----------CreditCard ('cardNumber' 'type' 'bank' 'expirationDate' 'paymentDay')

----------Invoice ('invoiceNumber')
------QuotationItem ('transaction' 'destinationParty' 'resource' 'price' 'expirationDate' 'quantity')
------ResourceInstance ('resourceIdCode' 'code' 'allocation' 'status')
------ResourceLot ('number' 'qtty')
------ResourceStCond ('resourceIdCode' 'stCond' 'condition')
------StaticObject ('idCode' 'description')
---------DestinationParty ()
---------LocationZone ('capacity' 'storageConditions' 'storageLocationIdCode')
---------MeasureUnity ()
---------QualifiableObject ('types')
----------NestedType ()
----------Resource ('quantification')
----------StockResource('cost' 'salePrice')
----------StorableResource ('locationZone' 'storageConditions')
----------SimpleType()
----------Warehouse ('storageLocations')
---------SourceParty ()
---------StorageCondition ()
---------StorageLocation ('locationZones' 'warehouseIdCode')
---------TransactionExecutor ('specialty' 'percentage' 'minimumValue' 'salary')
------TransactionItem ('transaction' 'resource' 'aValue' 'itemQuantification')
------QuantificationStrategy ('resource')
---------InstantiableResource ('instances')
---------MeasurableResource ('quantityInStock' 'reSupplyLevel' 'measureUnity')
----------LotableResource ('lotNumber')
---------SingleResource ('status')
------TransactionQuantificationStrategy ('transaction')
---------InstantiableResTransaction ('instanceCode')
---------ItemQuantificationStrategy ('transactionItem')
----------InstResTransItem ('instanceCode')
----------MeasResTransItem ('quantity')
----------LotResTransItem ('lotNumber')
----------SingleResTransItem ()
---------MeasurableResTransaction ('quantity' 'unitaryValue')
----------LotableResTransaction ('lotNumber')
---------SingleResTransaction ()
---Label ('linesPerPage' 'spaces' 'rows' 'columns' 'title' 'header' 'showDate')
---Report ('header' 'title' 'showDate' 'showPageNumber' 'linesPerPage' 'spaces' 'rows' 'columns')
---ReportColumn ('number' 'label' 'width' 'group' 'classify' 'totalize' 'totalizeGroup' 'calculatePercentage')
---ReportBySourceParty ('description' 'totalValue')
---ReportDailyGains ('date' 'totalValue')
---ReportMonthlyGains ('monthNumber' 'totalValue')
---ReportMostTransacted ('resourceDescription' 'quantity')
---ReportQuotation ('resourceDescription' 'destinationPartyDescription' 'price' 'expirationDate' 'quantity')

Camada de Interface com o Usuário

Object ()
---*Model* ('dependents')
------*ApplicationModel* ('builder' 'uiSession')
---------BasicBusinessTransactionForm ('number' 'aTransaction' 'date' 'observation' 'status' 'totalPrice'
'totalDiscount' 'aDestinationParty' 'allDestinationParties' 'aSourceParty' 'allSourceParties'
'aExecutor' 'allExecutors' 'editing' 'inserting' 'finalTotal' 'basicBtnsCanvas' 'totalsCanvas'
'totalsCanvasSpec')
----------MultiResourceTransactionForm ('itemsSubform' 'itemsSpec')
----------MultiResourceRentalForm ('finishingDate' 'returnDate' 'reservationNumber'
'returning' 'fineValue')
----------MultiResourceNegotiationForm ('freight' 'taxes')
----------MultiResourceDeliveryForm ('tradeNumber')
----------MultiResourceTradeForm ('quotationNumber')
----------MultiResourceReservationForm ('beginDate' 'endDate')
----------OneResourceTransactionForm ('aResource' 'allResources' 'instanceCode' 'quantity'

'unitaryValue' 'lotNumber')
----------OneResourceReservationForm ('beginDate' 'endDate')
----------InstantiableResReservationForm ()
----------OneResourceMaintenanceForm ('faultsPresented' 'exitDate' 'closing')
----------OneResourceMaintNPNTForm ('tasks' 'parts')
----------OneResourceMaintWPWTForm ('tasks' 'parts')
----------OneResourceMaintQuotForm ('faultsPresented expirationDate effected')
----------OneResourceMaintQuotationNPNTForm ('tasks' 'parts')
----------OneResourceMaintQuotationWPWTForm ('tasks' 'parts')
----------OneResourceRentalForm ('finishingDate' 'returnDate' 'reservationNumber' 'returning' 'fineValue')
----------InstantiableResourceRentalForm ()
----------MeasurableResourceRentalForm ()
----------OneResourceNegotiationForm ('freight' 'taxes')
----------OneResourceDeliveryForm ('tradeNumber')
----------OneResourceTradeForm ()
----------InstantiableResourceTradeForm ()
----------MeasurableResourceTradeForm ()
---------BasicQuotationForm ('number aTransaction date observation status chosenDestinationPartiesList
chosenResourcesList allDestinationParties aResource allResources aSourceParty allSourceParties
aExecutor allExecutors editing inserting basicBtnsCanvas ')
---------ExactNumberCalculatorForm ('aFineRate' 'percentageRate' 'number')
----------InterestRateForm()
---------GRENApplicationMainForm ()
---------GRENApplicationModel ('parentApplication')
----------BasicButtonsForm ()
----------BasicReturnButtonsForm ()
----------BasicNvgButtonsForm ()
----------BasicTotalsForm ('totalPrice' 'totalDiscount' 'finalTotal')
----------BasicMaintenanceTotalsForm ('totalPart' 'totalTask')
----------BasicRentalTotalsForm ('fineValue' 'saleValue')
----------BasicTradeTotalsForm ('freight' 'taxes')
----------CreateListForm('nameList' 'valuesList' 'isChanged')
----------DiscreteListForm()
----------TableListForm()
----------MaintenancePartForm ('partDescription' 'qtty' 'cost' 'total' 'subtotal' 'partsUsedList'
'editing' 'inserting' 'aPartUsed' 'allParts' 'isChanged')
----------MaintenanceTaskForm ('laborDescription' 'hoursSpent' 'problemToSolve' 'total' 'cost'
'tasksList' 'editing' 'inserting' 'aTask' 'aExecutor' 'allExecutors' 'isChanged
----------NToNRelationshipForm ('selectedList' 'completeList' 'isChanged')
----------PaymentForm ('aPayment allInstallments isSaved dueDate paymentDate installmentNumber
totalValue status interest fine finalTotal paymentType allPaymentTypes navigationBtns
totalInstallments suppliedValue changeValue cardNumber type bank accountNumber checkNumber
invoiceNumber expirationDate paymentDay paymentForm ')
----------QualificationForm0 ()
----------QualificationForm1 ('resourceType1' 'allResourceTypes1')
----------QualificationForm2 ('resourceType2' 'allResourceTypes2')
----------QualificationForm3 ('resourceType3' 'allResourceTypes3')
----------QualificationForm4 ('resourceType4' 'allResourceTypes4')
----------TransactionItemsDataSetForm ('resource' 'aValue' 'quantity' 'instanceCode' 'lotNumber'
'rowsList' 'itemsList' 'selectedItem')
---------NumberRangeCalculatorForm ('aFineRate' 'allFineRates' 'percentageRate' 'lowerNumber' 'upperNumber')
----------FineRateForm
---------StaticObjectForm ('aStaticObject' 'allStaticObjects' 'w_idCode' 'w_description'
'navigationBtns' 'basicBtnsCanvas')
----------DestinationPartyForm ()
----------LocationZoneForm ('storageConditions' 'capacity' 'storageLocationDescr' 'warehouseDescr'
'allStorageConditions')
----------MeasureUnityForm ()
----------QualifiableObjectForm ('typesCanvas' 'typesCanvasSpec')
----------ResourceForm ()
----------InstantiableResourceForm ('instancesList' 'selectedInstance' 'selectedRow')
----------MeasurableResourceForm ('quantityInStock' 'reSupplyLevel' 'measureUnity' 'allMeasureUnities')

----------SingleResourceForm ('status')
----------WarehouseForm ('storageLocations')
----------SourcePartyForm ()
----------StorageCondit ionForm ()
----------StorageLocationForm ('locationZones' 'warehouseDescr')
----------TransactionExecutorForm ('specialty' 'percentage' 'minimumValue' 'salary')
---------*SimpleDialog* (. . .)
----------DatePeriodForm ('aDatePeriod' 'initialDate' 'finishingDate')
----------LanguageModificationForm ('language')
----------QuotationItemForm ('quantity' 'expirationDate' 'price' 'quotationForm' 'aQuotationItem' 'destPartList'
'resourceList')
----------ResourceInstanceForm ('aResourceInstance' 'code' 'allocation' 'status')
----------StorageForm ('aResource' 'locationZoneDescr' 'storageConditions' 'allLocationZones' 'allStorageConditions')
----------TransactionItemForm ('anItem' 'transactionNumber' 'resourceDescription' 'aValue' 'resourcesList'
'quantity' 'instanceCode' 'lotNumber' 'editing')
----------RentalItemForm ()

