Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Modularização com Orientação a Aspectos de Frameworks Desenvolvidos com Linguagens de Padrões de Análise"

ANDRÉ LUIZ DE OLIVEIRA

ORIENTADORA: PROFª. DRª. ROSÂNGELA A. DELLOSSO PENTEADO

São Carlos - SP
Setembro/2010

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Modularização com Orientação a Aspectos de Frameworks Desenvolvidos com Linguagens de Padrões de Análise"

ANDRÉ LUIZ DE OLIVEIRA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.
Orientadora: Dra. Rosângela A. Dellosso Penteado.

São Carlos - SP
Setembro/2010

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

O48mo

Oliveira, André Luiz de.
Modularização com orientação a aspectos de frameworks desenvolvidos com linguagens de padrões de análise / André Luiz de Oliveira. -- São Carlos : UFSCar, 2010.
176 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2010.

1. Análise e projeto de sistemas. 2. Linguagem de padrões. 3. Frameworks. 4. Programação orientada a aspectos. 5. Linha de produtos de software. 6. Linha de produtos de frameworks. I. Título.

CDD: 004.21 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Modularização com Orientação a Aspectos de Frameworks Desenvolvidos com Linguagens de Padrões de Análise"

ANDRÉ LUIZ DE OLIVEIRA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Profa. Dra. Rosângela A. Dellosso Penteado
(Orientadora - DC/UFSCar)

Prof. Dr. Marco Túlio de Oliveira Valente
(DCC/UFMG)

Prof. Dr. Valter Vieira de Camargo
(DC/UFSCar)

São Carlos
Setembro/2010

A meus Pais Luiz Carlos e Divina e a minha irmã Thiellen

AGRADECIMENTOS

Agradeço primeiramente a Deus pela vida.

À professora Dra. Rosângela Ap. Dellosso Penteado por ter me escolhido como seu orientando, pela amizade, carinho, atenção e pelos conselhos e dicas que contribuíram enormemente para a realização deste trabalho e para o meu crescimento pessoal e profissional.

Ao professor Dr. Valter Vieira de Camargo pela amizade e pelas valiosas dicas que contribuíram grandemente para a melhoria deste trabalho.

Aos meus pais Luiz Carlos e Divina por me apoiarem em todos os momentos de minha vida.

À minha irmã Thiellen pelo carinho e amizade.

Aos colegas Maykon, Pablo e Rodolfo pela amizade e pela convivência durante o período do mestrado.

Aos colegas de laboratório Matheus, Paulo Afonso, Rafael Durelli, Thiago, Daniel, e Renato.

Aos ex-colegas de laboratório Kamila e Ivan pela amizade.

Ao Guilherme e a sua mãe Rosana por terem me acolhido nos meus primeiros dias em São Carlos.

A todos os meus colegas do mestrado pela amizade e bons momentos que passamos durante esse período.

A todos os funcionários do Departamento de Computação.

A todos que contribuíram direta ou indiretamente para a realização deste trabalho.

Ao CNPq pelo apoio financeiro no primeiro ano do mestrado e à FAPESP pelo apoio financeiro no período restante.

A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original.

Albert Einstein

RESUMO

A linguagem de padrões GRN (Gestão de Recursos de Negócio) fornece um conjunto de padrões em nível de análise que apóiam o desenvolvimento de aplicações que tratam de transações de aluguel, compra, venda e manutenção de um bem ou serviço. GRENJ-OO é um *framework* de aplicação orientado a objetos (OO) construído para apoiar a instanciação de aplicações no domínio da GRN na linguagem Java. O *framework* GRENJ-OO instancia aplicações que incluem em sua arquitetura todas as variabilidades do *framework*. As unidades desse *framework*, que implementam cada padrão da GRN e suas variantes, estão altamente acopladas entre si, em virtude da existência de entrelaçamento e espalhamento de interesses relacionados a cada um desses padrões. Assim, a orientação a aspectos (OA) foi utilizada em cada um dos padrões a fim de minimizar esses problemas e uma nova versão do *framework* foi obtida, denominada GRENJ-OA. A melhoria dos níveis de separação de interesses, a redução do acoplamento, o aumento da coesão e redução do número de linhas de código da maioria dos padrões implementados no GRENJ-OA foram os resultados obtidos após a realização de uma avaliação quantitativa com base em métricas de separação de interesses, acoplamento, coesão e tamanho. A partir da abordagem utilizada na modularização desse *framework*, é introduzido o conceito de Linha de Produtos de *Frameworks*, que consiste em uma linha de produtos na qual seus produtos são *frameworks*, ao invés de aplicações de software. Com a modularização do GRENJ-OO também foi possível extrair um processo, que pode ser aplicado na modularização de *frameworks*. Esse processo tem o objetivo de transformar um *framework* em uma Linha de Produtos de *Frameworks*.

Palavras-chave: linguagem de padrões, *frameworks*, programação orientada a aspectos, linha de produtos de software, linha de produtos de *frameworks*, processo, métricas.

ABSTRACT

GRN (Gestão de Recursos de Negócio – Business Resource Management) pattern language provides a set of patterns in analysis level to support the development of applications which deal with rental, purchase, sale and maintenance transactions of a good or service. GRENJ-OO is an object-oriented (OO) application framework built to support the instantiation of Java applications in the GRN domain. GRENJ-OO instantiates applications that include in their architecture all framework variabilities. The units of this framework, which implement each GRN pattern and their variants, are highly coupled between them, because there are concern tangling and concern scattering related to each one of those patterns. So, the aspect-orientation (OA) techniques were used in each pattern to minimize those problems and a new framework version was obtained, called GRENJ-OA. The improvements of separation of concerns, the coupling reduction, the cohesion increasing and the reduction of the number of lines of code of the majority of the patterns implemented in GRENJ-OA was the result reached after performing a quantitative evaluation based on separation of concerns, coupling, cohesion and size metrics. From the approach used to modularize this framework is introduced the Framework Product Line concept, that consists in a product line which their products are frameworks instead of software applications. From the GRENJ-OO modularization was also possible to extract a process that can be applied to modularize frameworks. This process aims to transform a framework in a Framework Product Line.

Keywords: pattern language, frameworks, aspect-oriented programming, software product line, framework product line, process, metrics.

.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CBC – *Coupling Between Components*

CDC – *Concern Difusion over Components*

CDO – *Concern Difusion over Operations*

CDLOC – *Concern Difusion over Lines of Code*

DIT – *Depth Inheritance Tree*

DH – Decomposição Horizontal

FE – Framework Específico

FODA – *Feature-Oriented Analysis Domain*

FT – *Framework* Transversal

GRENJ - Gestão de REcursos de Negócio em Java

GuiWe – *Graphical User Interface for Web*

GRN – Gestão de Recursos de Negócio

LCOO – *Lack of Cohesion in Operations*

LPS – Linha de Produtos de Software

LPF – Linha de Produtos de *Frameworks*

NOA – *Number of Attributes*

POA – Programação Orientada a Aspectos

OA – Orientação a Aspectos

OO – Orientação a Objetos

SGBD – Sistema Gerenciador de Banco de Dados

UML – *Unified Modeling Language*

WOC - *Weighted Operations per Component*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

As aplicações desenvolvidas com o apoio de *frameworks* geralmente incluem em sua arquitetura final a implementação vazia de vários métodos gancho, que correspondem às variabilidades não relacionadas aos requisitos da aplicação. O desenvolvimento de uma aplicação necessita de somente um subconjunto das variabilidades do *framework.* Porém, aplicações construídas com base em *frameworks* incluem todas as variabilidades do *framework* em sua arquitetura final. Dessa forma, grande quantidade de código desnecessário é incluída na arquitetura final dessas aplicações, prejudicando o projeto das mesmas, provocando problemas de manutenção, evolução e de reúso.

Os problemas relatados acima também ocorrem com *frameworks* transversais, que são os que encapsulam a implementação de interesses relacionados a requisitos não funcionais, como persistência, segurança, *caching*, dentre outros (CAMARGO e MASIERO, 2008). Apesar desse tipo de *framework* encapsular a implementação de um único interesse, dentro de sua implementação, também há variabilidades fazendo com que toda aplicação que o utilize tenha que carregar todas as suas variabilidades, mesmo que não sejam utilizadas. Nesse caso, também há a necessidade de fornecer implementações para métodos gancho mesmo para as variabilidades não utilizadas pela aplicação.

Uma *feature* refere-se à funcionalidade do domínio que tem valor para o cliente (KANG *et al.*, 1990). *Features* podem ser classificadas como obrigatórias, opcionais ou alternativas, e representam as similaridades e variabilidades do software. Os problemas acima mencionados ocorrem porque a implementação das *features* existentes em um *framework*

estão altamente acopladas entre si, reduzindo a coesão das mesmas. O entrelaçamento e o espalhamento de interesses relacionados a essas *features* provocam alto acoplamento e reduz a coesão dessas *features* em *frameworks*, o que faz com que o código relacionado a todas essas seja incluído na arquitetura de aplicações construídas com o apoio de *frameworks*.

Camargo e Masiero (2008) apresentaram uma solução para esse problema que é dividir a arquitetura de um *framework* transversal em várias "*features* reusáveis". Essas *features* podem ser combinadas para gerar um *framework* mais restrito, somente com aquelas necessárias para a aplicação. Dessa forma, essa abordagem permite a construção de uma família de *frameworks transversais*, na qual seus membros (*frameworks*) podem ser gerados a partir da combinação de um conjunto de *features* disponíveis.

Para desacoplar a implementação do núcleo de um *framework* (funcionalidades básicas) de suas funcionalidades ortogonais (interesses transversais relacionados a requisitos funcionais e não funcionais) Zhang e Jacobsen (2004) introduziram o conceito de Decomposição Horizontal (*Horizontal Decomposition*). Esse conceito consiste em refatorar um sistema de software em um núcleo comum e em um conjunto de funcionalidades variáveis. O objetivo da Decomposição Horizontal é suprir as limitações de metodologias convencionais de decomposição arquitetural, aquelas que não tratam da interação de interesses transversais presentes nos requisitos de uma aplicação. A Decomposição Horizontal fornece um conjunto de diretrizes para a utilização de métodos convencionais, como orientação a objetos (OO), para implementar o núcleo de um *framework*, e de orientação a aspectos para implementar suas propriedades ortogonais (transversais). Esse conceito faz a distinção de funcionalidades "aspectuais" e "não-aspectuais", estabelecendo as responsabilidades da POA. Para comprovar a efetividade da Decomposição Horizontal, seus princípios foram aplicados na decomposição do *framework* Prevayler (GODIL e JACOBSEN, 2005) e de um sistema de *Middleware* (ZHANG e JACOBSEN, 2004).

O *framework* GRENJ-OO (DURELLI, 2008), desenvolvido por um processo de reengenharia, instancia aplicações do domínio de gestão de recursos de negócio na linguagem de programação Java. Esse *framework* apresenta os problemas listados nesta seção e é utilizado nesta dissertação como um estudo de caso.

1.2 Motivação

A linguagem de padrões GRN (BRAGA *et al.*, 1999) é composta por quinze padrões em nível de análise para a modelagem de aplicações no domínio de gestão de recursos de negócios, que envolvem transações de aluguel, compra, venda e manutenção de bens ou serviços. GRENJ-OO (DURELLI, 2008) é um *framework* que permite a instanciação de aplicações por meio da escolha dos padrões da GRN implementados nele, que atendem aos requisitos da aplicação. Os padrões da GRN e suas variantes podem ser vistos como *features* desse *framework*, uma vez que cada um representa uma funcionalidade proeminente e distinta de uma aplicação de software que é visível ao usuário.

Os problemas mencionados por Camargo e Masiero (2008) também ocorrem com o *framework* GRENJ-OO, ou seja, as aplicações desenvolvidas com o seu apoio incluem em sua arquitetura todo o *framework*. A implementação desse *framework* tem as *features* altamente acopladas, com entrelaçamento e espalhamento de interesses relacionados a cada padrão da GRN.

A programação orientada a aspectos (POA) proporciona melhor separação de requisitos não funcionais, como persistência, distribuição, segurança, rastreamento e tratamento de exceções, que estão entrelaçados e espalhados pelo código de aplicações de software. Como há entrelaçamento e espalhamento de interesses na implementação dos padrões da GRN no *framework* GRENJ-OO, a orientação a aspectos é candidata a ser utilizada para modularizar a implementação de cada um desses padrões. Dessa forma, pretende-se melhorar a manutenibilidade e a reusabilidade do *framework* GRENJ-OO e de aplicações construídas com o seu apoio.

Uma forma de avaliar os resultados obtidos com o uso da orientação a aspectos para modularizar o *framework* GRENJ-OO é a utilização de métricas de software orientadas a aspectos (OA) (SANT'ANNA *et al.*, 2003). Essas métricas fornecem ao engenheiro de software uma visão dos ganhos obtidos com a modularização do código OO para OA. Experimentos controlados (BASILI *et al*., 1986) possibilitam endossar os resultados obtidos com a aplicação das métricas.

1.3 Objetivos

A partir do contexto apresentado e da motivação em modularizar o *framework* GRENJ-OO de modo que seja mais manutenível, possa ser reusado mais facilmente e que as aplicações geradas sejam de qualidade, esta dissertação tem os seguintes objetivos:

- Avaliar quantitativamente, utilizando de métricas de software orientadas a aspectos, a efetividade da orientação a aspectos na modularização de cada padrão da GRN e suas variantes implementadas no GRENJ-OO;
- Elaboração de diretrizes que auxiliem ao engenheiro de software na modularização de *frameworks* desenvolvidos com linguagem de padrões de análise;
- Reduzir o número de linhas de código fonte das aplicações instanciadas com o GRENJ-OO para que contenham somente o código relacionado às funcionalidades (*features)* necessárias;
- Possibilitar que o GRENJ-OO seja de fácil evolução bem como as aplicações por ele instanciadas;

A metodologia utilizada para alcançar esses objetivos contou com várias etapas: a) o entendimento da GRN e GRENJ-OO, b) busca na literatura por referencial teórico sobre os assuntos envolvidos nesta pesquisa; c) aprimoramento nas técnicas, linguagens e ferramentas orientadas a aspectos; d) a modularização de cada um dos padrões da GRN e testes para garantir que a funcionalidade de cada um permaneça inalterada; e) avaliação da implementação OA no GRENJ-OO, gerando a versão denominada GRENJ-OA; f) seleção e utilização de métricas orientadas a aspectos para avaliar a implementação realizada e compará-la com a implementação OO existente; g) planejamento e condução de experimento controlado para avaliar a utilização/manutenção/reúso do GRENJ-OA.

Os conceitos de orientação a aspectos foram utilizados para modularizar o *framework* GRENJ-OO de modo que sua arquitetura fosse dividida em várias "*features* reusáveis". Desse modo, uma configuração mais específica desse *framework* pode ser gerada incluindo somente as *features* necessárias para atender à funcionalidade da aplicação. Assim, evita-se que *features* não utilizadas sejam incluídas na arquitetura final da aplicação. Com a geração de várias configurações específicas de um *framework* o conceito de Linha de Produtos de *Frameworks* é aqui introduzido e consiste em uma linha de produtos na qual seus membros são *frameworks* ao invés de aplicações de software.

1.4 Organização do Trabalho

Esta dissertação está organizada em oito capítulos. No Capítulo 1 é apresentado o contexto, a motivação, os objetivos do estudo e uma breve descrição da metodologia seguida para alcançá-los. No Capítulo 2 é apresentada a revisão bibliográfica em que são ilustrados os principais conceitos e técnicas utilizadas para que os objetivos fossem alcançados. No Capítulo 3 a linguagem de padrões GRN e o *framework* de aplicação GRENJ-OO que são os principais objetos de estudo desta dissertação são descritos. No Capítulo 4 são apresentados o conceito de Linha de Produtos de *Frameworks* e um processo de modularização de *frameworks*, com orientação a aspectos elaborado a partir da modularização do GRENJ-OO. No Capítulo 5 são fornecidos detalhes da modularização com orientação a aspectos realizada em alguns padrões da linguagem GRN implementados no *framework* GRENJ-OO. No Capítulo 6 são apresentados os resultados da avaliação quantitativa das implementações OO e OA de cada padrão da GRN. No Capítulo 7 são descritos os passos para a realização de um experimento conduzido para avaliar a manutenibilidade e a reusabilidade do *framework* GRENJ-OA. No Capítulo 8 são apresentadas as conclusões desta dissertação, com enfoque nas limitações, contribuições obtidas e sugestões para trabalhos futuros.

Capítulo 2

REVISÃO BIBLIOGRÁFICA

2.1 Considerações Iniciais

No estudo apresentado nesta dissertação foram utilizados conceitos como padrões, linguagens de padrões, orientação a aspectos e métricas, aplicados para modularizar com orientação a aspectos um *framework* de aplicação orientado a objetos desenvolvido com base em linguagem de padrões. Métricas de software foram utilizadas para avaliar a efetividade da orientação a aspectos na modularização desse *framework*.

Neste capítulo são apresentados conceitos utilizados no desenvolvimento desta dissertação juntamente com alguns trabalhos relacionados. Na Seção 2.2 são apresentados os conceitos relacionados à Orientação a Aspectos. Na Seção 2.3 os conceitos de padrões, linguagens de padrões e linha de produtos de software são descritos. Na Seção 2.4 são apresentados os conceitos de métricas. Na Seção 2.5 são apresentados os trabalhos relacionados a esta dissertação. Na Seção 2.6 são expostas as considerações finais.

2.2 Orientação a Aspectos

O paradigma Orientado a Objetos surgiu da crescente necessidade de se desenvolver softwares de qualidade, buscando maiores níveis de reúso e de manutenibilidade, aumentando assim a produtividade do desenvolvimento e o apoio a mudanças de requisitos (MEYER, 1997). Apesar das boas características de modularidade e reúso de componentes que esse paradigma oferece, não há a capacidade de modularizar os códigos de diferentes propósitos

que se encontram espalhados e entrelaçados por diversas classes da aplicação (OSSHER e TARR, 1999), denominados de interesses transversais (LADDAD, 2003). Interesse transversal consiste em uma funcionalidade do sistema cuja implementação está diluída por vários módulos.

Dentre os interesses transversais encontrados em diversas aplicações estão o controle e manipulação de exceções, rastreamento, aplicação de procedimentos de auditoria, implementação de regras e restrições arquiteturais, acesso e persistência de dados (CHAVEZ *et al.*, 2003). Quando esses interesses estão presentes no código dos componentes da aplicação, a modularidade e a manutenibilidade dessas aplicações ficam prejudicadas, reduzindo o potencial de reúso de diversos componentes.

A Programação Orientada a Aspectos (POA) surgiu como uma técnica de programação complementar a de orientação a objetos para tratar, de maneira independente, os interesses transversais que se encontram espalhados e entrelaçados por vários módulos de um sistema, promovendo a reutilização e facilitando a manutenção do software desenvolvido. Esses interesses são encapsulados em unidades de modularização denominadas aspectos (KICZALES *et al*., 1997).

Na Subseção 2.2.1 os conceitos relacionados à programação orientada a aspectos são apresentados e na Subseção 2.2.2 a linguagem AspectJ, que é uma extensão orientada a aspectos da linguagem Java, é descrita.

2.2.1 Programação Orientada a Aspectos (POA)

POA permite que a implementação de um sistema seja separada em requisitos funcionais e não-funcionais (SOARES e BORBA, 2002). Os requisitos funcionais resultam em um conjunto de componentes expressos em uma linguagem de programação atual, como Java (GOSLING *et al.*, 2000). Os requisitos não-funcionais resultam em um conjunto de aspectos (interesses transversais) relacionados às propriedades que afetam o comportamento do sistema. Dessa forma, a POA possibilita o desenvolvimento de programas utilizando tais aspectos, o que inclui isolamento, composição e reúso do código de implementação dos aspectos. POA também propõe um meio para identificar e separar interesses, encapsular esses interesses em módulos coesos e pouco acoplados em vários estágios (requisitos, análise, projeto e implementação), visando manter a rastreabilidade dos interesses de um sistema (CHAVEZ *et al*., 2003).

No desenvolvimento de software, um interesse pode ser visto como um requisito funcional ou não funcional de um sistema (LADDAD, 2003). Os interesses de um sistema de software podem ser classificados em:

- **Interesses de negócio:** fornecem a funcionalidade central de um módulo, por exemplo, procedimento de quitação de uma compra;
- **Interesses em nível de sistema:** captam requisitos periféricos, no nível do sistema e que entrecortam múltiplos módulos, por exemplo, segurança, *logging*, persistência. Esses interesses são os relacionados a interesses transversais, que também podem ser requisitos funcionais, como a implementação de uma regra de negócio que se encontra entrelaçada e espalhada pelo sistema.

2.2.2 A Linguagem AspectJ

AspectJ (KICZALES *et al*., 1997) é uma extensão orientada a aspectos da linguagem Java. A seguir são descritos os seus principais conceitos.

***Join Point* (Ponto de Junção):** é um ponto bem definido no fluxo de execução de um programa (KICZALES *et al.,* 1997). Exemplos de *join points* são chamadas e execuções de métodos, execuções de construtores, referências a campos (set e get), manipulação de exceções, inicializações estáticas e combinações desses utilizando os operadores lógicos ! (not), && (and) e | | (or) (LADDAD, 2003). Em AspectJ, *join points* são lugares onde se pode injetar ações de *crosscutting* (entrecortantes).

***Pointcut* (Ponto de Corte):** é uma construção que permite a seleção de um ou mais *join points* e a obtenção do conjunto dos valores do contexto de execução desses *join points* (KICZALES *et al.*, 2001). *Pointcuts* são formados pela composição de *join points* através dos operadores lógicos ! (not), && (and) e | | (or).

***Advice* (Adendo)**: é um mecanismo similar a um método, que deve ser executado quando determinado *join point* é alcançado (KICZALES *et al.,* 2001). *Advices* podem ser executados antes (*before*), depois (*after*), ou durante (*around*) a chamada ou a execução de um *join point*.

Aspecto: é a principal construção da linguagem AspectJ (KICZALES *et al*., 2001). Aspectos são unidades modulares de implementação de interesses transversais. Um aspecto pode declarar atributos e métodos, e pode estender outro aspecto por definir comportamento concreto para algumas declarações abstratas.

Aspectos podem ser utilizados para afetar a estrutura estática de programas Java, pelo uso do mecanismo de *static crosscutting* presente na linguagem AspectJ (KICZALES *et al.*, 2001). Esse mecanismo permite que novos métodos e campos sejam introduzidos em uma classe existente, a conversão de *checked* em *unchecked exceptions* e permite mudanças na hierarquia de classes (LADDAD, 2003). Os aspectos também afetam a estrutura dinâmica de um sistema por modificar o modo de execução de um programa (KICZALES *et al.*, 2001).

2.3 Padrões, Linguagens de Padrões e Linha de Produtos

No desenvolvimento de software há necessidade de reutilizar não somente código como também outros artefatos em nível mais alto de abstração, como processos, modelos de análise e projeto, casos de teste, dentre outros. Dentre os métodos e tecnologias de reúso existentes podem-se citar os padrões de software, linguagens de padrões e linhas de produtos de software. Padrões fornecem soluções para problemas recorrentes durante o desenvolvimento de software (GAMMA *et al.*, 1995). Linguagens de Padrões representam o conhecimento de um domínio de aplicação específico, que consiste na experiência obtida por projetistas em resolver uma classe similar de problemas nesse domínio (BRUGALI e SYCARA, 2000). Linha de Produtos de Software (LPS) provê o reúso identificando as características (*features*) comuns entre vários sistemas de software de um domínio específico (GRISS, 2000).

Na Subseção 2.3.1 são descritos os conceitos de padrões e linguagem de padrões. Na Subseção 2.3.2 os conceitos relacionados à linha produtos de software são abordados. Na Subseção 2.3.3 são apresentados conceitos de modelagem de *features*.

2.3.1 Padrões e Linguagens de Padrões

Engenheiros de software geralmente são sobrecarregados com decisões de projeto durante o desenvolvimento de software. A aquisição de experiência em projeto orientado a objetos leva tempo, no qual classes, interfaces, hierarquias de herança e relacionamentos de associação entre classes devem ser definidos (GAMMA, *et al.*, 1995). Com a aquisição do conhecimento para solucionar problemas que ocorrem no desenvolvimento de software, os desenvolvedores tendem a reutilizar soluções de sucesso do passado para resolver problemas

de projeto atuais (BECK *et al*., 1996). Dessa forma, o conhecimento adquirido estava restrito à experiência pessoal do desenvolvedor; posteriormente esse conhecimento pôde ser documentado na forma de padrões, permitindo sua reutilização por outros desenvolvedores que se depararem com esses problemas.

Os padrões apresentam soluções para problemas recorrentes em determinados contextos (GAMMA *et al*., 1995). Em desenvolvimento de software os padrões estão presentes em diversos níveis de abstração: análise, projeto, implementação, dentre outros (SHALLOWAY e TROTT, 2001).

Além da classificação por níveis de abstração, os padrões de projeto podem ser classificados em três categorias com base em dois critérios: propósito e escopo, como mostrado na Tabela 2.1 (GAMMA *et al.*, 1995). O critério propósito reflete o que um padrão faz e o critério escopo especifica se o padrão se aplica primariamente a classes ou objetos.

Tabela 2.1 – Classificação dos padrões de projeto (GAMMA *et al.*, 1995).

Classificação			
Propósito		**Escopo**	
Criação	Padrões relacionados à criação de objetos.	**Classe**	Tratam de relacionamentos entre classes e suas subclasses, estabelecidos por meio de herança, definidos em tempo de compilação.
Estrutural	Padrões que tratam da composição de classes ou objetos.		
Comportamental	Padrões que descrevem como classes ou objetos interagem entre si para executar uma tarefa.	**Objeto**	Tratam de relacionamentos entre objetos que podem ser alterados em tempo de execução.

O conjunto de padrões destinados a resolução de uma classe de problemas de um domínio específico constitui uma linguagem de padrões. Uma linguagem de padrões representa o conhecimento de um domínio de aplicação específico a partir da experiência dos desenvolvedores em resolver problemas similares (BRUGALI e SYCARA, 2000).

Linguagens de padrões podem ser estruturadas em árvores ou em grafos, em que cada padrão leva a uma série de outros padrões pertencentes à linguagem (BRUGALI e SYCARA, 2000). As linguagens de padrões podem ser empregadas como guias durante o desenvolvimento de aplicações de um mesmo domínio (BRAGA e MASIERO, 2002). Há várias linguagens de padrões disponíveis. Nesta dissertação a linguagem de padrões de Gestão de Recursos de Negócio (GRN) (BRAGA *et al.*, 1999) foi utilizada e está descrita no Capítulo 3.

2.3.2 Linha de Produtos de Software

Uma Linha de Produtos de Software (LPS) consiste em um conjunto de sistemas de software que compartilham um conjunto de características (*features*) comuns que satisfazem às necessidades específicas de um domínio e que é desenvolvida a partir de um conjunto de ativos principais (*core assets*) da linha (SEI/CMU, 2010). Uma *feature* consiste em uma característica proeminente e distinta de um sistema de software que é visível ao usuário (KANG *et al.*, 1990).

As partes comuns, denominadas similaridades, a todos os sistemas do domínio são reutilizadas e as características específicas, denominadas variabilidades, são desenvolvidas como componentes específicos. Para John e Muthig (2002) a engenharia de linha de produtos consiste em uma abordagem para reúso que fornece métodos para planejar, controlar e melhorar a infra-estrutura de reúso para o desenvolvimento de famílias de produtos de um determinado domínio.

Para o desenvolvimento de uma LPS é essencial primeiramente determinar os produtos que devem ser incluídos, que consiste em um processo conhecido como escopo (HUNT e MCGREGOR, 2006). Existem três diferentes formas de escopo, sendo que cada uma delas consiste em uma maneira diferente de planejamento de reúso, que é aplicável a diferentes estágios do ciclo de vida. Esses tipos de escopo correspondem a: a) escopo do portfólio do produto, que objetiva identificar os produtos a serem desenvolvidos e as *features* que eles devem prover; b) escopo de domínio, que consiste na delimitação dos limites do domínio relevante para a linha de produtos; e c) escopo do *asset*, que possui o objetivo de identificar os componentes que devem ser desenvolvidos de maneira reutilizável (SCHMID, 2002).

2.3.3 Modelo de Features

Um modelo de *features* provê a base para o desenvolvimento, parametrização e configuração dos *core assets* de uma linha de produtos de software (KANG *et al.*, 2002). A modelagem de *features* apóia o reúso de software desde a etapa inicial do processo de desenvolvimento (PACIOS, 2006).

O modelo de *features* é uma representação hierárquica de um conjunto de características comuns e variáveis de um domínio específico e suas relações de dependência.

No contexto de LPS, o modelo de *features* representa a própria linha de produtos. Uma *feature* pode ser (KANG *et al.*, 1990):

a) Obrigatória: presente em todos os membros da linha de produtos;

b) Opcional: pode ou não estar presente em um membro da linha de produtos;

c) Alternativa: composta por um conjunto de *features* das quais se escolhe uma ou mais *features*.

Existem várias notações para modelagem de *features* tais como: a) a de Gomaa (2004), que consiste em uma notação baseada em UML (BOOCH *et al.*, 2005) e que integra a modelagem de *features* com visões de modelagem da UML; b) a de Kang *et al.* (1990), que é parte da abordagem *Feature Oriented Domain Analysis* (FODA); e c) a *Feature Description Language* (VAN DEURSEN e KLINT, 2001) que é uma linguagem textual para descrever *features*. Na Figura 2.1 é apresentado um modelo de *features* com a representação de cada um dos tipos de *features* existentes e usa a notação sugerida por Kang *et al.* (1990).