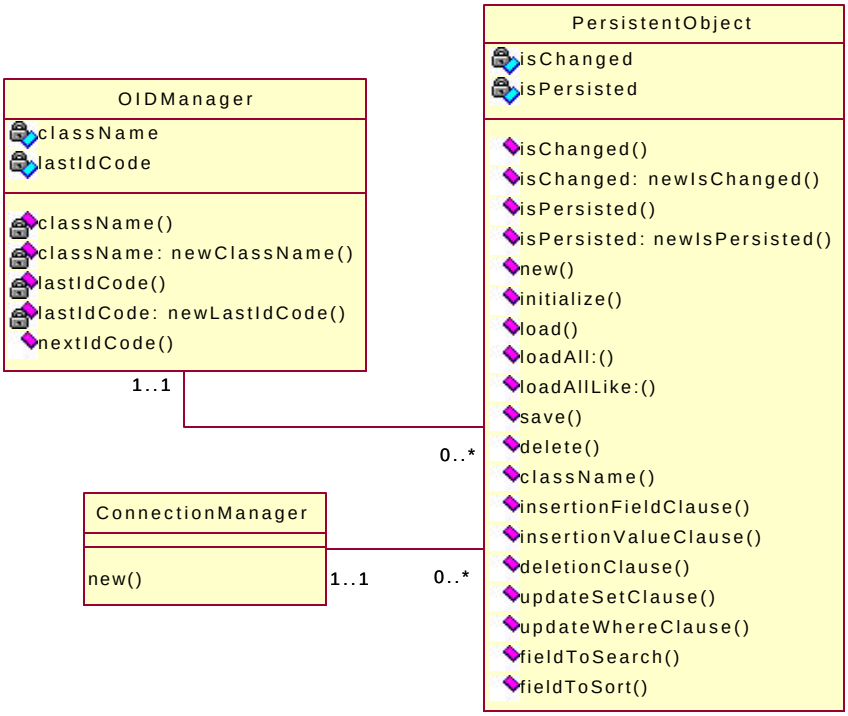
Hierarquia de Classes do GREN-Wizard

Object ()
---GRNHistory ('aHistory' 'appIdCode')
---*PersistentObject* ('isChanged' 'isPersisted')
------AppClass ('appClassIdCode' 'appIdCode' 'patternNumber' 'variantNumber' 'classNumber' 'name' 'plural')
---------TempAppClass()
------AppClassAttribute ('appIdCode' 'appClassName' 'sequentialNb' 'patternAttributeName' 'attributeName'
'attributeType' 'attributeLength' 'canDelete')
------AppliedPattern ('patternNumber' 'variantNumber')
------AppReport ('appIdCode' 'sequentialNb' 'reportNb' 'title')
------AttributeMapping ('p_name' 'p_type' 'p_length' 'app_name' 'app_type' 'app_length' 'canDelete')
------ClassMethod ('seq' 'className' 'methodName' 'methodType' 'protocolName' 'preCondition' 'methodBody')
---------ClassAtrMethod ()
---------ClassSpecialTypeMethod('atrType')
------Element ('appIdCode' 'sequentialNb' 'patternNumber' 'variantNumber' 'transactionClassName' 'resourceClassName'
'assocTransClassName')
---------OldElement ('visited' 'toBeDeleted')
------EquivalentPatternClass ('patternNumber' 'variantNumber' 'classNumber' 'ePatternNumber' 'eVariantNumber'
'eClassNumber')
------GRENApplication ('idCode' 'name' 'databaseName' 'language' 'databaseLocation' 'aGRNHistory' 'oldGRNHistory')
------GRENReport ('superClassName' 'grenReportMethod' 'methodNb' 'reportType')
------NextPattern ('patternNumber' 'nextPattern')
------Pattern ('patternNumber' 'patternName' 'specName' 'mandatory')
------PatternClass ('patternNumber' 'variantNumber' 'classNumber' 'className' 'pluralClassName' 'superclassName'
'refCode' 'canIncludeAttribute' 'mandatory')
------PatternClassAttribute ('patternNumber' 'variantNumber' 'classNumber' 'attributeName' 'attributeType'
'attributeLength' 'isMandatory')
------PatternClassForm ('refCode' 'seq' 'patternNumber' 'variantNumber' 'superclassName' 'specName' 'addToMenu'
'nextHorizontal' 'nextVertical')
------PatternReport ('patternNumber' 'className' 'operation' 'reportNb' 'grenReportMethod'
'grenReportTitleMethod' 'methodNb')
------PatternVariant ('patternNumber' 'variantNumber' 'description')
------SelectableReport ('number' 'name' 'title' 'wasSelected')
------*StaticObject* ('idCode' 'description')
---------AttributeType ('initializeString')
------TypeAppClass('appIdCode' 'appClassName' 'plural ')
---*Model* ('dependents')
------*ApplicationModel* ('builder' 'uiSession')
---------*GRENApplicationMainForm* ()
----------GrenWizardGUI ('aPatForm' 'editing' 'app' 'appsList' 'sequenceNb' 'transName' 'transPlName'
'resName' 'resPlName' 'assocTransName' 'lastNonLeafSeqNb' 'updateRes' 'updateTrans' 'isChanged'
'lastVisited' 'newGeneratedApp' 'aGenWin')
---------*GRENApplicationModel* ('parentApplication')
----------AttributeMappingForm ('anAttribute' 'aClassName' 'allAttributes' 'app_length' 'p_type' 'app_type' 'app_name'
'p_name' 'sequentialNb' 'p_length' 'isChanged' 'rolePlayed')
----------AttributesImportation ('classesPlayingRole' 'listOfAddedAttributes' 'aSuperClassName')
----------GenerationWindow ('consistency' 'application' 'attributes' 'interface' 'menus' 'methods')
----------GrenWizardCodeGenerator ('app' 'appNameSpace' 'menuClass' 'aConcreteClassName' 'anAddedAttribute'
'aClassMethod')
---------- PatternForm ('wasApplied' 'wasSkipped' 'patternNumber' 'variantApplied' 'allVariants' 'variantChosen'
'nextPattern' 'nextPatternChosen' 'allNextPatterns' 'currentElement' 'adding' 'onlyRevisiting'
'class1Name' 'class2Name' 'class3Name' 'class4Name' 'class5Name' 'class6Name' 'class7Name'
'class8Name' 'class9Name' 'class10Name' 'class11Name' 'class12Name' 'plClass1Name' 'plClass2Name'
'plClass3Name' 'plClass4Name' 'plClass5Name' 'plClass6Name' 'plClass7Name' 'plClass8Name'
'plClass9Name' 'plClass10Name' 'plClass11Name' 'plClass12Name')
----------ReportsSelection ('aTitle' 'history' 'appliedPatterns' 'selectedReports' 'listOfReports' 'listOfClasses'
'isChanged')
----------HtmlForms ('introductionText')
----------AttributeListForm('listApp_type' 'list_tables' 'list_attributes' 'isChanged')
---------- AttributeDiscreteListForm()