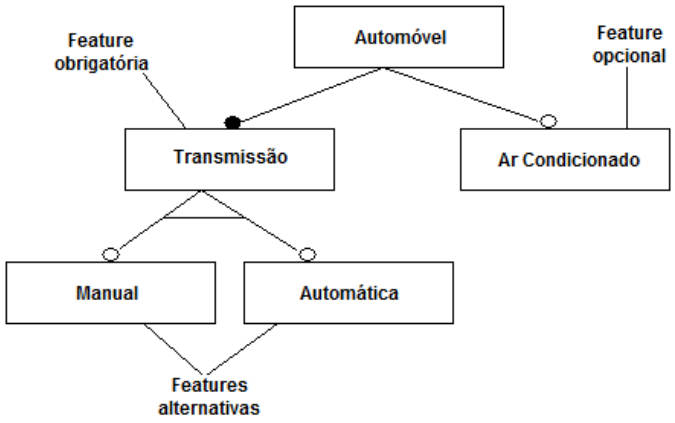

Figura 2.1. Modelo de *features*, adaptado de Kang *et al.* (1990).

O modelo de *features* é representado por uma árvore onde cada nó é uma *feature* (PACIOS, 2006) e as arestas são interpretadas junto com o tipo do nó e indicam a configurabilidade do nó. Caso o arco seja vazio, deve-se escolher apenas uma das *features* alternativas, se o arco for preenchido é permitido escolher mais de uma *feature* alternativa (VAN DEURSEN e KLINT, 2001). A raiz da árvore é comumente utilizada para especificar o que está sendo modelado, como ilustrado na Figura 2.1.

2.4 Métricas de Software

Métricas de software envolvem algum grau de medição, como estimativas de custo e de esforço, medidas e modelos de produtividade, coleta de dados, modelos de qualidade, medidas e modelos de confiabilidade, modelos de avaliação de *performance*, métricas estruturais e de complexidade, avaliação de modelos de capacidade-maturidade (CMM - *Capability Maturity Model*) e gerenciamento por ferramentas e métodos de avaliação (FENTON e PFLEGEER, 1997).

As métricas podem ser classificadas em três tipos: a) de processo – medidas de coleções de atividades de software relacionadas; b) de produto – mede a qualidade de qualquer artefato ou documento que resulta de uma atividade do processo; e c) de recurso – medida relacionada às entidades requeridas por uma atividade do processo. Cada tipo de métrica possui atributos internos e externos (FENTON e PFLEGEER, 1997). Atributos internos podem ser medidos em termos do próprio produto, processo ou recurso, ou seja, não dependem de outros fatores para serem medidos, pois é independente de comportamento. Dentre os exemplos de atributos internos citam-se as métricas de separação de interesses, acoplamento, coesão e tamanho. O número de linhas de código fonte (LOC) é um exemplo de atributo interno. Atributos externos podem ser medidos somente com respeito em como o produto, o processo ou o recurso se relaciona com seu ambiente. Exemplos de atributos externos são a manutenibilidade e a reusabilidade.

As métricas de software utilizadas nesta dissertação são as que estão associadas à medição da estrutura do software em níveis de projeto e implementação (SANT'ANNA *et al.*, 2003). Esse conjunto de métricas orientadas a aspectos atende aos seguintes requisitos: (i) separação de interesses, acoplamento, coesão e tamanho; (ii) estendem as métricas tradicionais de OO (CHIDAMBER e KEMERER, 1994). Na Tabela 2.2 é apresentada a descrição de cada uma dessas métricas.

Tabela 2.2 – Métricas, adaptado de Sant'Anna *et al.* (2003).

Atributo	Métrica	Definição
Separação de Interesses	Difusão do interesses por componentes (CDC)	Conta o número de classes e aspectos cujo propósito principal é contribuir para a implementação do interesse e o número de outras classes e aspectos que os acessam.
	Difusão do Interesse por operações (CDO)	Conta o número de métodos e *advices* cujo propósito principal é contribuir para a implementação do interesse e o número de outros métodos e *advices* que os acessam.
	Difusão do interesse por linhas de código (CDLOC)	Conta o número de pontos de transição para cada interesse por meio de linhas de código. Pontos de transição são pontos no código onde há uma troca de interesses.
Acoplamento	Acoplamento entre componentes (CBC)	Conta o número de outras classes e aspectos para o qual uma classe ou aspecto está acoplada.
	Profundidade da árvore de herança (DIT)	Conta a profundidade da hierarquia de herança em que uma classe ou aspecto é declarado.
Coesão	Falta de coesão nas operações (LCOO)	Mede a falta de coesão de uma classe ou aspectos em termos da quantidade de pares de métodos e *advices* que acessam a mesma variável de instância.
Tamanho	Linhas de código (LOC)	Conta o número de linhas de código.
	Número de atributos (NOA)	Conta o número de atributos de cada classe ou aspecto.
	Peso de operações por componente (WOC)	Conta o número de métodos e *advices* de cada classe ou aspecto e o número de seus parâmetros.

2.5 Trabalhos Relacionados

Nesta seção são apresentados trabalhos recentes que contribuíram para a elaboração deste. No Capitulo 8 são apresentadas tabelas comparativas entre os conceitos e resultados apresentados nesses trabalhos e o que foi desenvolvido nesta dissertação. Esses trabalhos incluem técnicas utilizadas na manutenção de *frameworks*, linha produtos de software, o uso da orientação a aspectos em LPS e trabalhos que envolvem a avaliação quantitativa de soluções OA.

Com relação à manutenção de *frameworks*, Kellens *et al.* (2008) relataram suas experiências em utilizar OA na implementação das regras de negócio em um sistema OO de larga escala para comprovar que a OA melhora a modularidade e a extensibilidade da implementação das regras de negócio. No estudo de caso utilizado, a implementação das regras de negócio estava entrelaçada com operações de persistência e espalhadas pelo código de classes "*Manager*" desse sistema. Além disso, encontraram duplicação de código relacionado às regras de negócio. A solução adotada foi a separação das regras de negócio do código de persistência e a refatoração delas para aspectos.

Zhang e Jacobsen (2004) modularizaram um *framework* de *middleware* com OA. Introduziram o conceito de Decomposição Horizontal (DH), para suprir as limitações de metodologias de decomposição arquitetural convencionais, como a dificuldade em tratar da

interação de interesses transversais presentes nos requisitos de uma aplicação. O conceito de Decomposição Horizontal, que consiste em refatorar um sistema de software em um núcleo comum e em um conjunto de funcionalidades variáveis, possui como base cinco princípios que sintetizam experiências do uso da POA no *refactoring* de *frameworks*:

1) Reconhecer a relatividade de aspectos (*Recoginize the relativity of aspects*): definido o núcleo de um *framework*, todas as demais *features* devem entrecortar (*crosscutts*) esse núcleo, adicionando suas funcionalidades;

2) Estabelecer a coerente decomposição do núcleo (*Establish the coherent core decomposition*): consiste em estabelecer as funcionalidades que constituem o núcleo de um *framework*;

3) Definir a semântica de um aspecto de acordo com a decomposição do núcleo (*Define the semantics of an aspect according to the core decomposition*): considera que o núcleo é utilizado como referência para implementar funcionalidades ortogonais com OA;

4) Manter uma arquitetura dirigida a classe (*Mantain a class-directional architecture*): estabelece que o núcleo do *framework* não deve ter conhecimento da implementação dos aspectos;

5) Aplicar *refactoring* incremental (*Apply incremental refactoring):* a decomposição com OA deve ser feita de maneira incremental.

Godil e Jacobsen (2005) refatoraram o *framework* Prevayler (Prevayler, 2010), um *framework* de persistência, utilizando os princípios de *refactoring* da Decomposição Horizontal (ZHANG e JACOBSEN, 2004). A refatoração do *framework* Prevayler teve o objetivo de avaliar a aplicabilidade dos princípios da Decomposição Horizontal em domínios diferentes dos quais esse conceito foi criado. Nessa avaliação foram utilizadas as métricas de separação de interesses, acoplamento, coesão e tamanho de Sant'Anna *et al*. (2003). Como resultado obteve-se que o uso dos princípios de DH e da OA para refatorar o Prevayler reduziu o nível de separação de interesses de seu núcleo, ou seja, elevou o número de componentes e de operações que implementam o seu núcleo. Também foi observada a redução do acoplamento, do número de linhas de código e do tamanho do vocabulário e o aumento da coesão de seu núcleo.

Frameworks transversais (CAMARGO, 2006) são *frameworks* que encapsulam a implementação de um único interesse relacionado a requisitos não-funcionais como: persistência, segurança e controle de acesso. Nesse contexto Camargo e Masiero (2008) apresentaram uma abordagem para evitar que aplicações que utilizem *frameworks* desse tipo

incluam em sua arquitetura final o código relacionado a todas suas *features*, quando somente um subconjunto dessas é necessário. A proposta é dividir a arquitetura de um *framework* transversal em várias "*features* reusáveis", que podem ser combinadas para gerar um *framework* mais restrito, somente com as *features* necessárias para a aplicação, evitando que *features* não necessárias à aplicação sejam incluídas em sua arquitetura final. Camargo e Masiero (2008) dão o enfoque na construção de uma família de *frameworks transversais*, na qual seus membros (*frameworks*) podem ser gerados a partir da combinação de um conjunto de *features* disponíveis.

Considerando a avaliação quantitativa, vários estudos foram conduzidos para verificar a efetividade do uso da orientação a aspectos na implementação de padrões de projeto, linha de produtos de software e arquiteturas de software. Esses estudos estão relacionados a esta dissertação em conseqüência da realização de uma avaliação quantitativa para verificar a efetividade da orientação a aspectos na modularização de padrões em nível de análise.

Um estudo quantitativo para avaliar a estabilidade do projeto na evolução de linhas de produtos com POA é apresentado em Figueiredo *et al.* (2008). A análise da evolução de linhas de produtos em termos de modularidade, propagação de mudanças e dependência entre *features* foram consideradas. Métricas foram utilizadas para comparar a implementação das variabilidades de linhas de produtos utilizando OA e Compilação Condicional. O resultado foi satisfatório ao utilizar OA na modularização de *features* opcionais e alternativas, mas não para *features* obrigatórias, pois compartilham código com outras *features*. Isso ocorre, pois qualquer alteração no código de alguma *feature* obrigatória requer a alteração no código de todas as *features* dependentes.

Oliveira *et al.* (2008) apresentaram um estudo preliminar quantitativo com base em métricas de separação de interesses, acoplamento, coesão e tamanho (SANT'ANNA *et al*., 2003) para avaliar o uso de abstrações OA na modularização do padrão *Data Access Object* (SUN MICROSYSTEMS, 2002), que é um padrão para a implementação do interesse de persistência. Posteriormente, Oliveira *et al.* (2008a) refinaram o estudo anterior, no qual foi apresentada uma avaliação quantitativa com base em métricas para verificar a efetividade de OA na implementação de três estratégias do padrão *Data Access Object*. Como resultado foi verificado que o uso de OA melhorou a separação de interesses para todas as implementações desse padrão e reduziu o acoplamento e aumentou a coesão de duas implementações desse padrão.

Oliveira *et al.* (2008b) propuseram uma arquitetura de software para o desenvolvimento de aplicações de ambiente multiplataforma e apresentaram um estudo comparativo com base nas métricas de Sant'Anna *et al.* (2003) para validar essa arquitetura.

Cacho *et al.* (2006) desenvolveram um estudo quantitativo que investiga o uso da OA na composição de padrões de projeto, com o objetivo de verificar como a POA trata da modularização de interesses específicos do padrão na presença de interações entre padrões. Neste estudo foram analisadas a composição de 62 pares de padrões GoF (GAMMA *et al.*, 1995) presentes em três sistemas implementados nas linguagens Java e AspectJ. Essa análise também incluiu composições envolvendo mais de dois padrões.

Garcia *et al.* (2005) apresentaram um estudo quantitativo com base em métricas de separação de interesses, acoplamento, coesão e tamanho (SANT'ANNA *et al.*, 2003) para avaliar as implementações OA dos padrões de projeto GoF (GAMMA *et al.*, 1995) desenvolvidas no estudo conduzido por Hannemann e Kiczales (2002). Foi verificado que o uso da OA na modularização desses padrões melhorou a separação de interesses relacionados a cada padrão, apesar de apenas quatro implementações OA terem exibido significativas melhorias de reúso.

2.6 Considerações Finais

Esta dissertação tem como tema principal a revitalização do *framework* de aplicação GRENJ-OO, desenvolvido com o paradigma orientado a objetos. Esse *framework* apresenta entrelaçamento e espalhamento de interesses em seu código fonte. Para resolver esse problema, a orientação a aspectos foi utilizada.

Frameworks necessitam de manutenção e diversos conceitos podem ser utilizados para realizá-la. Foram apresentados estudos que envolvem o uso de orientação aspectos na manutenção de *frameworks*, como os de Zhang e Jacobsen (2004) e Godil e Jacobsen (2005), que tratam do uso do conceito de Decomposição Horizontal para modularizar *frameworks*.

Orientação a aspectos pode ser utilizada para implementar linhas de produtos de software, como mostrado por Figueiredo *et al.* (2008). Camargo e Masiero (2008) apresentaram a utilização de orientação a aspectos para a construção de famílias de *frameworks* transversais.

A avaliação quantitativa pode ser realizada com o uso de métricas orientadas a aspectos (SANT'ANNA *et al.*, 2003) como são mostradas em Cacho *et al.* (2006) e Garcia *et al.* (2005).

No próximo capítulo a linguagem de padrões GRN, que foi uma das fontes de informação utilizadas no desenvolvimento deste trabalho, e o *framework* GRENJ, que é o enfoque deste estudo, são apresentados.

Capítulo 3

A LINGUAGEM DE PADRÕES GRN E O FRAMEWORK GRENJ

3.1 Considerações Iniciais

Linguagens de padrões fornecem um conjunto de soluções organizadas de maneira hierárquica para resolver uma classe de problemas recorrentes em um domínio de aplicação específico (BRAGA *et al*., 1999). Essas linguagens atuam como uma técnica de reúso de software, pois permitem o encapsulamento do conhecimento de um domínio de aplicação específico, de modo que seja facilmente reutilizado. Linguagens de padrões podem ser utilizadas para documentar *frameworks*, que podem ser utilizados para apoiar o reúso do conhecimento de domínio expresso em linguagens de padrões (BRAGA e MASIERO, 2002); e podem ser utilizados em conjunto para aumentar o reúso de software.

Um exemplo de linguagem de padrões é a de Gestão de Recursos de Negócio (GRN) (BRAGA *et al.*, 1999), que fornece um conjunto de padrões em nível de análise para o desenvolvimento de aplicações no domínio de transações de aluguel, compra, venda e manutenção de bens ou serviços. GRENJ (DURELLI, 2008) é um *framework* de aplicação orientado a objetos construído com base na linguagem de padrões GRN, possibilitando a instanciação de aplicações no domínio dessa linguagem.