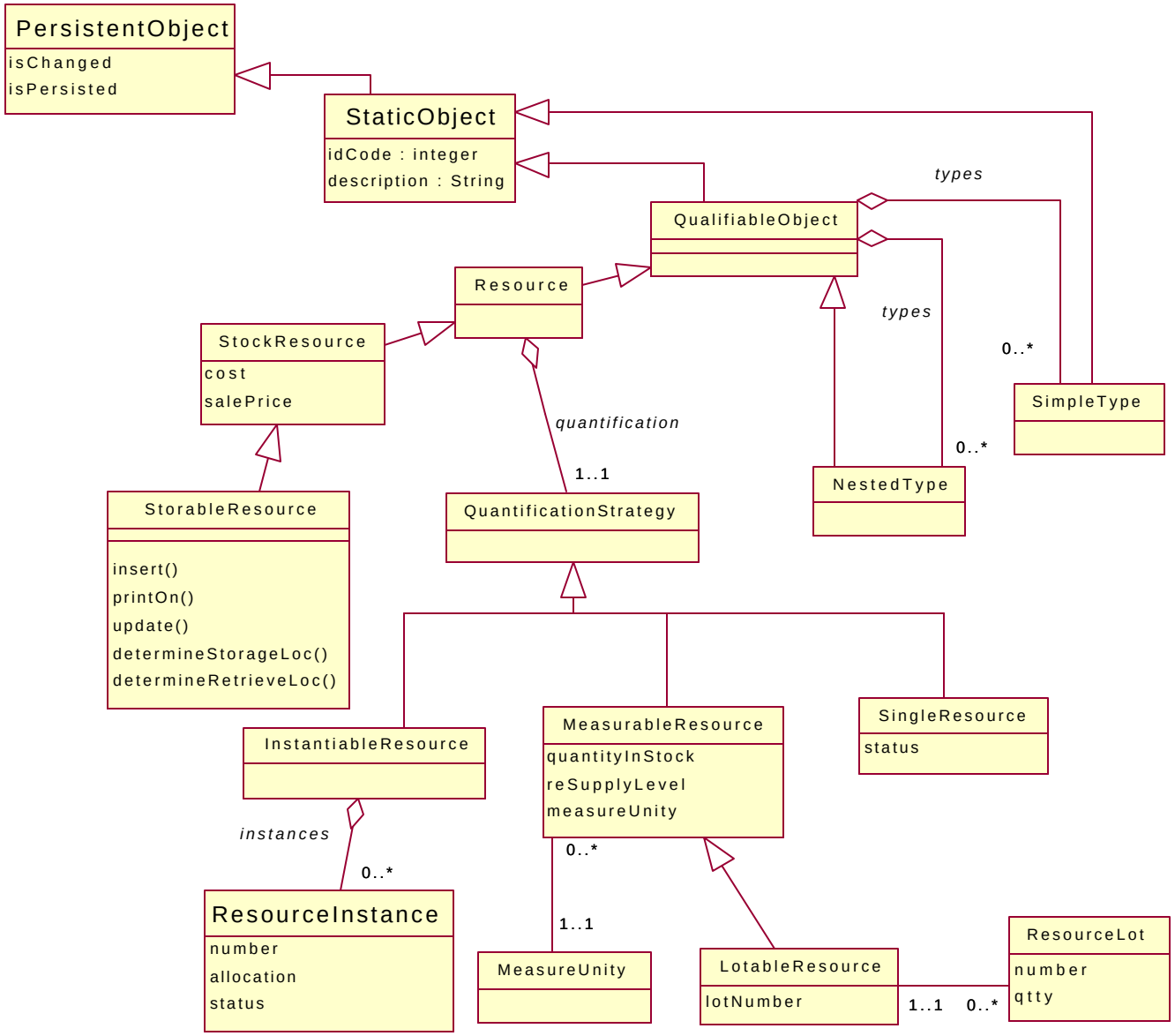
---------- AttributeTableListForm()
----------CreateListForm('nameList' 'valuesList' 'isChanged')
----------DiscreteListForm()
----------TableListForm()
---------PatternLanguageDatabase ()
---------*SimpleDialog* ('close' 'accept' 'cancel' 'preBuildBlock' 'postBuildBlock' 'postOpenBlock' 'escapeIsCancel' 'parentView')
----------GRENApplicationForm ('anApplication' 'idCode' 'name' 'databaseName' 'language' 'databaseLocation')

Diagramas de Classes do GREN (usando a UML)

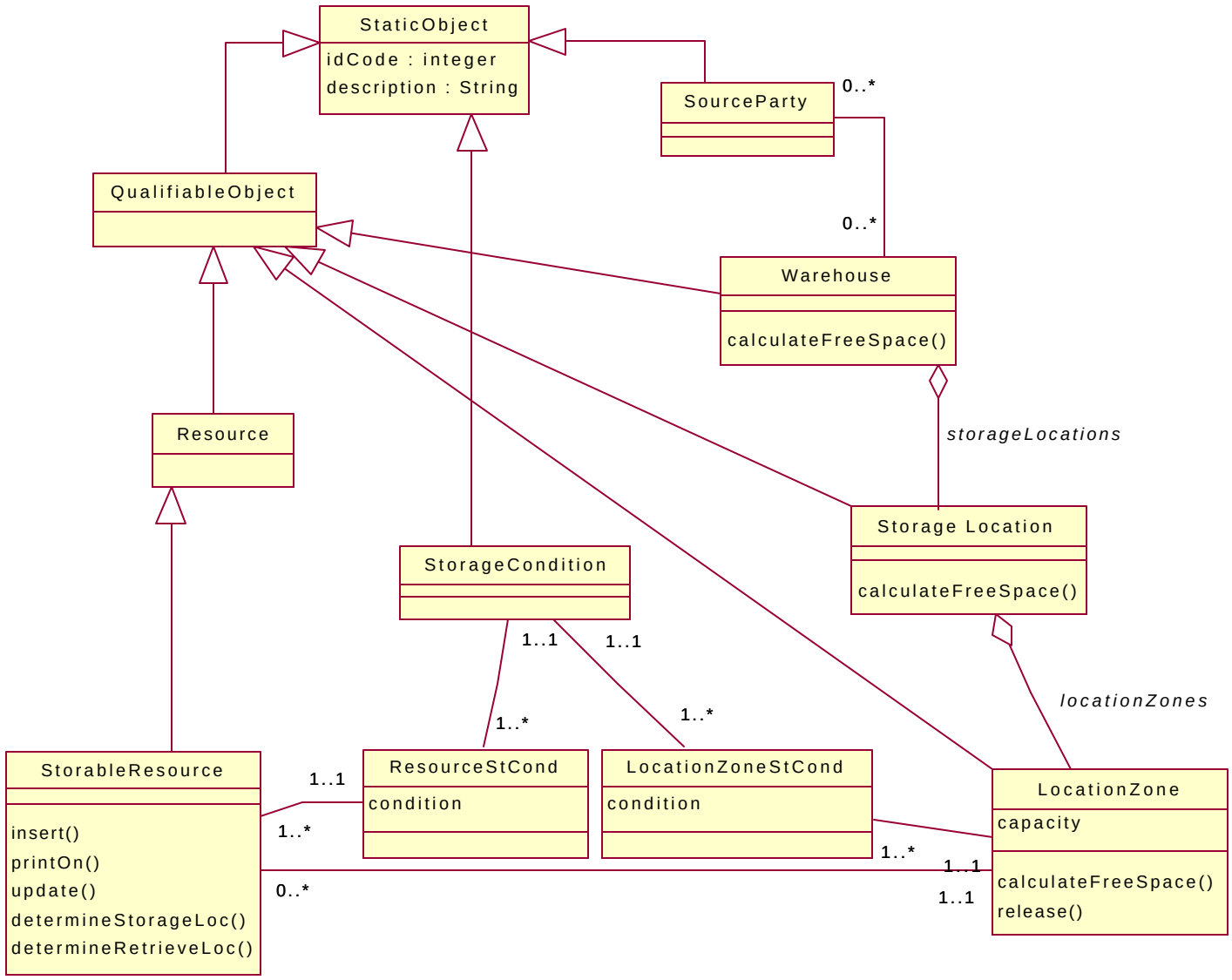
Camada de Persistência



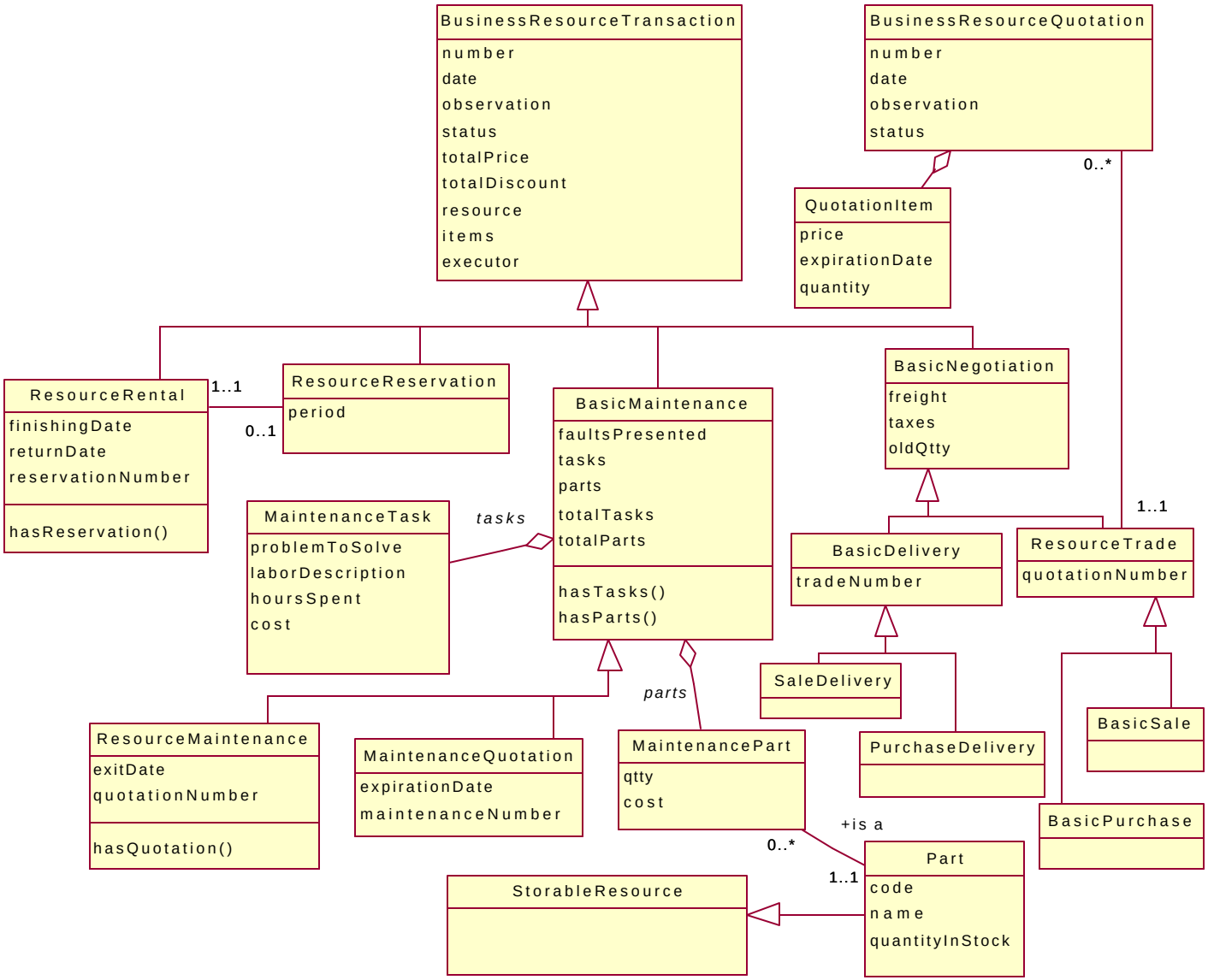
Camada de Negócios – Classes relacionadas a Recurso – Padrões 1 e 2



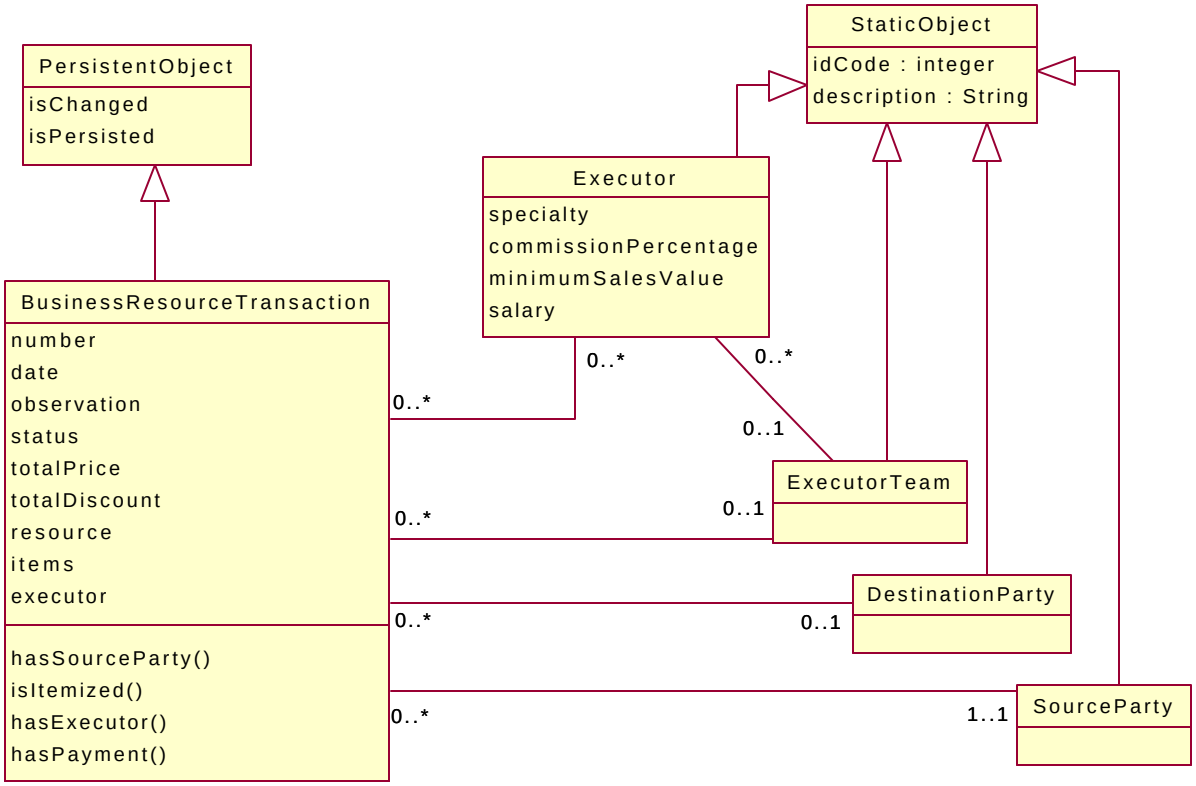
Camada de Negócios – Classes relacionadas a Recurso – Padrão 3



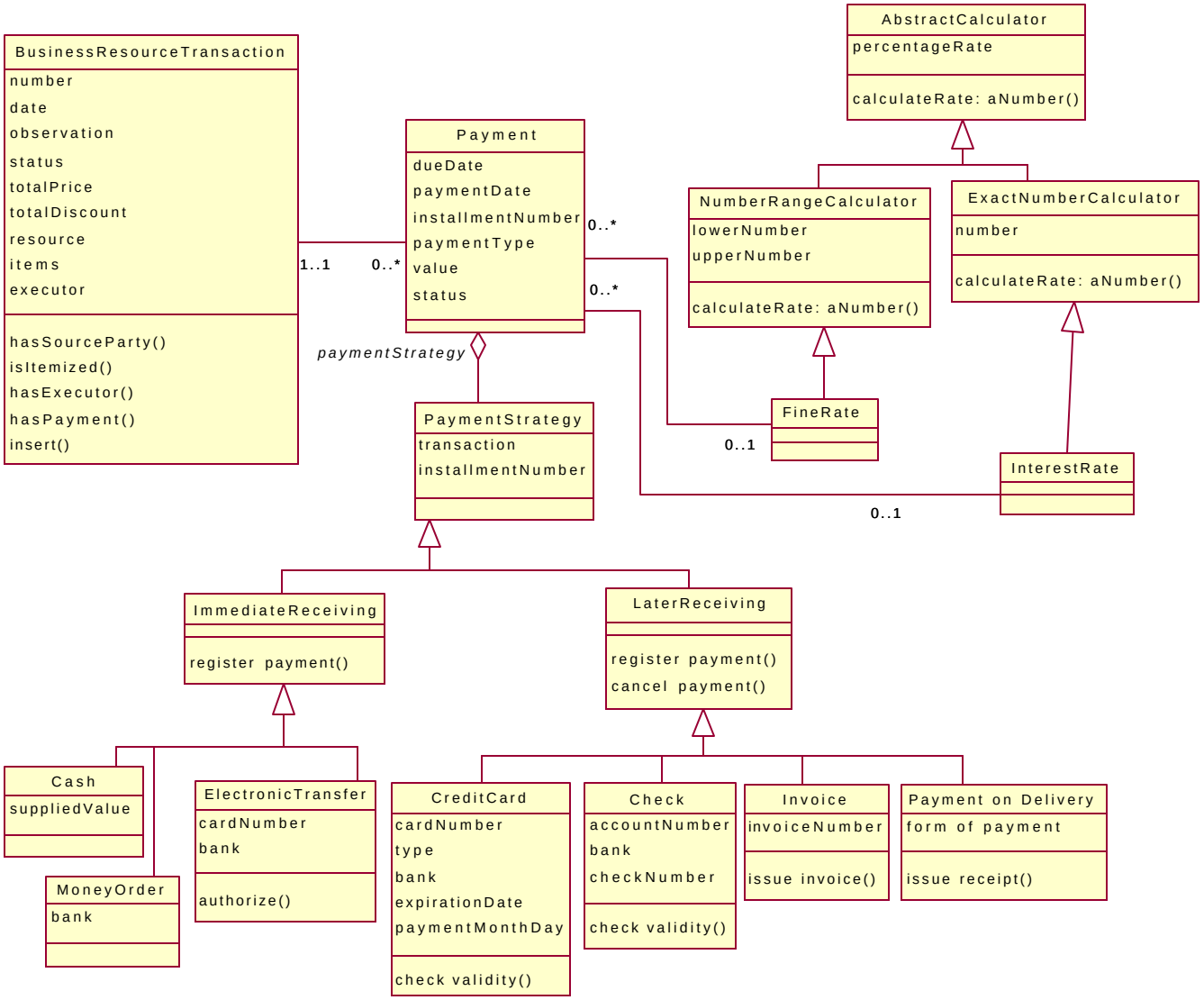
Camada de Negócios – Transações (padrões 4 a 10)