Neste capítulo são apresentados os conceitos relacionados à linguagem de padrões GRN e ao *framework* de aplicação GRENJ, utilizados no desenvolvimento desta dissertação. Na Seção 3.2 a linguagem de padrões GRN é apresentada. Na Seção 3.3 são comentadas as diferentes versões existentes para o *framework* GRENJ e as características de cada uma delas. Na Seção 3.4 são apresentadas as considerações finais.

3.2 A Linguagem de Padrões GRN

A linguagem de padrões GRN (Gestão de Recursos de Negócios) (BRAGA *et al.*, 1999) é composta de quinze padrões de análise que podem ser utilizados no desenvolvimento de aplicações que necessitam registrar transações de aluguel (bem ou serviço), de comercialização (transferência de propriedade) e de manutenção (reparo ou conservação de um produto).

Os padrões que compõem a GRN estão agrupados de acordo com seus propósitos e estão denotados por meio da linguagem de modelagem UML (BOOCH *et al.*, 2005). Alguns símbolos são incluídos como prefixos nos nomes das operações para denotar eventos de entrada (símbolo ?), eventos de saída (símbolo !) ou métodos enviados e coleções de objetos (símbolo *). Na Figura 3.1 é ilustrada a ordem de precedência na qual cada um desses padrões deve ser aplicado.

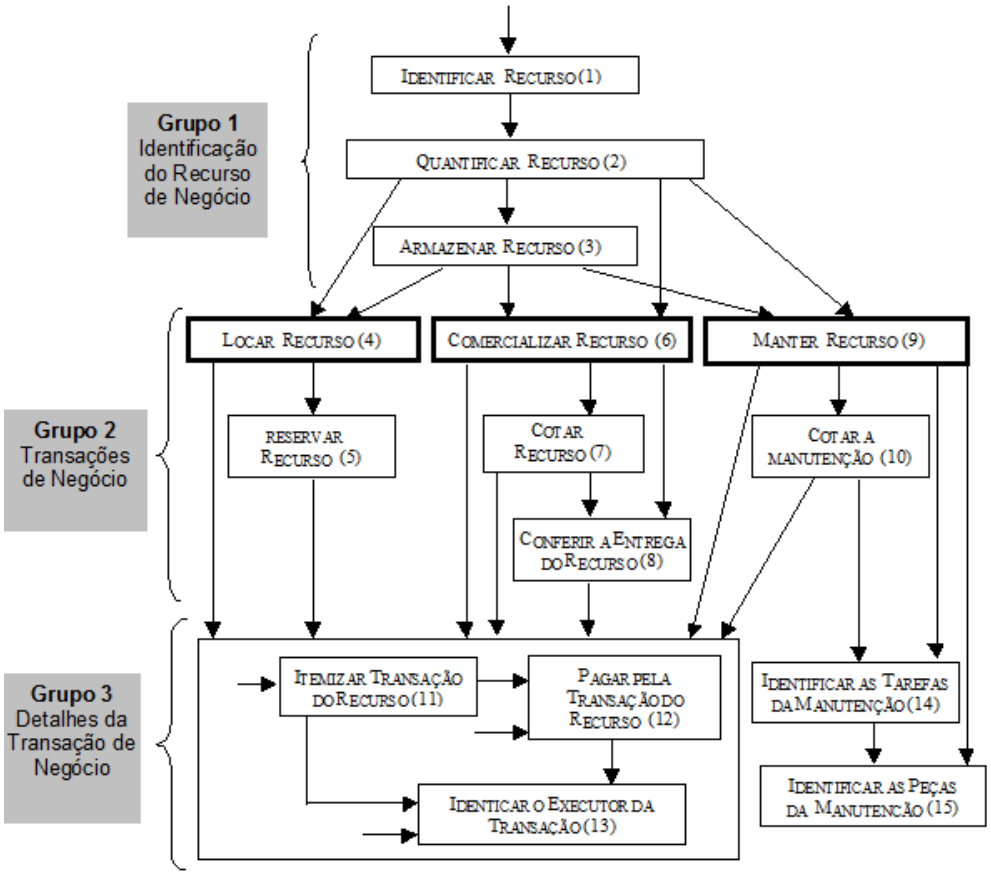

Figura 3.1. Visão geral da linguagem de padrões GRN (BRAGA *et al.*, 1999).

Na Tabela 3.1 são apresentados os padrões da GRN, o grupo ao qual pertencem e uma breve descrição.

Tabela 3.1 – Padrões da linguagem GRN.

Grupo	Nome do Padrão (número do padrão)	Descrição
Grupo 1: Identificação do Recurso de Negócio	IDENTIFICAR RECURSO (1)	Trata da identificação dos recursos de negócio envolvidos em transações processadas pelo sistema.
	QUANTIFICAR RECURSO (2)	Fornece um conjunto de soluções para a quantificação do recurso de negócio.
	ARMAZENAR RECURSO (3)	Controla as formas de armazenamento do recurso de negócio.
Grupo 2: Transações de Negócio	LOCAR RECURSO (4)	Trata do gerenciamento de aluguéis do recurso de negócio.
	RESERVAR RECURSO (5)	Gerencia as reservas relacionadas ao recurso de negócio antes da locação efetiva.
	COMERCIALIZAR RECURSO (6)	Trata do gerenciamento de transações de compra e venda de recursos.
	COTAR RECURSO (7)	Gerencia as cotações realizadas sobre o recurso de negócio antes de sua comercialização efetiva.
	CONFERIR A ENTREGA DO RECURSO (8)	Apresenta um mecanismo para conferir a entrega do recurso de negócio em relação à sua comercialização.
	MANTER RECURSO (9)	Efetua o gerenciamento de manutenções relacionadas ao recurso de negócio.
	COTAR A MANUTENÇÃO (10)	Trata do gerenciamento das cotações associadas à manutenção do recurso de negócio.
Grupo 3: Detalhes da Transação de Negócio	ITEMIZAR TRANSAÇÃO DO RECURSO (11)	Fornece uma solução para gerenciar vários recursos de negócio em uma única transação.
	PAGAR PELA TRANSAÇÃO DO RECURSO (12)	Gerencia os pagamentos associados às transações de negócio.
	IDENTIFICAR O EXECUTOR DA TRANSAÇÃO (13)	Fornece uma solução para identificar a pessoa ou entidade responsável pela execução da transação de negócio.
	IDENTIFICAR TAREFAS DE MANUTENÇÃO (14)	Trata da identificação das tarefas envolvidas em uma manutenção ou cotação da manutenção associada a um recurso de negócio.
	IDENTIFICAR PEÇAS DE MANUTENÇÃO (15)	Trata da identificação das peças utilizadas em uma manutenção ou cotação da manutenção associada a um recurso de negócio.

A fim de exemplificar, alguns dos padrões da linguagem GRN são detalhados a seguir. A funcionalidade de cada padrão é representada por um modelo de classes, o que facilita o seu uso por desenvolvedores para a criação de aplicações no domínio de Gestão de Recursos de Negócio. O detalhamento dos demais padrões pode ser encontrado em Braga *et al.* (1999).

3.2.1 Identificar Recurso

O padrão IDENTIFICAR RECURSO propõe uma solução para representar os recursos de negócio envolvidos nas transações processadas pelo sistema. O modelo de classes para representá-lo é o exibido na Figura 3.2.

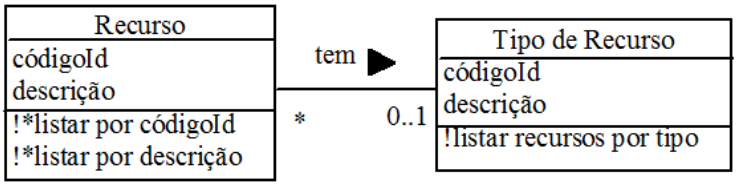

Figura 3.2. Modelo de classes do padrão IDENTIFICAR RECURSO (BRAGA *et al.*, 1999).

3.2.2 Quantificar Recurso

O padrão QUANTIFICAR RECURSO é útil em situações nas quais se necessita ter o controle de instâncias específicas do recurso, em aplicações em que os recursos são tratados em quantidades específicas ou em lotes e em aplicações em que o recurso é único. Esse padrão fornece quatro subpadrões, sendo que cada um apresenta soluções para esse problema, dependendo da forma de quantificação:

a) RECURSO SIMPLES: aplicável quando o recurso é único;

b) RECURSO MENSURÁVEL: lida com situações nas quais o recurso é tratado em quantidades específicas;

c) RECURSO INSTANCIÁVEL: aplicável quando é necessário distinguir entre instâncias de recursos;

d) RECURSO EM LOTES: lida com situações em que o recurso é tratado como lotes.

Na Figura 3.3 é apresentado o modelo de classes de cada uma das estratégias de quantificação do recurso de negócio fornecida pelo padrão QUANTIFICAR RECURSO.

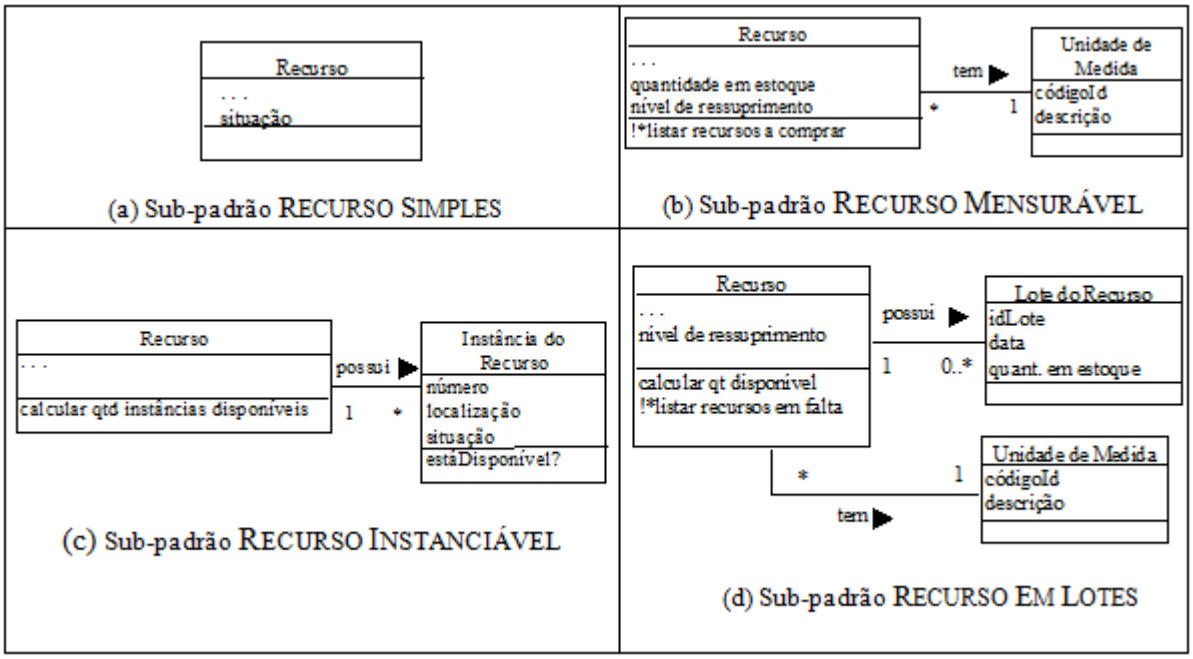

Figura 3.3. Modelo de classes do padrão QUANTIFICAR RECURSO (BRAGA *et al.*, 1999).

3.2.3 Comercializar Recurso

O padrão COMERCIALIZAR RECURSO é aplicável quando se necessita gerenciar a comercialização de recursos em uma aplicação, como ilustrado na Figura 3.4 (BRAGA *et al.*, 1999). A comercialização abrange transações de compra e venda de recursos de negócio. A compra consiste em uma transação que representa um pedido ao fornecedor, que entrega o recurso solicitado dentro de um prazo determinado, enquanto que a venda consiste em uma transação na qual um recurso de negócio é vendido para o cliente.

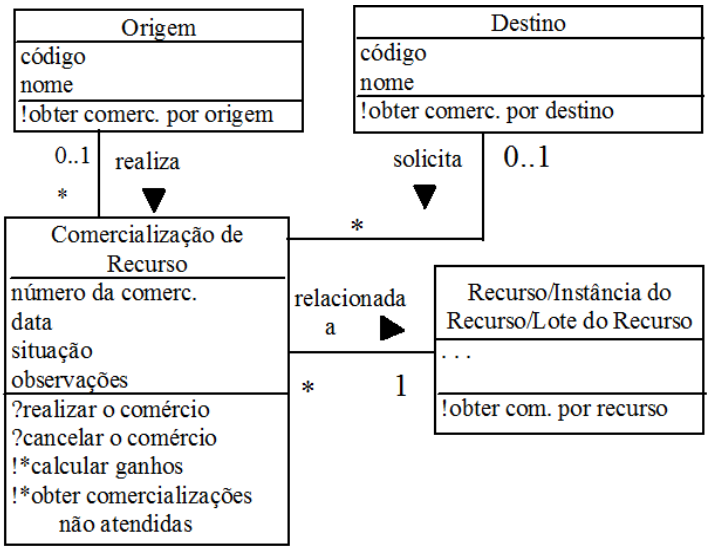

Figura 3.4. Modelo de classes do padrão COMERCIALIZAR RECURSO (BRAGA *et al.*, 1999).

3.2.4 Manter Recurso

O padrão MANTER RECURSO, mostrado na Figura 3.5, apresenta uma alternativa para resolver problemas de aplicações que atuam no contexto de manutenção ou restauração de um determinado recurso de negócio, que pode ser um produto como um eletrodoméstico ou uma fotografia antiga (BRAGA *et al*., 1999).

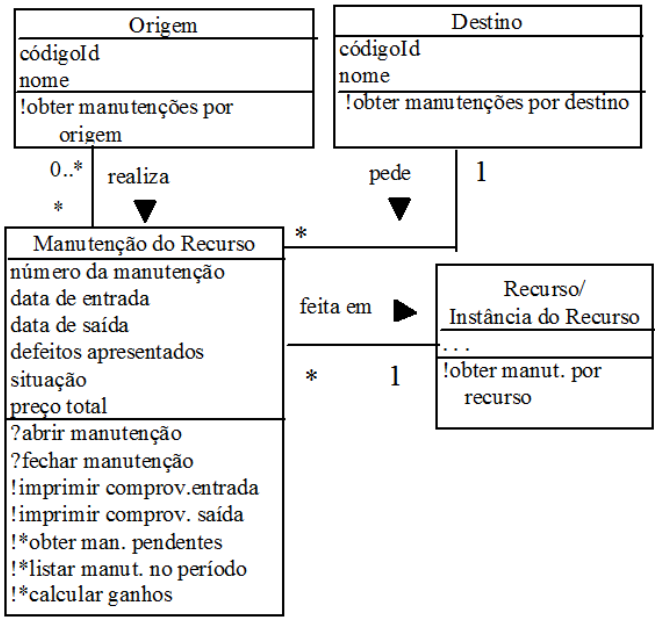

Figura 3.5. Modelo de classes do padrão MANTER RECURSO (BRAGA *et al.*, 1999).