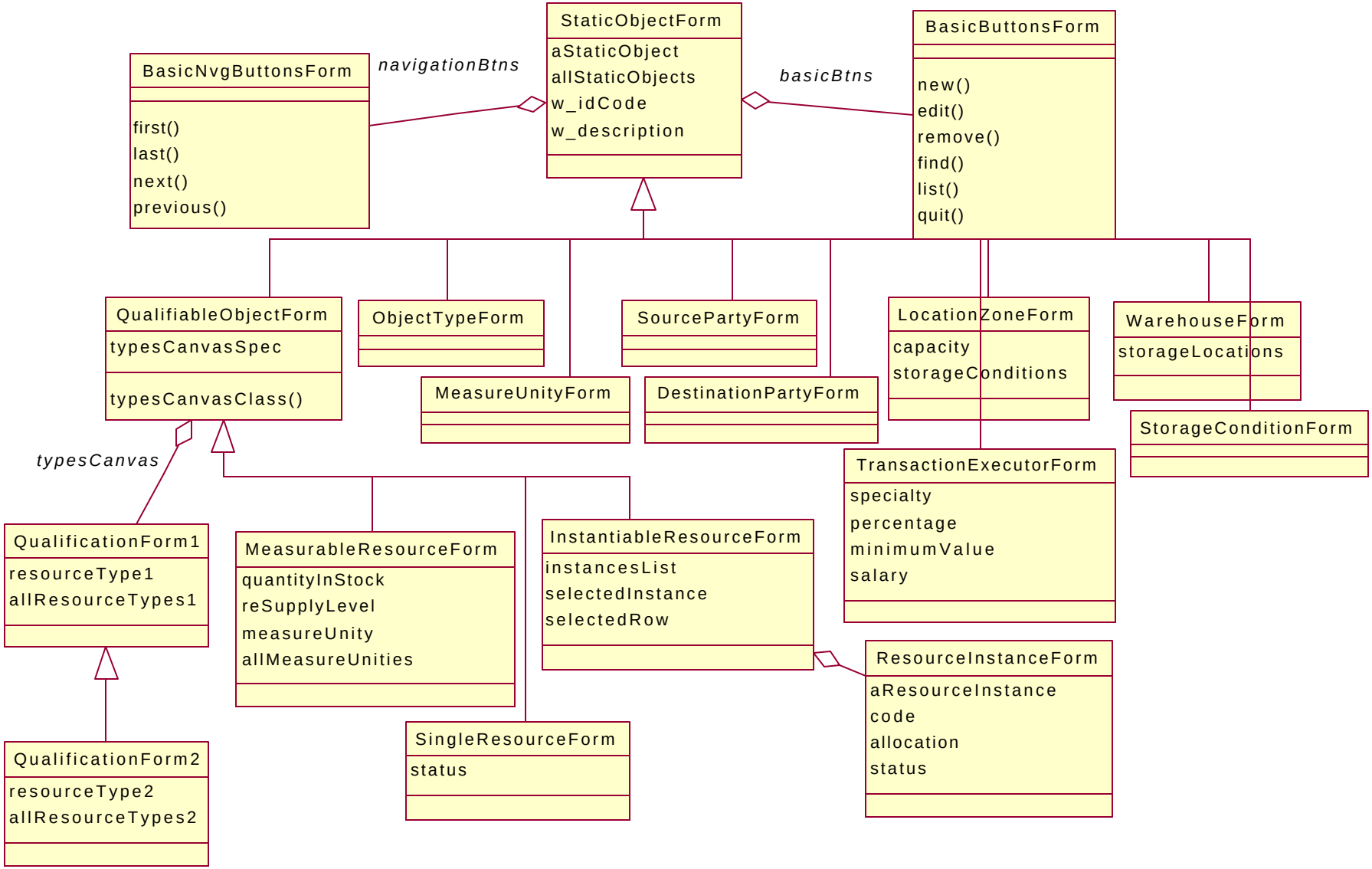
Camada de Negócios – Transações de Negócios – Padrão 13



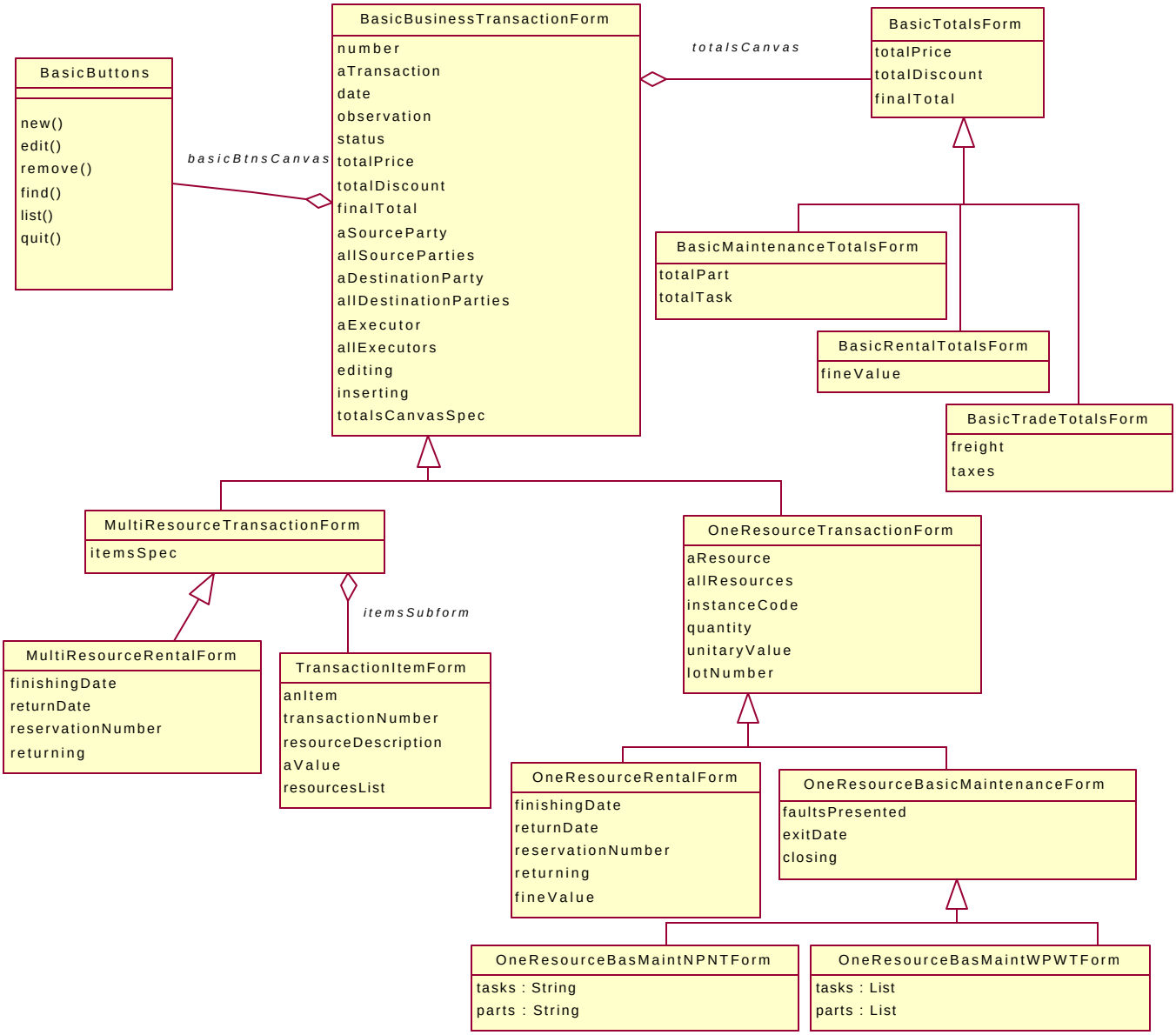
Camada de Negócios - Pagamento



Camada de Interface Gráfica com o Usuário – classes herdando de StaticObjectForm



Camada de Interface Gráfica com o Usuário – Classes referentes a Transações



Apêndice G - Figuras Referentes aos Processos

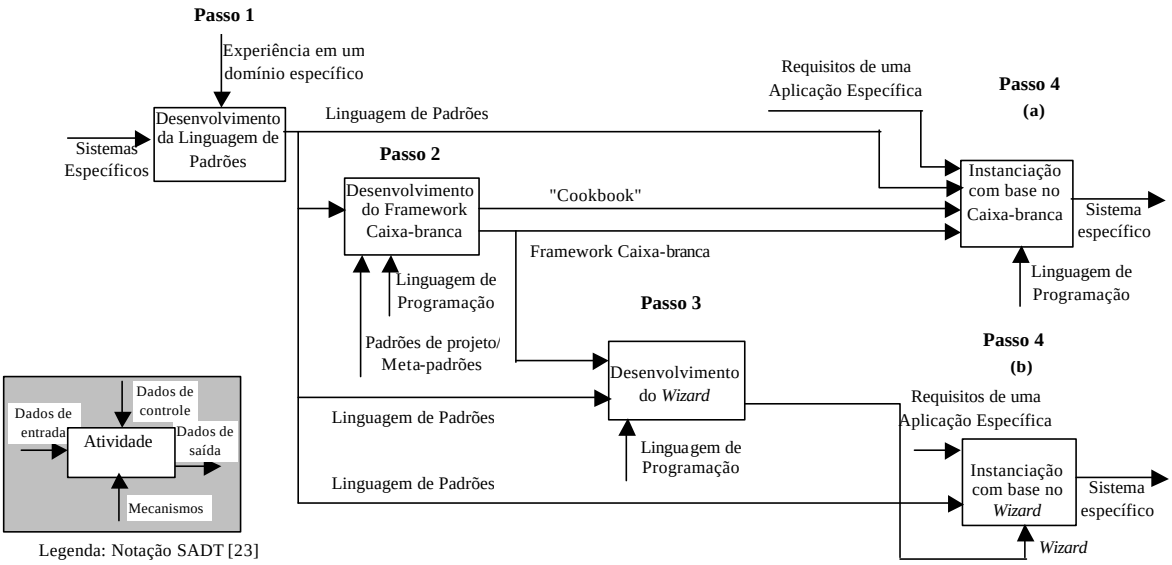


Figura 3 .2: Processo Proposto

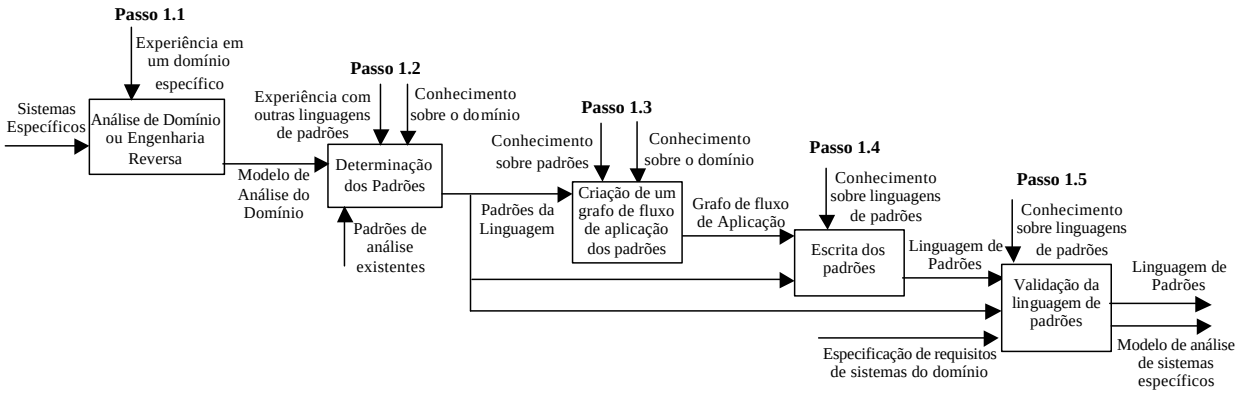


Figura 3.3: Processo de Construção da Linguagem de Padrões

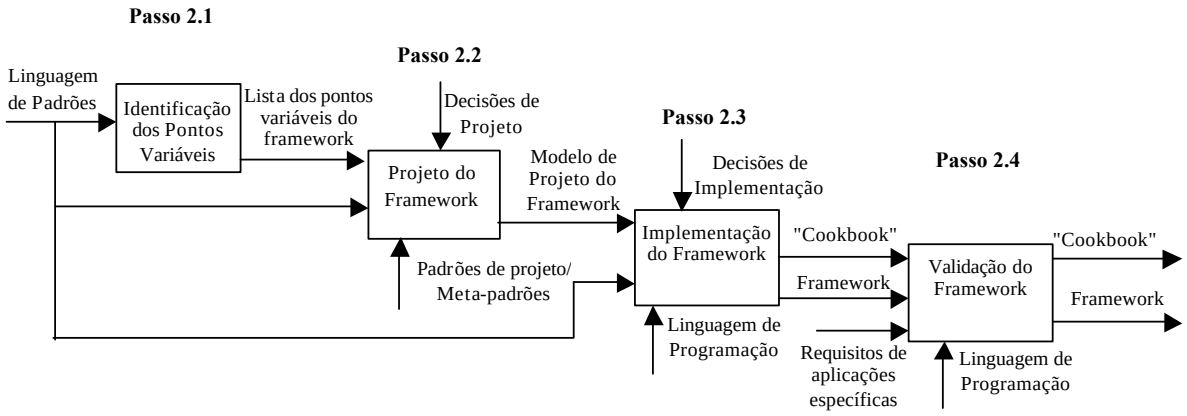


Figura 4 .1: Processo para construção de um framework baseado em uma linguagem de padrões