3.2.5 Itemizar Transação do Recurso

O padrão ITEMIZAR TRANSAÇÃO DO RECURSO fornece uma solução para gerenciar vários recursos em uma mesma transação. Esse padrão é usado, por exemplo, em um sistema de locadora de DVDs, em que um cliente pode solicitar a locação de mais de um recurso (DVD) por visita (BRAGA *et al.*, 1999). Sua modelagem pode ser feita como ilustrado na Figura 3.6.

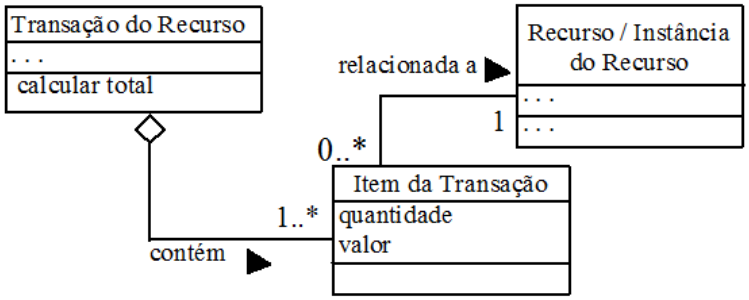

Figura 3.6. Modelo de classes do padrão ITEMIZAR TRANSAÇÃO DO RECURSO (BRAGA *et al.*, 1999).

O uso da linguagem de padrões GRN guia o projetista da aplicação na análise e modelagem de sistemas de informação pertencentes ao domínio de gestão de recursos de negócio, por essa linguagem encapsular os requisitos comuns a todas as aplicações inerentes a esse domínio. Isso proporciona maior agilidade no processo de análise e projeto de aplicações desse domínio, pelo reúso do conhecimento de domínio que uma linguagem de padrões oferece.

3.3 O Framework GREN e o GRENJ

O *framework* GREN é um *framework* de aplicação construído com base na linguagem de padrões GRN (BRAGA, 2002). É um *framework* caixa branca, uma vez que seu reúso ocorre por meio de herança, foi implementado na linguagem Smalltalk (SMALLTALK DOT ORG, 2010) e utiliza o banco de dados relacional MySQL (MYSQL, 2010). Esse *framework* passou por um processo de reengenharia sendo implementado com a linguagem Java, porém preservou a sua funcionalidade e deu origem ao *framework* GRENJ-OO (Gestão de REcursos de Negócio em Java) (DURELLI, 2008).

A arquitetura do *framework* GRENJ-OO está dividida em duas camadas: a de Negócios e a de Persistência, como ilustrado na Figura 3.7. A Camada de Negócios contém um conjunto de classes que representa os padrões da linguagem GRN. A Camada de Persistência é responsável por operações de consulta e persistência de dados em um banco de dados relacional. As Aplicações Específicas consistem em classes específicas da aplicação que estendem as classes da Camada de Negócios. Na parte inferior da Figura 3.7 estão os recursos básicos utilizados pelo *framework* GRENJ-OO, que correspondem ao banco de dados MySQL, à linguagem Java e aos itens de hardware e sistema operacional.

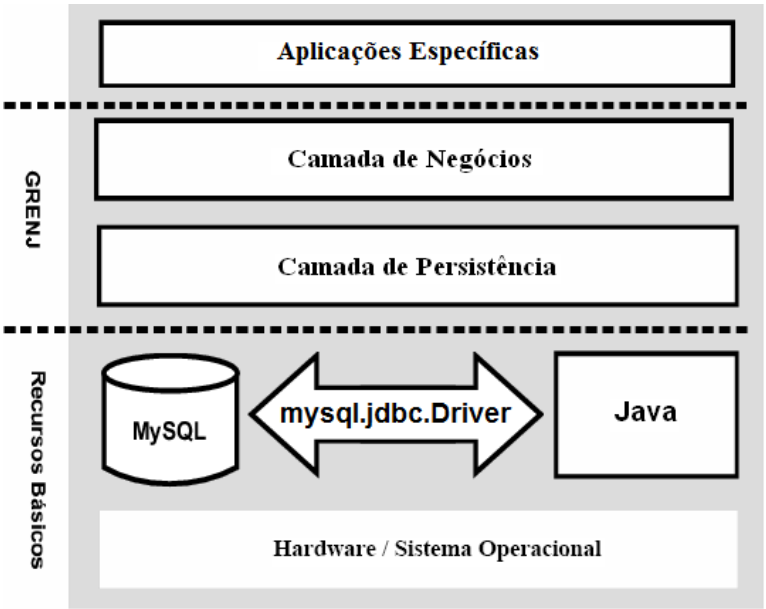

Figura 3.7. Arquitetura do *framework* GRENJ-OO (DURELLI, 2008).

O *framework* GRENJ-OO não previa mecanismos para a geração de aplicações *Web*, assim Viana (2009) adicionou mais três camadas: a de Controle, a de Interface Gráfica *Web* e a de *Wizard*, como mostrado na Figura 3.8. A Camada de Controle interliga a Camada de

Interface Gráfica *Web* com a Camada de Negócios do *framework*, sendo responsável por carregar e enviar informações que são disponibilizadas à Camada de Interface Gráfica *Web*. Essa camada, por sua vez, é responsável por fornecer a estrutura das interfaces *Web*, por enviar mensagens à Camada de Controle e obter resposta e construir páginas, formulários, painéis, tabelas e relatórios. A Camada de *Wizard* consiste em um gerador de aplicações com base em formulários para facilitar a instanciação de aplicações construídas com o apoio da versão do *framework* GRENJ desenvolvida por Viana (2009) denominada GRENJ-GuiWe.

Com a adição dessas três camadas é possível a geração de aplicações *Web* do domínio da linguagem de padrões GRN por meio do preenchimento de formulários providos pelo *Wizard*. Com isso, o engenheiro da aplicação necessita de somente ter o conhecimento da linguagem de padrões GRN para instanciar uma aplicação utilizando o *framework* GRENJ.

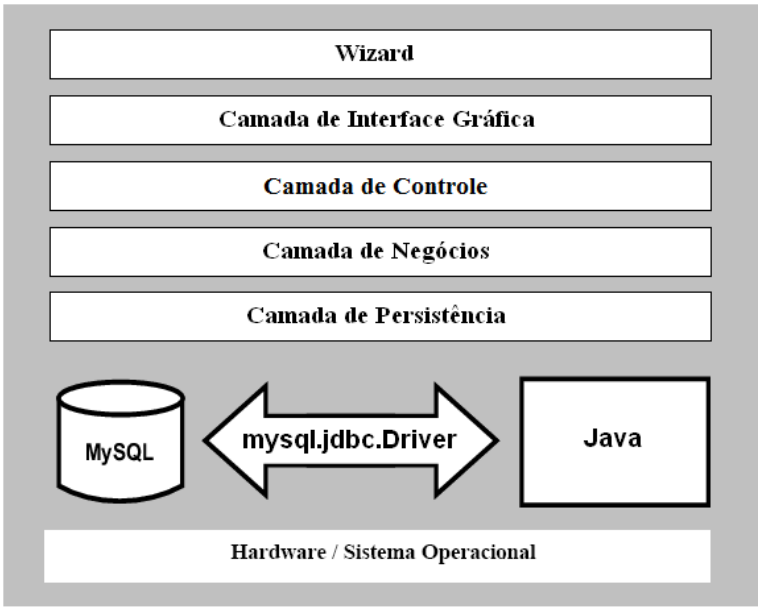

Figura 3.8. Arquitetura do GRENJ-GuiWe com as camadas de controle, interface gráfica e de wizard (VIANA, 2009).

Posteriormente, outro estudo realizado com o *framework* GRENJ-OO foi quanto a utilização de um *framework* de persistência na Camada de Persistência do *framework* (ZANON, 2009). A implementação das funcionalidades de persistência do GRENJ-OO está entrelaçada e espalhada pelo código das classes que implementam a Camada de Negócios desse *framework*. Para resolver esses problemas foi feita a integração do *framework* transversal de persistência desenvolvido por Camargo (2006) com a Camada de Negócios do *framework* GRENJ-OO. Dessa forma, o código relacionado às funcionalidades de persistência foi removido das classes que implementam a Camada de Negócios do GRENJ-OO, desacoplando a implementação das camadas de Persistência e de Negócios. Essa versão do

GRENJ é denominada GRENJ-FT e pode utilizar qualquer sistema de gerenciamento de banco de dados e não somente o utilizado durante o seu desenvolvimento.

3.4 Considerações Finais

Linguagens de padrões fornecem um conjunto de soluções para uma classe de problemas existentes em um determinado domínio de aplicação. Nesta dissertação a linguagem GRN, que é constituída por quinze padrões em nível de análise, foi utilizada. Essa linguagem de padrões deu origem ao *framework* GREN, que posteriormente passou por um processo de reengenharia que originou o *framework* GRENJ-OO, que apóia a instanciação de aplicações do domínio da GRN na linguagem Java. Posteriormente, o *framework* GRENJ-OO passou por outras modificações possibilitando a geração de aplicações *Web* utilizando um *wizard* (versão GRENJ-GuiWe) e a modularização com orientação a aspectos de sua Camada de Persistência, possibilitando o uso de diferentes SGBDs (versão GRENJ-FT). Na Tabela 3.2 é mostrada cada uma das versões do GRENJ juntamente com sua descrição.

Tabela 3.2 – Versões do *framework* GRENJ.

Versão do *framework* GRENJ	**Descrição**
GREN (BRAGA, 2002)	Instancia aplicações em Smalltalk utilizando os padrões da GRN.
GRENJ-OO (DURELLI, 2008)	Instancia aplicações em Java após a reengenharia do GREN. Inclui as camadas de Negócios e de Persistência.
GRENJ-GuiWe (VIANA, 2009)	Além das camadas de Negócios e de Persistência do GRENJ-OO, inclui as camadas de Controle, Interface Gráfica *Web* e de Wizard. Permite a geração de aplicações *Web* no domínio da GRN e instancia aplicações em Java.
GRENJ-FT (ZANON, 2009)	Apresenta a mesma funcionalidade que o GRENJ-OO. A sua Camada de Persistência foi modularizada com um *framework* transversal de persistência. Permite o uso de diversos SGBDs.

Outro problema que ocorre com o GRENJ-OO e também com outros *frameworks*, como o Prevayler (GODIL e JACOBSEN, 2005) e o *framework* de *middleware* apresentado por Zhang e Jacobsen (2004) é que as aplicações que são desenvolvidas com o seu apoio geralmente incorporam toda a sua arquitetura. A arquitetura final da aplicação desenvolvida engloba as classes da aplicação e também todas as classes do *framework*, independentemente se são utilizadas ou não. Isso ocorre, pois há grande quantidade de relacionamentos de dependência entre classes que implementam a Camada de Negócios do GRENJ-OO. Essa dependência ocorre, pois a implementação de cada padrão da GRN e suas variantes no GRENJ-OO está entrelaçada e espalhada pelas classes do *framework*. Com isso, não é

possível instanciar aplicações de maneira incremental com o apoio desse *framework*. Para resolver esse problema o *framework* GRENJ-OO deve passar por um processo de modularização, descrito no próximo capítulo, para que as classes da Camada de Negócios sejam refatoradas com orientação a aspectos, separando os interesses relacionados à implementação de cada padrão e suas variantes do núcleo do *framework*.

Capítulo 4

UM PROCESSO PARA MODULARIZAÇÃO DE FRAMEWORKS

4.1 Considerações Iniciais

As aplicações desenvolvidas com o apoio de *frameworks* como o GRENJ-OO (DURELLI, 2008), geralmente, incluem em sua arquitetura final toda a arquitetura do *framework* ao invés de somente o subconjunto de variabilidades necessárias para o desenvolvimento da aplicação. Dessa forma, há problemas quanto à manutenção, à evolução, ao reúso e ao projeto da aplicação. Outro problema que ocorre em *frameworks* como o GRENJ-OO é que a presença de todas as suas unidades modulares (classes/interfaces/aspectos) exige que vários métodos gancho sejam concretizados durante o processo de instanciação, mesmo que esses não tenham relacionamento com os requisitos da aplicação. O ideal é que aplicações construídas com o apoio de *frameworks* carreguem somente as características (*features*) do *framework* estritamente necessárias para atender aos requisitos da aplicação a ser desenvolvida.

Para solucionar os problemas mencionados foi utilizada a orientação a aspectos para modularizar cada padrão da linguagem de padrões GRN implementado no GRENJ-OO (DURELLI, 2008). A versão GRENJ-OO foi utilizada, pois as versões do GRENJ-GuiWe (VIANA, 2009) e GRENJ-FT (ZANON, 2009) não estavam concluídas quando este projeto foi iniciado. Durante a realização da modularização do *framework* GRENJ-OO foi possível extrair um processo para auxiliar a modularização de *frameworks* criados a partir de linguagem de padrões. Nesse processo, os padrões são considerados como *features* reusáveis que podem ser combinadas para gerar um *framework* mais específico. Um ganho que se

obtém ao utilizar um *framework* modularizado é o comentado acima, que somente as *features* necessárias para atender aos requisitos da aplicação a ser desenvolvida devem constar da arquitetura final da aplicação. Além de facilitar o reúso e a manutenção tanto do *framework* quanto das aplicações por ele instanciadas.

A modularização do GRENJ-OO teve por base o conceito de Linha de Produtos de Software, em que é possível gerar vários produtos (neste caso, várias configurações do *framework*) a partir da combinação de suas *features*. A extensão desses conceitos para *frameworks* introduz, neste trabalho, o conceito de Linha de Produtos de *Frameworks*, LPF, (em inglês, *Framework Product Line - FPL*) que será detalhado a seguir. Este capítulo está organizado em três seções além desta. Na Seção 4.2 o conceito de Linha de Produtos de *Frameworks* é apresentado e ilustrado. Na Seção 4.3 o processo de modularização de *frameworks* é descrito. Na Seção 4.4 são apresentadas as considerações finais.

4.2 Linha de Produtos de Frameworks

Linha de Produtos de *Frameworks* consiste em uma linha de produtos cujos membros são *frameworks* ao invés de aplicações de software. Várias configurações específicas de um *framework* podem ser obtidas, sendo que cada uma inclui somente um subconjunto restrito de *features*. Assim, um novo *framework* denominado *framework* específico (FE) é gerado. Como cada *feature*, que atende a uma funcionalidade específica do *framework*, está implementada em módulos coesos e pouco acoplados, há maior reúso e facilidade de manutenção desse *framework*, pois o desenvolvedor localiza rapidamente a *feature* que deseja manter ou reusar.

Na Figura 4.1 é mostrada a representação esquemática de uma Linha de Produtos de *Frameworks*. O círculo dividido em oito partes (Figura 4.1a)) representa *um framework* modularizado, sendo que cada uma das letras de “A” a “H” denota uma *feature*. O núcleo da Linha de Produtos de *Frameworks* é denotado pela *feature* “A” e por essa razão está sombreada. As demais *features* representadas pelas letras de “B” a “H” são *features* opcionais e alternativas da linha de produtos.