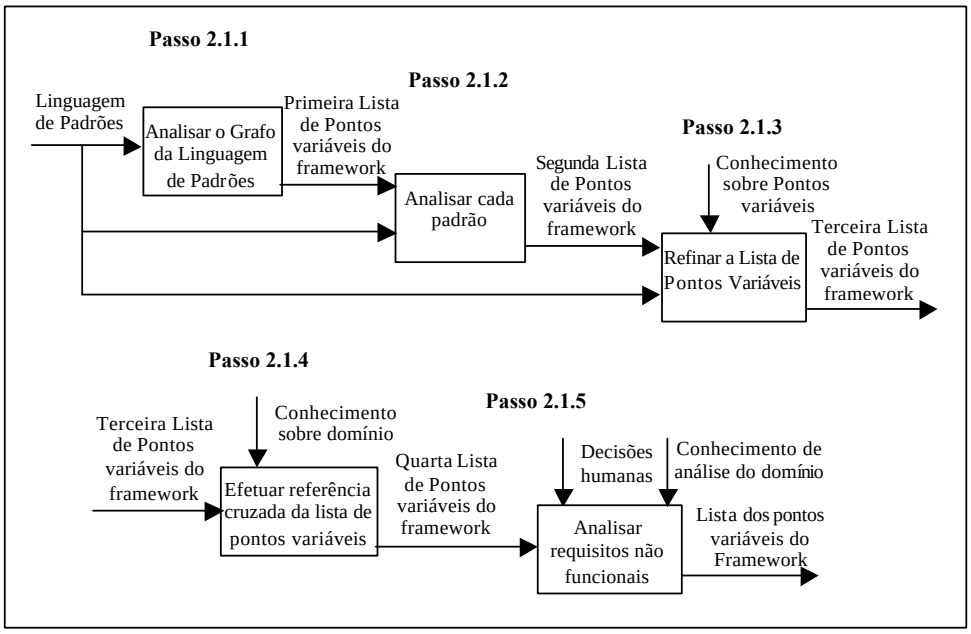


Figura 4 .2: Identificação dos pontos variáveis do framework

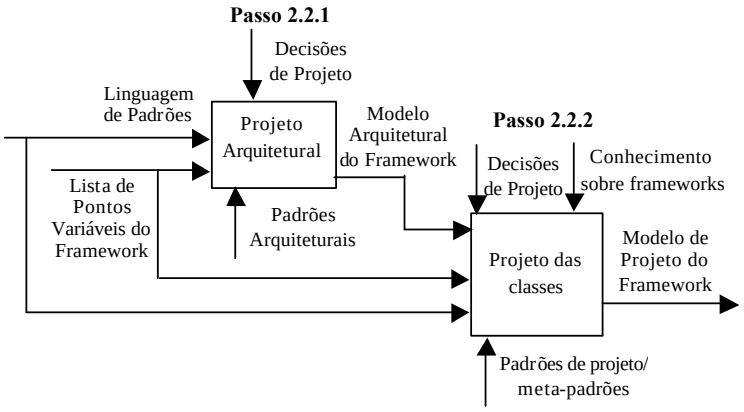


Figura 4.3: Projeto do Framework

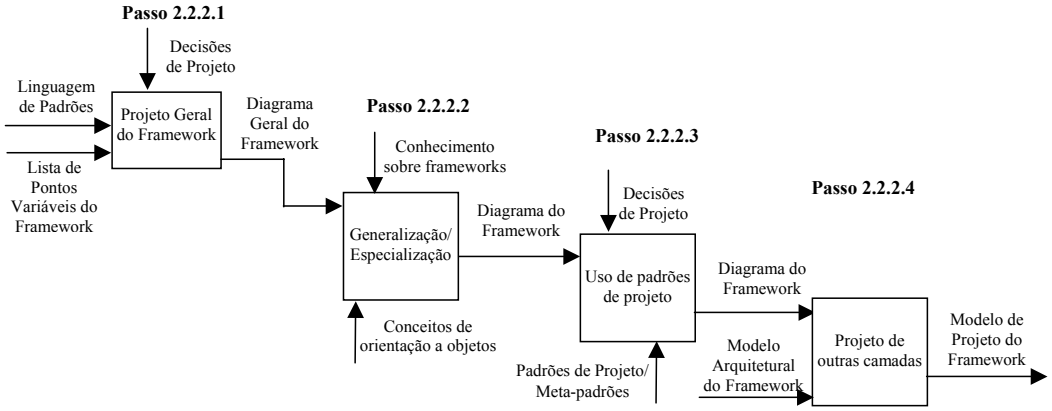


Figura 4 .4: Criação da Hierarquia de Classes do Framework

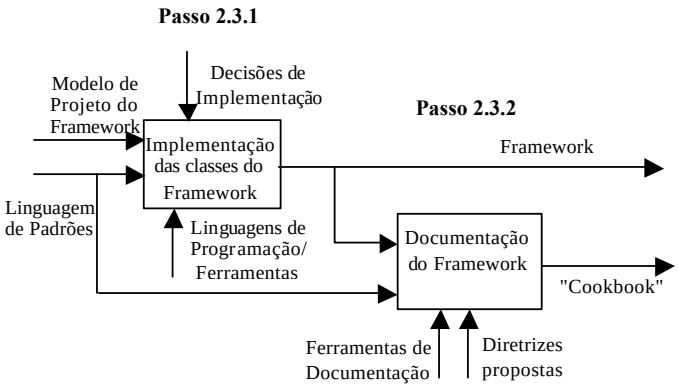


Figura 4 .5: Implementação do Framework

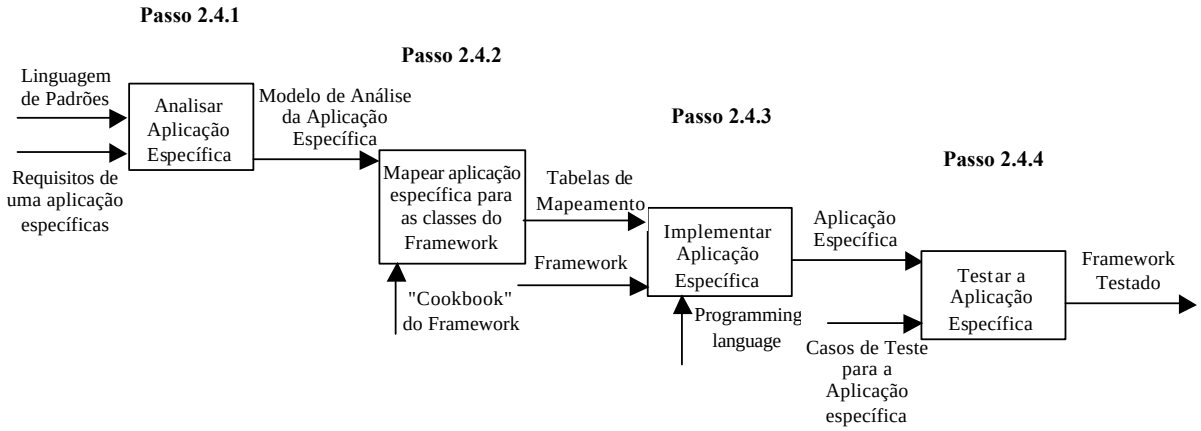


Figura 4 .6: Validação do Framework

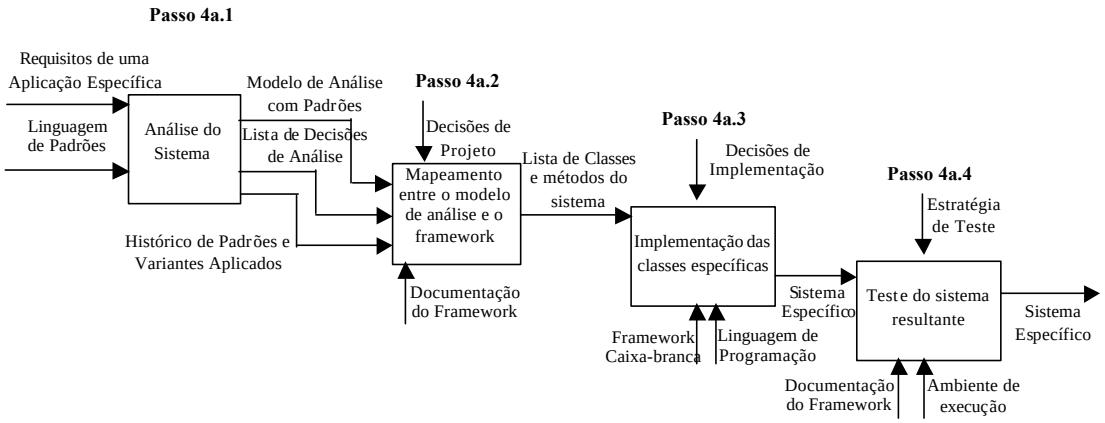


Figura 4 .9: Processo de Instanciação do Framework Caixa branca

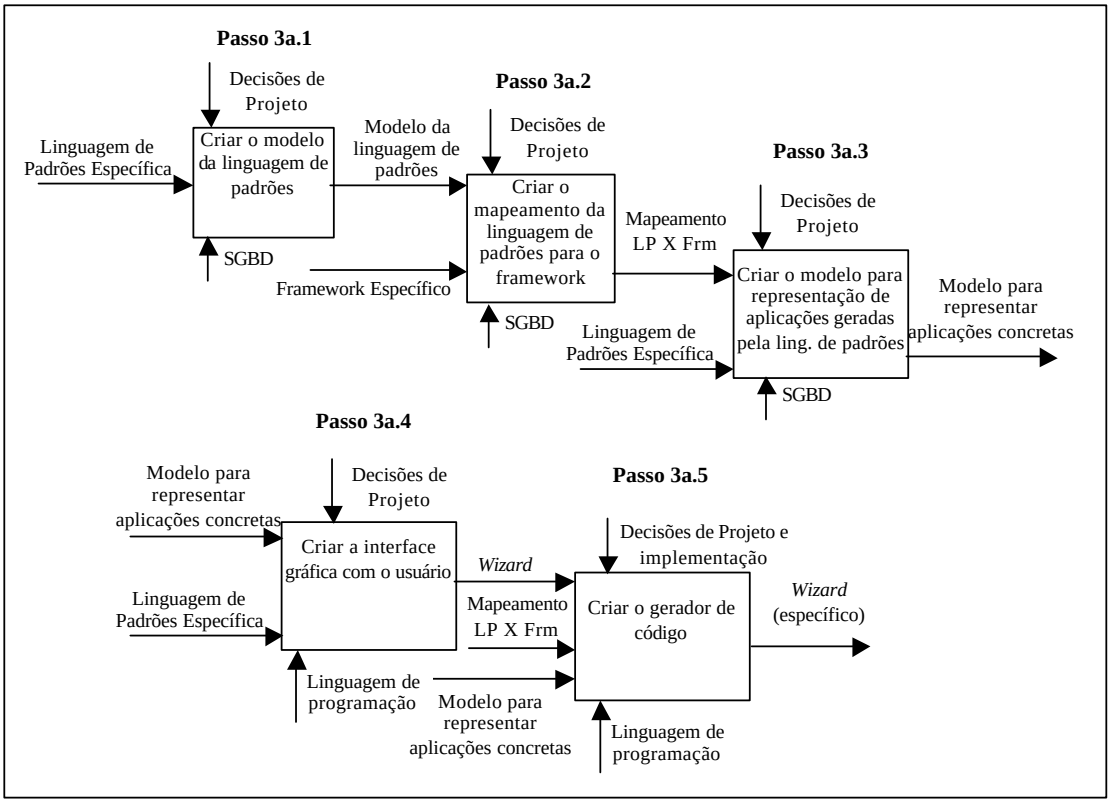


Figura 5 .3: Processo para construção de um *Wizard* específico para instanciação de um framework baseado em uma linguagem de padrões

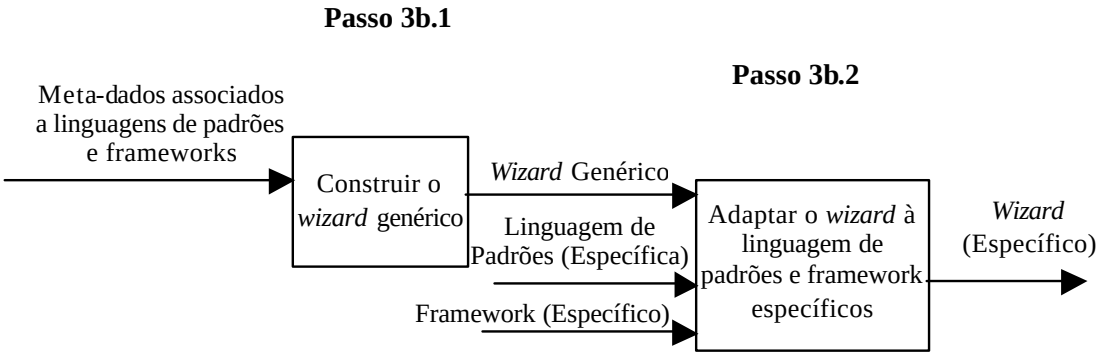


Figura 5 .19: Processo para desenvolvimento de *Wizards* para instanciação de frameworks baseados em linguagens de padrões