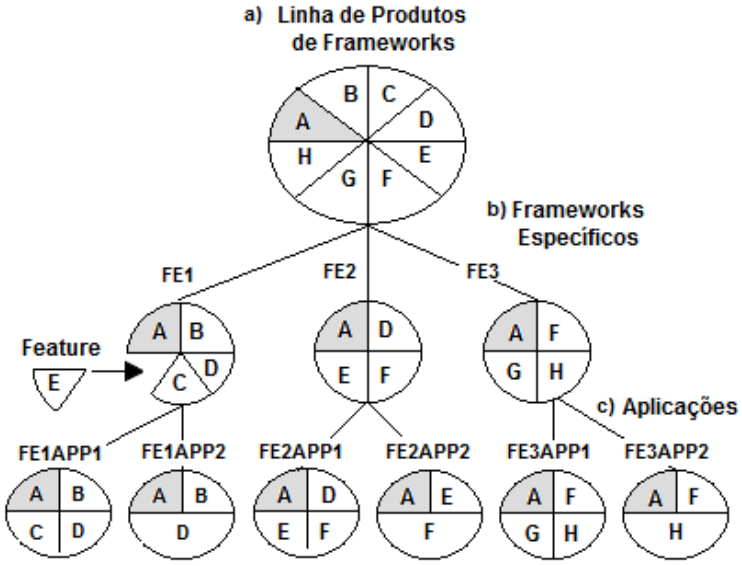

Figura 4.1. Linha de Produtos de Frameworks.

A partir do conjunto de *features* do *framework* modularizado é possível gerar vários *frameworks* específicos (FE) como: FE1 {A, B, C, D}, FE2 {A, D, E, F} e FE3 {A, F, G, H}. Em cada um desses *frameworks* ainda pode haver outras variabilidades, o que possibilita a geração de uma configuração mais específica. Considere, por exemplo, que no *framework* FE1 (Figura 4.1b)) as *features* "C" e "D" sejam independentes. Logo, é possível obter ainda *frameworks* mais específicos como FEC1 = {A, B, C} e FEC2 = {A, B, D}. A partir de cada um desses *frameworks* específicos é possível criar várias aplicações por instanciação. Com o *framework* específico FE1 pode-se instanciar aplicações como FE1APP1 e FE2APP2 exibidas na Figura 4.1c). Assim, as aplicações geradas incluem um conjunto de classes, interfaces e aspectos do *framework*, além das classes de aplicação que estendem classes do *framework*.

Em uma Linha de Produtos de *Frameworks* a incorporação de novas *features* à linha ou a um *framework* específico existente é facilitada. Por exemplo, tanto para adicionar a *feature* "E" ao *framework* específico FE1 (Figura 4.1b)) quanto para adicionar uma nova *feature* à linha (Figura 4.1a)) não é necessário modificar o comportamento (implementação) das demais *features* utilizadas. Com isso, é possível evoluir incrementalmente e com facilidade tanto a Linha de Produtos de *Frameworks* quanto seus membros (FE). Dessa forma, a manutenibilidade é melhorada, uma vez que é mais fácil encontrar um erro ao analisar o código fonte de determinada *feature*, que é um incremento, do que o código fonte de todo o *framework*. Além disso, a extensibilidade das aplicações também é melhorada, por manter separada a implementação de *features* obrigatórias, alternativas e opcionais de um *framework*.

Destaca-se que o conjunto de combinações possíveis entre as *features* de uma LPF é determinado por regras de composição. Essas regras restringem o número de diferentes

frameworks específicos que podem ser gerados a partir da combinação de *features* da LPF. Regras de composição podem ser determinadas pelo modelo de *features* ou por uma linguagem de padrões, que é o caso do *framework* de aplicação GRENJ-OO. Essas regras são a base para a criação de *frameworks* específicos a partir da Linha de Produtos de *Frameworks*.

O conceito de LPF foi aplicado ao *framework* GRENJ-OO como mostrado na Figura 4.2a), sendo que as *features* P1 a P15 representam os padrões da GRN implementados no GRENJ-OO. A partir da combinação dessas *features* é possível obter vários *frameworks* específicos. Por exemplo, um FE {P1, P2, P4, P11} é o que trata de transações de aluguel, gerado pela combinação de um subconjunto de suas *features* (Figura 4.2b)), com o qual é possível instanciar aplicações do subdomínio de locação tais como as de "Controle de Locação de DVDs", "Controle de Empréstimos de Livros", "Controle de Locação de Veículos", dentre outros, como ilustrado na Figura 4.2c).

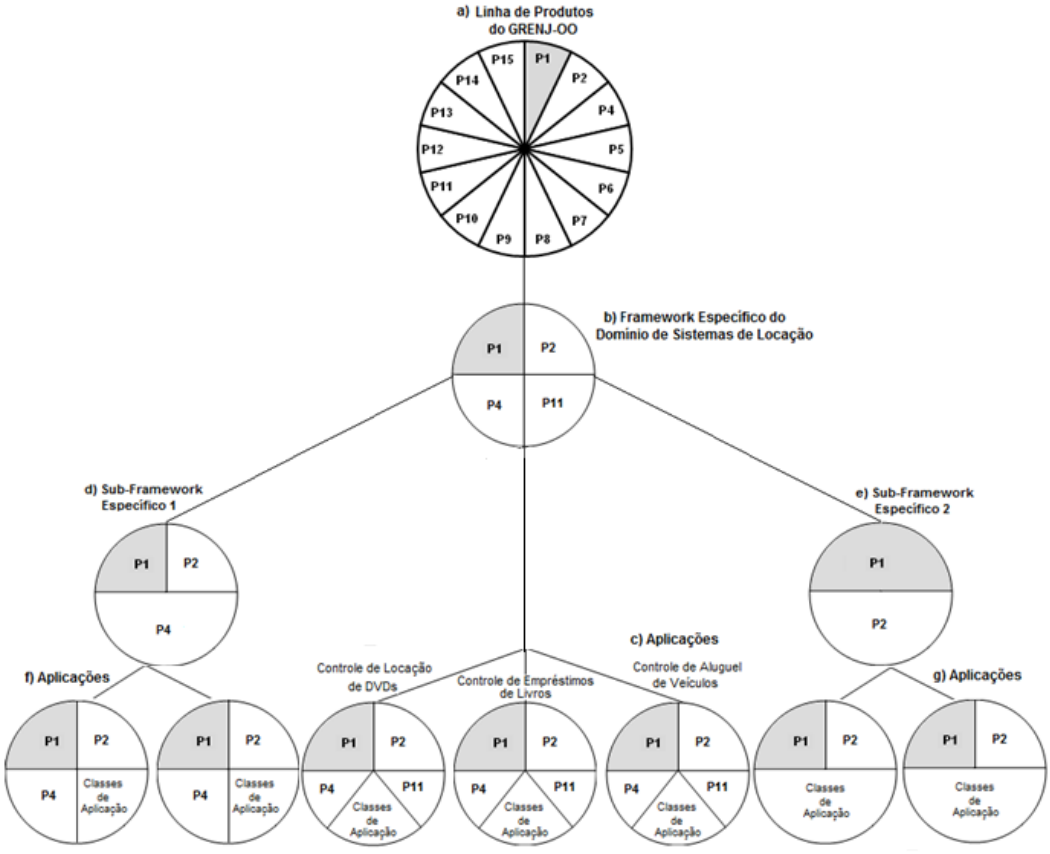

Figura 4.2. Linha de produtos do *framework* GRENJ-OO, *framework* específico, *subframeworks* específicos e aplicações.

O FE da Figura 4.2b) é composto por *features* que implementam os padrões Identificar Recurso (P1), Quantificar Recurso (P2), Locar Recurso (P4) e Itemizar Transação do Recurso (P11) da linguagem GRN. Considerando o grafo de aplicação dos padrões da GRN (Figura 3.1) tem-se que as *features* P4 e P11 são opcionais, o que possibilita a geração de

subconfigurações desse FE, denominados *subframeworks* (SFrws), que consistem nas combinações entre as *features* {P1, P2, P4} e {P1, P2} (Figuras 4.2d) e 4.2e)). Dentro dessas subconfigurações ainda há variabilidades, podendo ser subdivididas para gerar outros *SFrws*. A partir da instanciação desses *SFrws* várias aplicações podem ser geradas (Figuras 4.2f) e 4.2g)). Apesar da *feature* P11 ser opcional ela é dependente da *feature* P4, razão pela qual não é possível gerar uma subconfiguração com {P1, P2, P11}.

O conceito de LPF deve ser aplicado na modularização de *frameworks* que contêm grande quantidade de *features* e de classes, de modo a propiciar que a partir do conjunto de *features* seja possível gerar várias configurações de um *framework*. Como a implementação de suas *features* é desacoplada umas das outras, a composição das mesmas é facilitada, permitindo a geração de `n` configurações mais restritas desse *framework*.

Na próxima seção é descrito o processo de modularização de *frameworks* que foi criado a partir da modularização do *framework* GRENJ-OO.

4.3 Processo de Modularização de Frameworks

O processo de modularização de *frameworks* tem o objetivo de apoiar engenheiros de software interessados em evoluir *frameworks* de modo a transformá-los em Linhas de Produtos de *Frameworks*. Esse processo foi definido com base em um conjunto de diretrizes de manutenção de *frameworks* propostas por Oliveira *et al.* (2010).

Na Figura 4.3 é mostrado que a partir de um *framework* desenvolvido com linguagem de padrões (*features)*, com a implementação de suas *features* (F1, F2, F3 e F4) altamente acoplada, é possível obter uma LPF de modo que cada aplicação instanciada a partir de um FE contenha somente as *features* necessárias para a aplicação.

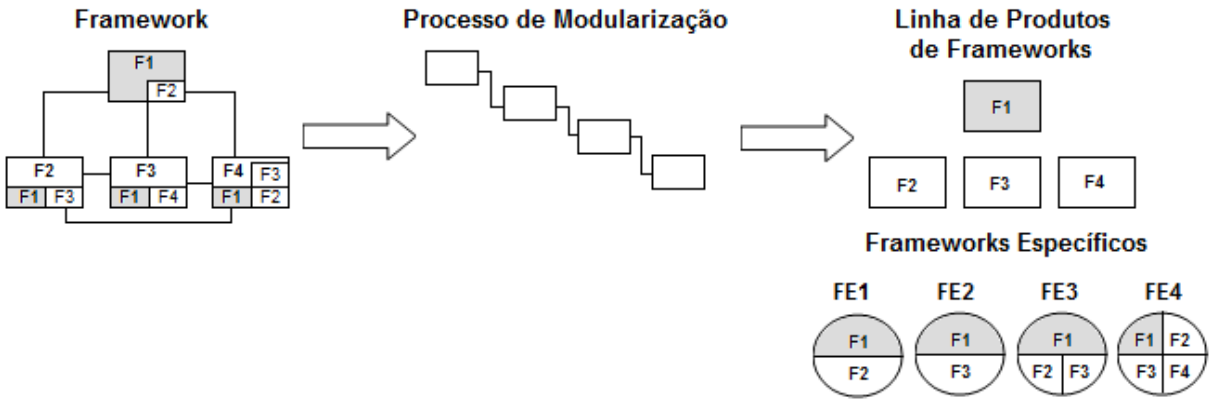

Figura 4.3. Aplicação do processo de modularização em um *framework* desenvolvido com linguagem de padrões de análise.

O processo de modularização de *frameworks*, mostrado na Figura 4.4, usa a notação SADT (ROSS, 1977), é composto por quatro etapas: 1) Identificar as Unidades Originais do Interesse, 2) Identificar Espalhamento de Interesses, 3) Identificar Entrelaçamento de Interesses e 4) Modularizar com Aspectos. Esse processo é iterativo e incremental, no qual todas as suas etapas devem ser aplicadas para cada interesse (*concern*) presente no *framework* a ser modularizado. Esse processo pode ser denominado orientado a interesse, uma vez que a cada iteração, a implementação de um interesse inerente ao *framework* é modularizada.

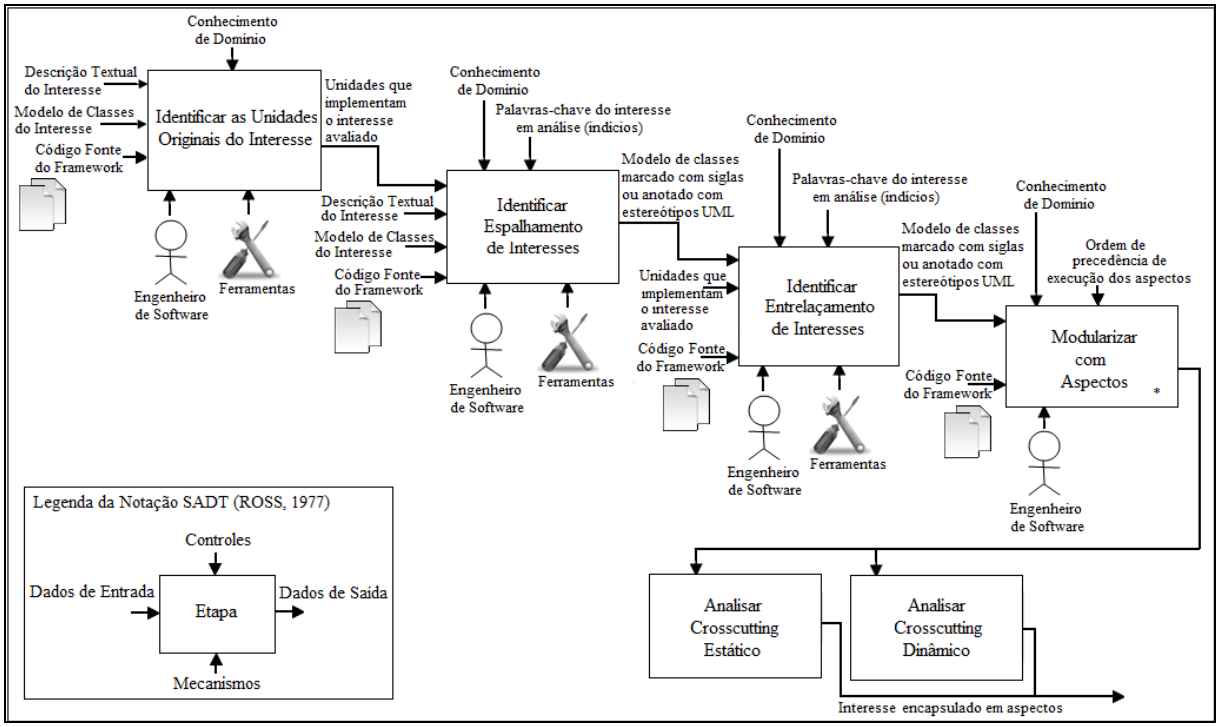

Figura 4.4. Processo de modularização de *frameworks*.

As etapas "Identificar as Unidades Originais do Interesse", "Identificar Espalhamento de Interesses" e "Identificar Entrelaçamento de Interesses" podem ser apoiadas por ferramentas como o ComSCId (*Computational Support for Concern Identification*)

(PARREIRA JÚNIOR *et al.*, 2010) e DMAsp (*Design Model to Aspect*) (COSTA *et al*., 2009), que são *plugins* do Eclipse (THE ECLIPSE FOUNDATION, 2010). ComSCId apóia a identificação de interesses transversais em um código fonte e pode ser customizado para adicionar novos interesses presentes nesse código. Isso é feito adicionando o nome do interesse e uma lista de palavras-chave, as quais representam indícios desse interesse, que é armazenado em um arquivo XML utilizado pelo ComSCId.

DMAsp apóia a geração de um modelo de classes orientado a objetos anotado com indícios de interesses transversais encontrados pelo ComSCId. Nesse modelo, atributos e operações são anotados com estereótipos UML, indicando que eles contribuem para a implementação de um determinado interesse transversal cadastrado na ferramenta ComSCId. Esse modelo auxilia engenheiros de software a pensar em estratégias e alternativas para construir um modelo de classes orientado a aspectos de melhor qualidade (COSTA *et al*., 2009).

Nas próximas subseções cada uma das etapas do processo de modularização é apresentada detalhadamente.

4.3.1 Identificar as Unidades Originais do Interesse

Dado um interesse a ser modularizado, que pode ser uma *feature*, um interesse transversal, um padrão ou variante de padrão, esta etapa tem como objetivo identificar quais as unidades (classes/interfaces/aspectos) do *framework* que foram originalmente projetadas para implementá-lo. Essas unidades são chamadas de "Originais" desse interesse. Uma maneira de verificar se uma unidade é "Original" é por meio da análise de sua descrição textual e de seu modelo de classes. Caso esses artefatos do interesse em análise não estejam disponíveis, o engenheiro de software usa o seu conhecimento do domínio do *framework* para definir um conjunto de palavras-chave que identificam o interesse em análise.

Em seguida, deve-se minerar o código fonte de cada classe do *framework* em busca dos indícios do interesse que estão em sua descrição textual e em seu modelo de classes (Figura 4.5) ou na lista de palavras-chave definidas pelo engenheiro de software. A mineração é de responsabilidade do engenheiro de software e pode ser feita manualmente ou com o apoio automatizado, como o ComSCId (PARREIRA JÚNIOR *et al.*, 2010). Posteriormente, o engenheiro de software deve analisar as unidades do *framework* que contêm indícios do interesse avaliado juntamente com o seu modelo de classes ou com o seu conhecimento de

domínio para determinar quais unidades foram originalmente projetadas para implementá-lo, que é o artefato de saída produzido nesta etapa.

O conhecimento de domínio influencia a realização desta etapa do processo, pois facilita a análise das entradas para encontrar as unidades que foram originalmente projetadas para implementar o interesse em análise.

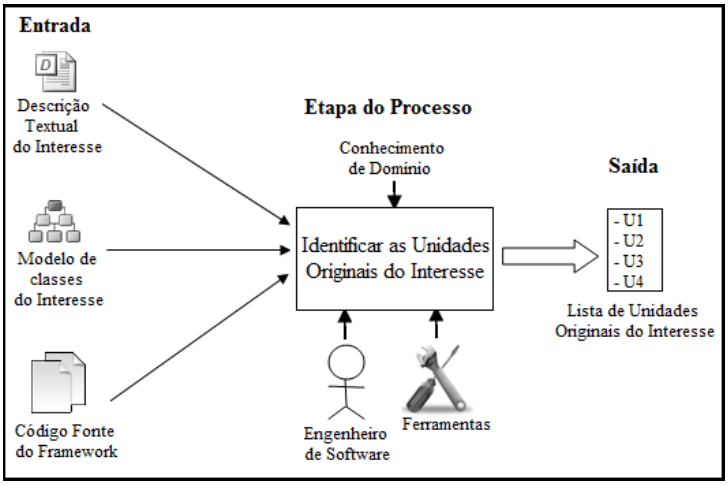

Figura 4.5. Etapa Identificar as Unidades Originais do Interesse.

4.3.2 Identificar Espalhamento de Interesses

Nesta etapa, o objetivo é verificar se há código relacionado ao interesse em análise em outras unidades do *framework* a ser modularizado, além das unidades originais. Caso sejam encontrados indícios do interesse pode-se constatar que há espalhamento na implementação do interesse em análise.

Na realização desta etapa, o engenheiro de software primeiramente deve analisar a lista de unidades que foram projetadas para implementar o interesse em análise, a descrição textual e o modelo de classes do interesse para extrair um conjunto de palavras-chave, que representam indícios do interesse em análise, como mostrado na Figura 4.6. Caso esses artefatos não estejam disponíveis, as palavras-chave devem ser extraídas a partir da lista de unidades originais do interesses criada na etapa anterior e pelo conhecimento de domínio do *framework* que o engenheiro de software possui. Em seguida o código fonte do *framework* deve ser minerado em busca de indícios do interesse em análise. Isso pode ser feito manualmente, analisando cada classe do *framework*, ou pode contar com o apoio automatizado de ferramentas como o ComCSId (PARREIRA JÚNIOR *et al.*, 2010).

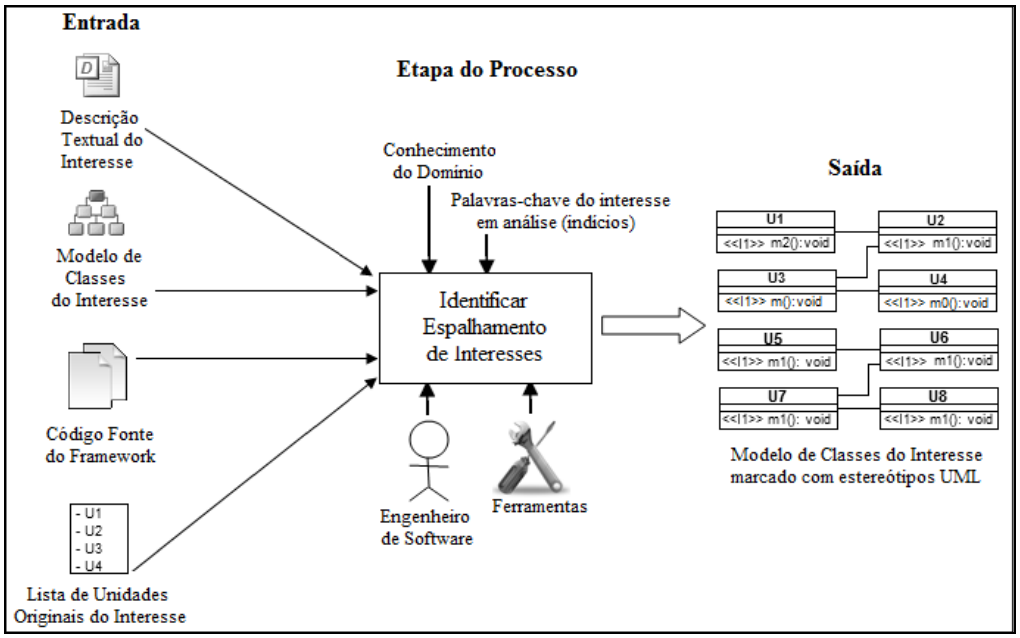

Figura 4.6. Descrição da etapa "Identificar Espalhamento de Interesses".

Após a mineração do código fonte do *framework*, o engenheiro de software deve elaborar o modelo de classes do interesse a partir da lista de suas unidades originais e da análise de outras unidades do *framework* que contêm indícios do interesse a ser modularizado. Os atributos e operações de cada uma dessas unidades devem ser marcados com siglas ou estereótipos da UML (BOOCH *et al.*, 2005) que os associam com o interesse em análise. O engenheiro de software pode utilizar ferramentas de modelagem UML como o Astah (CHANGE VISION, 2010) na elaboração desses modelos que compõem a saída desta etapa.

Na Figura 4.6, o exemplo de modelo de classes gerado nesta etapa do processo contém as unidades U1, U2, U3 e U4 que representam as unidades que foram originalmente projetadas para implementar o interesse I1, que é o interesse a ser modularizado. As operações dessas unidades que estão relacionadas ao interesse I1 estão marcadas com o estereótipo <<I1>>. As unidades U5, U6, U7 e U8 não foram originalmente projetadas para implementar o interesse I1, mas contêm operações relacionadas a ele e estão marcadas com o estereótipo <<I1>>. Isso denota que a implementação do interesse I1 está espalhada pelas unidades do *framework* a ser modularizado. A operação não relacionada ao interesse em análise que contém código fonte do interesse que está sendo modularizado deve ser marcada com um estereótipo UML no modelo de classes do interesse em análise. Isso indica que a operação contribui para a implementação do interesse em análise.

4.3.3 Identificar Entrelaçamento de Interesses

Após a identificação do espalhamento de interesses relacionados ao interesse em análise pelas classes do *framework* a ser modularizado, deve-se verificar se há indícios relacionados a outros interesses pelas unidades originais do interesse do *framework* que está sendo modularizado. Se forem encontrados significa que essas unidades também contribuem para implementá-los, logo há entrelaçamento de interesses nessas unidades.

Para realizar esta etapa os artefatos produzidos nas etapas anteriores bem como o conhecimento do domínio pelo engenheiro de software são necessários como mostrado na Figura 4.7. O engenheiro de software deve minerar o código fonte das unidades do *framework* que foram projetadas para implementar o interesse em análise, em busca de atributos e operações que não contêm indícios relacionados ao interesse em análise. Isso pode feito manualmente, analisando-se os atributos e operações, de cada unidade original, que não foram marcados na etapa "Identificar Espalhamento de Interesses" e utilizar o conhecimento de domínio para determinar os interesses aos quais esses atributos e operações estão relacionados. Ferramentas automatizadas, como o ComSId, podem ser utilizadas para que o engenheiro de software forneça conjuntos de palavras-chave relacionadas a outros possíveis interesses que podem estar localizados nas unidades originais.

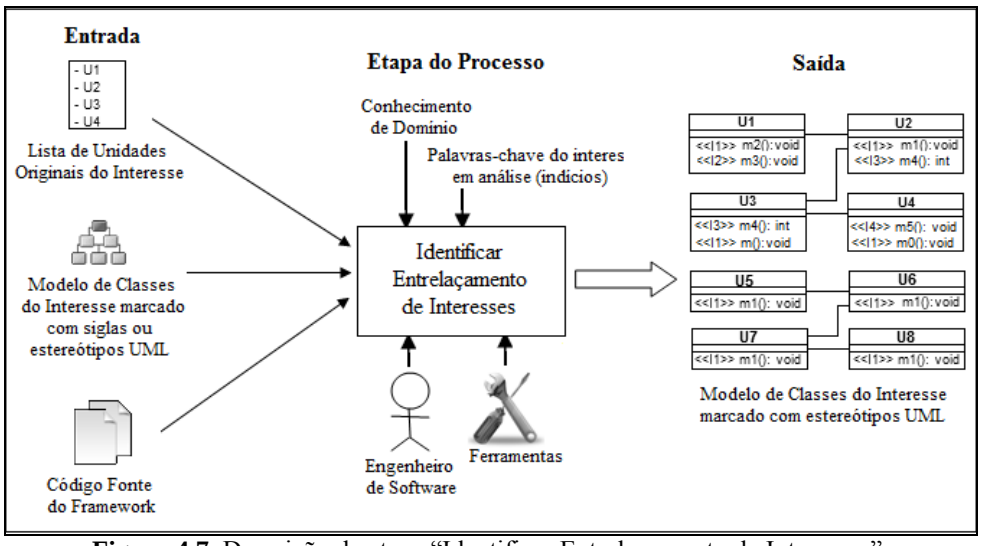

Figura 4.7. Descrição da etapa "Identificar Entrelaçamento de Interesses".

A definição inadequada dessas palavras-chave pode conduzir o engenheiro de software a associar de maneira incorreta os atributos e operações das unidades originais com os interesses que implementam.

Após a mineração do código fonte do *framework*, o engenheiro de software deve refinar o modelo de classes do interesse em análise elaborado na etapa "Identificar

Espalhamento de Interesses", adicionando os atributos e operações das unidades originais que contribuem para a implementação de outros interesses. Esses atributos e operações devem ser marcados com siglas ou estereótipos da UML (BOOCH *et al.*, 2005) os quais os associam aos interesses que cada um implementa. As ferramentas já comentadas em etapas anteriores podem ser usadas para gerar esse modelo de classes, que é a saída desta etapa do processo.

Na Figura 4.7 é ilustrado um exemplo de refinamento do modelo de classes apresentado na Figura 4.6, produzido na etapa "Identificar Espalhamento de Interesses". Os métodos `m3()`, `m4()` e `m5()` foram adicionados nas unidades originais (U1, U2, U3 e U4) do interesse em análise (I1), mas contribuem para a implementação de outros interesses, denotando um entrelaçamento de interesses. Por essa razão foram marcados com estereótipos <<I2>>, <<I3>> e <<I4>> que representam os interesses I2, I3 e I4 do *framework* a ser modularizado. Quando o entrelaçamento é observado, novos estereótipos são gerados para representar o interesse encontrado. No exemplo acima, `m4()` contribui para a implementação do interesse I3, assim o estereótipo <<I3>> foi criado para representá-lo.

4.3.4 Modularizar com Aspectos

O objetivo desta etapa é analisar as marcações que foram feitas no modelo de classes produzido na etapa anterior e refatorar atributos e métodos marcados para se tornarem aspectos, a fim de remover o entrelaçamento e o espalhamento de interesses relacionados a esse interesse do *framework* que está sendo modularizado.

Em virtude de linguagens orientadas a aspectos, como AspectJ (KICZALES *et al.*, 1997), fornecerem estruturas para alterar o comportamento estático e dinâmico do fluxo de execução de um programa, esta etapa do processo está dividida em duas subetapas:

- Analisar o *Crosscutting* Estático, que consiste em verificar atributos e operações que devem ser adicionados a unidades do *framework* em tempo de compilação, por meio de *inter-type declarations*. Por exemplo, considere a classe `A` da Figura 4.8 que contém uma declaração de atributo relacionada ao interesse implementado pela classe `B` (linha 05). Essa declaração deve ser modularizada por meio de *crosscutting* estático. Dessa forma, o atributo `atrB` foi removido da classe `A` e encapsulado no aspecto `ConcernB`, que novamente o introduz por meio de *inter-type declarations* na classe `A` (linha 06) ; e
- Analisar o *Crosscutting* Dinâmico, que consiste em analisar trechos de código relacionados ao interesse em análise que devem ser adicionados em tempo

execução, por meio de *advices*. Por exemplo, dentro do método construtor da classe `A` da Figura 4.8 há código relacionado ao interesse implementado pela classe `B` (linha 09), o que indica que esse código que deve ser modularizado por meio de *crosscutting* dinâmico. Assim, esse código foi removido do método construtor da classe `A` e encapsulado no aspecto `ConcernB`, por meio de um *advice* (linhas 08-14), que captura a execução do método construtor de `A` e inicializa o atributo `atrB`.

```
01 package sampleapp.model;
02
03 public class A {
04    //Crosscutting estático
05    private B atrB;
06
07    public A(){
08       //Crosscutting dinâmico
09       atrB = null;
10    }
11 }

////////////////////////////////////////
01 package sampleapp.aspects;
02
03 public privileged aspect ConcernB{
04
05   //Crosscutting estático
06   private B A.atrB;
07
08   void around(A a):
09       execution(public A.new())
10       && target(a){
11          //Crosscutting dinâmico
12          a.atrB = null;
13          proceed(a);
14   }
15 }
```

Figura 4.8. Exemplos de *crosscutting* estático e *crosscutting* dinâmico.