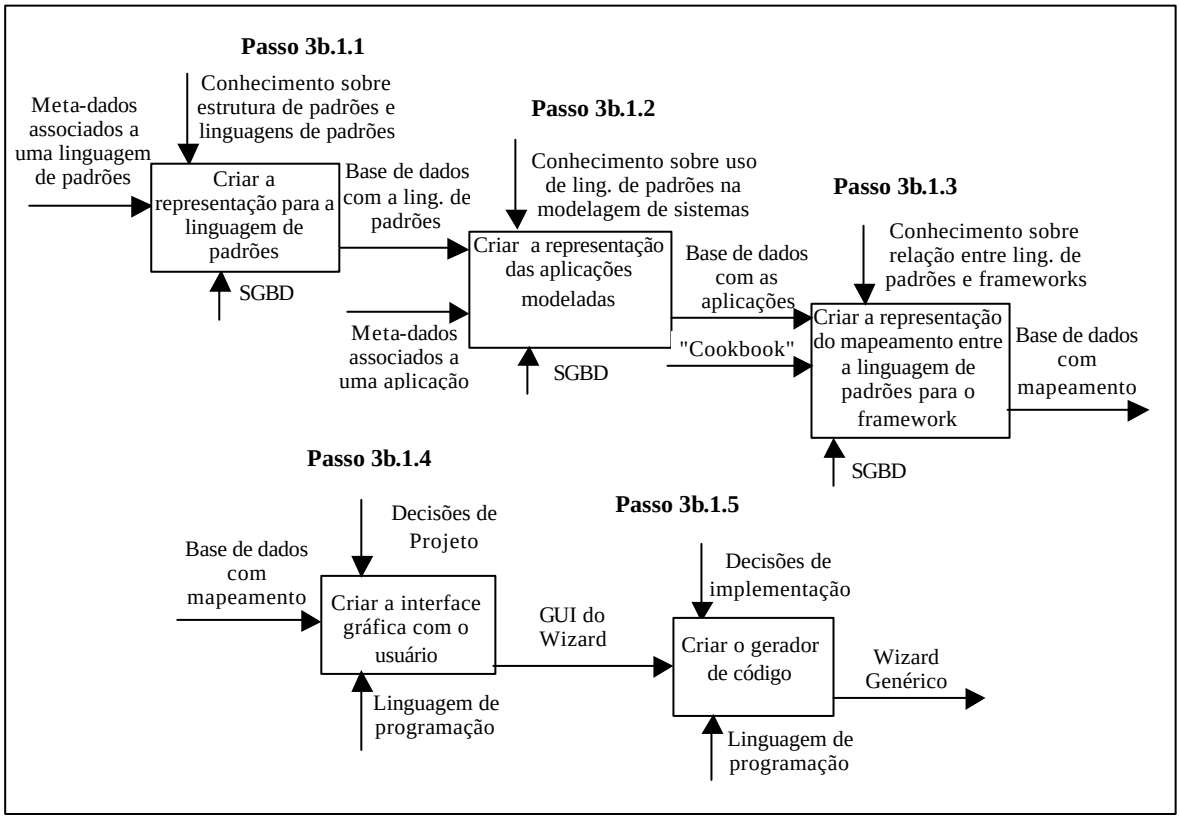


Figura 5 .20: Processo para construção de um *Wizard* genérico para instanciação de frameworks baseados em linguagens de padrões

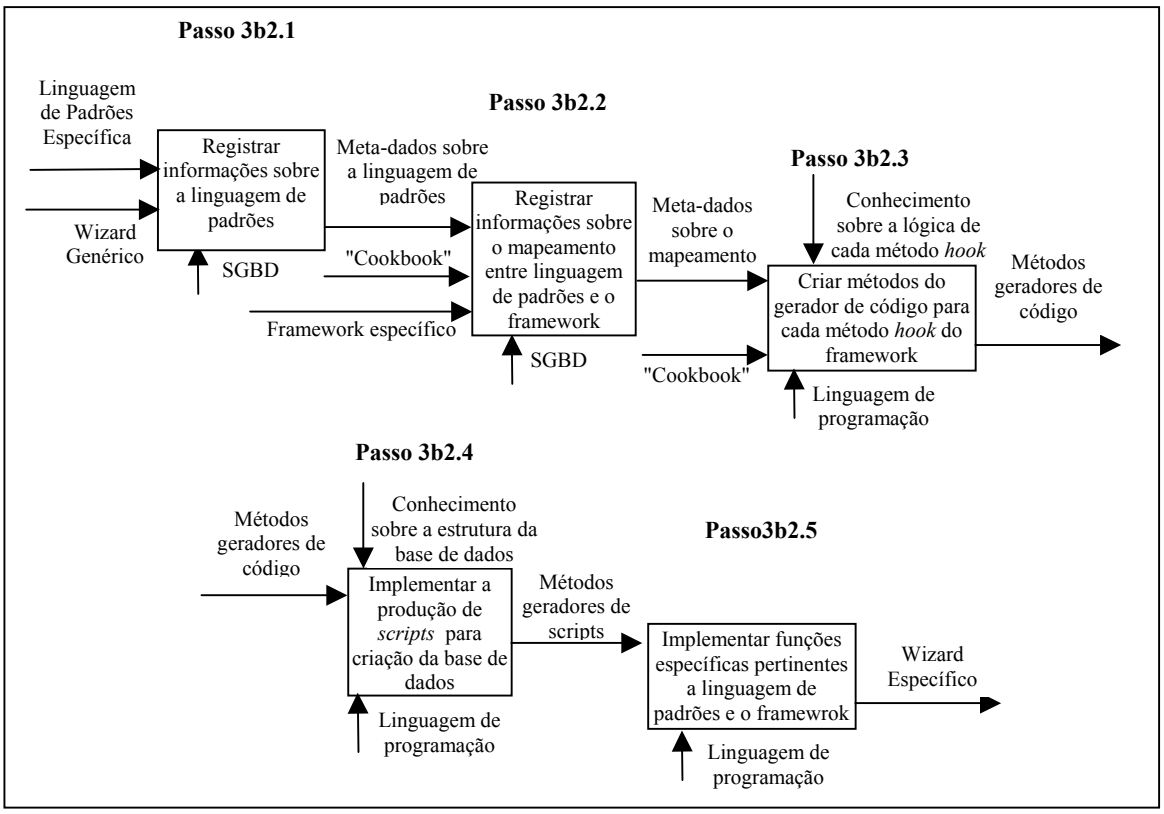


Figura 5 .24: Processo para adaptação do *Wizard* para um framework e linguagem de padrões específicos

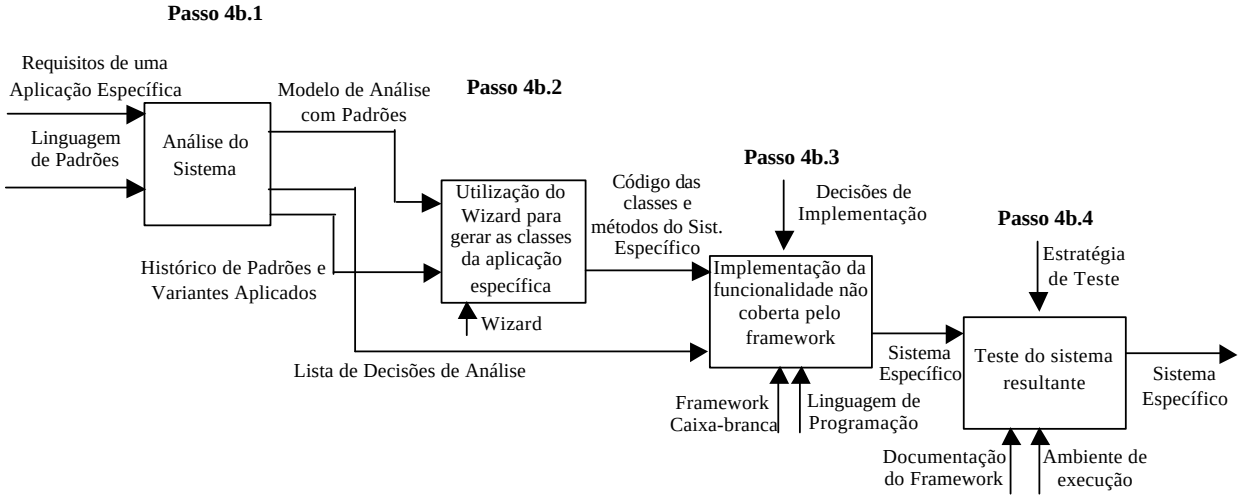


Figura 5 .14: Processo para utilização de um *Wizard* na geração de uma aplicação específica