Para a realização dessa etapa o engenheiro de software deve inicialmente identificar quais atributos e operações relacionados ao interesse em análise, que estão em unidades do *framework* projetadas para implementar outros interesses que devem ser modularizados por meio de *crosscutting* estático e *crosscutting* dinâmico. Isso significa definir o que deve ser implementado utilizando o mecanismo de *inter-type declarations* e o que deve ser implementado por meio de *advices* (Figura 4.9). Posteriormente, deve-se analisar o código fonte das unidades originais do interesse em análise para identificar os atributos e operações que estão relacionados a outros interesses, que devem ser removidos dessas unidades e modularizados em outras iterações do processo.

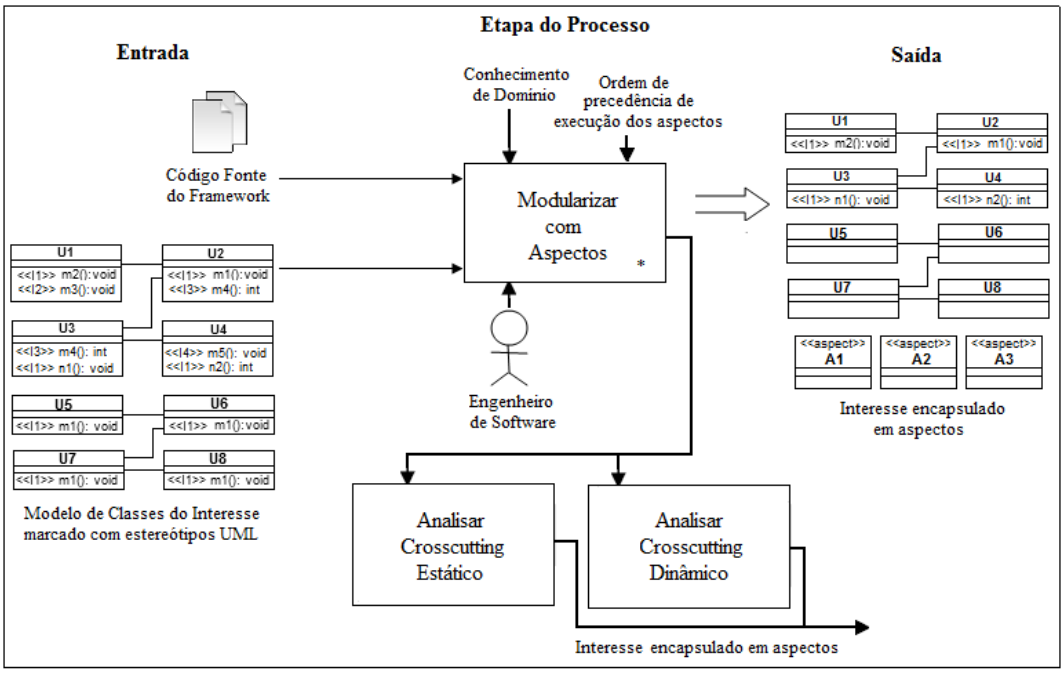

Figura 4.9. Descrição da etapa "Modularizar com Aspectos".

Após análise do código fonte do *framework*, o engenheiro de software deve utilizar o seu conhecimento de domínio para estabelecer os aspectos que devem ser criados para modularizar a implementação do interesse em análise das demais unidades do *framework* que estão relacionadas a outros interesses. Isso deve ser feito para desacoplar a implementação do interesse em análise das unidades que foram projetadas para implementar outros interesses. Como comentado na Seção 4.3.1, podem ser considerados interesses: uma *feature*, um interesse transversal, um padrão ou variante de padrão.

Com a definição dos aspectos que devem ser criados, deve-se iniciar a modularização do interesse em análise com orientação a aspectos. Primeiramente, os atributos e operações classificados pelo engenheiro de software como *crosscuting* estático devem ser removidos de unidades do *framework* que foram projetadas para implementar outros interesses. Esses atributos e operações devem ser encapsulados em aspectos, que novamente os introduzem nessas unidades por meio de *inter-type declarations*. É possível utilizar os idiomas propostos por Hanenberg *et al.* (2003) como o *Container Introduction*, para implementar o *crosscutting* estático.

Implementado o *crosscutting* estático, o engenheiro de software deve remover o código relacionado ao interesse em análise das operações classificadas como *crosscutting* dinâmico. Esse código deve ser removido desses métodos e *advices* devem ser criados em um aspecto para separar o código relacionado ao interesse em análise desses métodos. Por exemplo, considere o modelo de classes da Figura 4.10 com duas classes. Uma que foi projetada para implementar o "interesse 1" (`Classe1`) e outra para implementar o "interesse

2" (`Classe2`). Na `Classe2` há código relacionado ao "interesse 1" em seu método construtor, o que é um exemplo de funcionalidade que deve ser modularizada por meio de *crosscutting* dinâmico. Para modularizar a implementação da `Classe2`, o trecho de código relacionado ao "interesse 1" deve ser removido e encapsulado em um aspecto, por meio de um *advice*. Dessa forma, evita-se o espalhamento de interesses relacionado ao "interesse 1" pela `Classe2`.

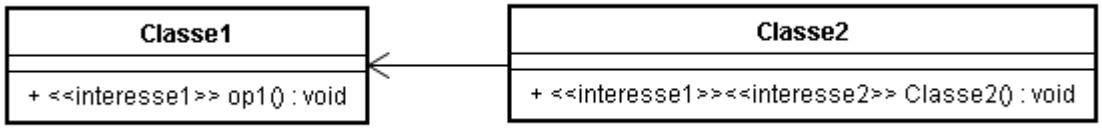

Figura 4.10. Exemplo de *crosscutting* dinâmico.

Durante a modularização de interesses de um *framework* com orientação a aspectos é importante considerar a ordem de precedência de execução dos aspectos existentes, caso existam, para implementar novos aspectos. Isso é relevante uma vez que determinado aspecto pode entrecortar um conjunto de métodos que outro aspecto introduziu por *inter-type declarations* em uma classe. Sendo assim, para evitar erros de compilação é necessário que o aspecto que utiliza *inter-type declarations* para introduzir métodos nessa classe execute primeiro que o aspecto que os entrecorta para adicionar comportamento aos mesmos. Outro problema que pode ocorrer quando não se considera a ordem de precedência de execução dos aspectos são comportamentos inesperados que podem causar erros de execução em situações nas quais há dois ou mais *advices* que acessam o mesmo ponto de junção (chamada ou execução de um método) para adicionar determinados comportamentos ao mesmo. Esses *advices* podem estar localizados no mesmo aspecto ou em diferentes aspectos, como exemplificado na Figura 4.11. Os aspectos `X` e `Y` entrecortam o método `m1()` da classe `Z` para adicionar comportamentos distintos a esse método. Por esse motivo é necessário utilizar o mecanismo de controle de precedência presentes em linguagens orientadas a aspectos nessas situações, de modo a evitar erros de compilação e de execução.

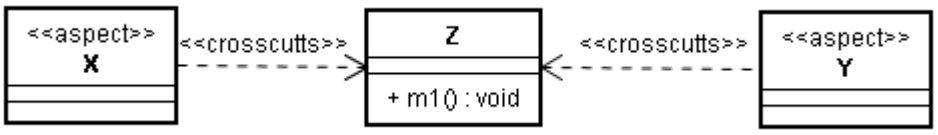

Figura 4.11. Problemas de ordem de precedência de execução entre aspectos.

4.4 Considerações Finais

O processo de modularização de *frameworks* apresentado foi obtido a partir da modularização do *framework* de aplicação GRENJ-OO, que é baseado na linguagem de padrões GRN, e pode ser aplicado na modularização de outros *frameworks* que tenham as mesmas características. Os *frameworks* modularizados podem ser transformados em Linhas de Produtos de *Frameworks*, na qual seus membros são *frameworks*, ao invés de aplicações.

A aplicação do processo de modularização auxilia o engenheiro de software a separar a implementação de *feature*s opcionais e alternativas de um *framework* de seu núcleo, por desacoplá-las umas das outras. Isso permite ao engenheiro de aplicação selecionar apenas o conjunto de *features* necessário para satisfazer os requisitos da aplicação.

Esse processo possui relação com alguns princípios da Decomposição Horizontal (ZHANG e JACOBSEN, 2004), em virtude de ambos terem o enfoque em separar o núcleo do *framework* da implementação de outras *features* e no *refactoring* incremental de *features*. Mas diferente da Decomposição Horizontal, que possui maior enfoque na modularização de *crosscutting features*, que são *features* relacionadas a interesses transversais, o processo apresentado neste capítulo também trata da modularização de *features* opcionais e alternativas de *frameworks*. No capítulo seguinte é apresentada a aplicação desse processo na modularização do *framework* GRENJ-OO.

Capítulo 5

MODULARIZAÇÃO DOS PADRÕES DA LINGUAGEM GRN NO GRENJ

5.1 Considerações Iniciais

A linguagem de padrões GRN fornece um conjunto de quinze padrões em nível de análise para o desenvolvimento de aplicações no domínio de gestão de recursos de negócio, que consistem em aplicações que tratam de aluguel, compra, venda ou manutenção de um bem ou serviço. O *framework* GRENJ-OO (DURELLI, 2008) foi construído com base nesses padrões e suas variantes, os quais podem ser vistos como *features* desse *framework*. Uma *feature* consiste em uma funcionalidade do sistema que possui algum valor para o cliente (KANG *et al.*, 1990).

Durante a construção do *framework* GRENJ-OO não houve a preocupação em separar os interesses inerentes a cada um dos padrões da GRN e de construí-lo de maneira a facilitar o gerenciamento de variabilidades. Assim, a arquitetura da versão original desse *framework* apresenta muitos relacionamentos de dependência entre classes, dificultando seu uso e processos de manutenção. Neste capítulo é apresentado como a orientação a aspectos foi utilizada para modularizar cada um dos padrões implementados nesse *framework*, seguindo o processo de modularização de *frameworks* descrito no Capítulo 4.

Este capítulo está organizado em duas seções além desta. Na Seção 5.2 é apresentada a modularização dos padrões Identificar Recurso, Quantificar Recurso e Locar Recurso da GRN implementados no *framework* GRENJ-OO. A modularização dos demais padrões é apresentada em Oliveira e Penteado (2010). Na Seção 5.3 são apresentadas as considerações finais do capítulo.

5.2 Modularização dos padrões da Linguagem GRN com Orientação a Aspectos

Os padrões foram modularizados utilizando a linguagem AspectJ (ASPECTJ PROJECT, 2010), que é uma extensão orientada a aspectos da linguagem Java. A modularização de cada um dos padrões da GRN é descrita seguindo as etapas definidas no processo de modularização de *frameworks* apresentado no Capítulo 4.

5.2.1 Identificar Recurso

O padrão Identificar Recurso fornece uma maneira de representar os "recursos de negócio" envolvidos em transações processadas pelo sistema (BRAGA *et al*., 1999). São considerados "recursos de negócio", por exemplo, um livro, em um sistema de biblioteca ou um eletrodoméstico em um sistema de manutenção de equipamentos eletrônicos.

A implementação desse padrão na arquitetura do GRENJ-OO encontra-se diluída por vários atributos, métodos, classes e interfaces. Não há um conjunto coeso de unidades de implementação que o representa. Por exemplo, há classes que foram originalmente projetadas para a implementação de outros padrões, mas contêm atributos e métodos relacionados ao padrão Identificar Recurso. Da mesma forma, também há classes que foram originalmente projetadas para implementar o Identificar Recurso que possuem atributos e métodos que contribuem para a implementação de outros padrões.

Na primeira etapa do processo – Identificar as Unidades Originais do Interesse – foram analisados a descrição textual e o modelo de classes do padrão, extraídos da linguagem de padrões GRN. A partir deles, o código fonte do *framework* foi analisado sendo que as classes `Resource`, `SimpleType` e `NestedType` foram projetadas exclusivamente para implementar esse padrão. Não houve dificuldades na realização desta etapa porque as informações presentes na descrição textual e no modelo de classes do padrão foram suficientes.

Na segunda etapa – Identificar Espalhamento de Interesses – o objetivo é analisar todas as classes do *framework* para verificar a presença de indícios de interesse relacionados ao padrão Identificar Recurso. Nesta etapa, as unidades originais do padrão, a descrição textual e o modelo de classes do padrão foram analisados para extrair um conjunto de palavras-chave que representam indícios de interesses relacionados ao padrão. Com base

nessas informações, o código fonte do *framework* foi analisado para identificar atributos e operações de suas classes que contribuem para a implementação do padrão. Além das classes `Resource`, `SimpleType` e `NestedType`, foi verificada a presença de código relacionado ao Identificar Recurso em classes como: `QuantificationStrategy`, `Instantiable-Resource`, `BusinessResourceTransaction`, `BusinessResourceQuotation`, `TransactionItem` e `QuotationItem`, que foram originalmente projetadas para implementar outros padrões. Por exemplo, na classe `QuantificationStrategy`, que foi exclusivamente projetada para implementar o padrão Quantificar Recurso, há um atributo do tipo `Resource` e métodos como `getResource()` e `setResource()`, que contribuem para a implementação do padrão Identificar Recurso, como ilustrado no trecho de código da Figura 5.1.

```
01 package grenj.model;
02
03 public abstract class QuantificationStrategy {
04
05    private Resource resource;
06
07    public Resource getResource(){ … }
08
09    public void setResource(Resource resource){ … }
10 }
```

Figura 5.1. Trecho de código da classe QuantificationStrategy com indícios de interesses relacionados ao padrão Identificar Recurso no GRENJ-OO.

Após a análise do código fonte do *framework* em busca de indícios relacionados ao padrão Identificar Recurso, foi elaborado, com o auxílio de uma ferramenta de modelagem UML, o modelo de classes do padrão a partir das unidades originais e de outras unidades do *framework* que contêm indícios do padrão, como mostrado na Figura 5.2. Os atributos e operações relacionados ao padrão Identificar Recurso que estão em classes que implementam outros padrões, como `QuantificationStrategy`, foram marcados com o estereótipo <<IR>> (Identificar Recurso) nesse modelo de classes. Esse diagrama fornece o mapeamento dos atributos e operações de outras classes/interfaces/aspectos do *framework* que contêm indícios de interesse relacionados ao padrão.

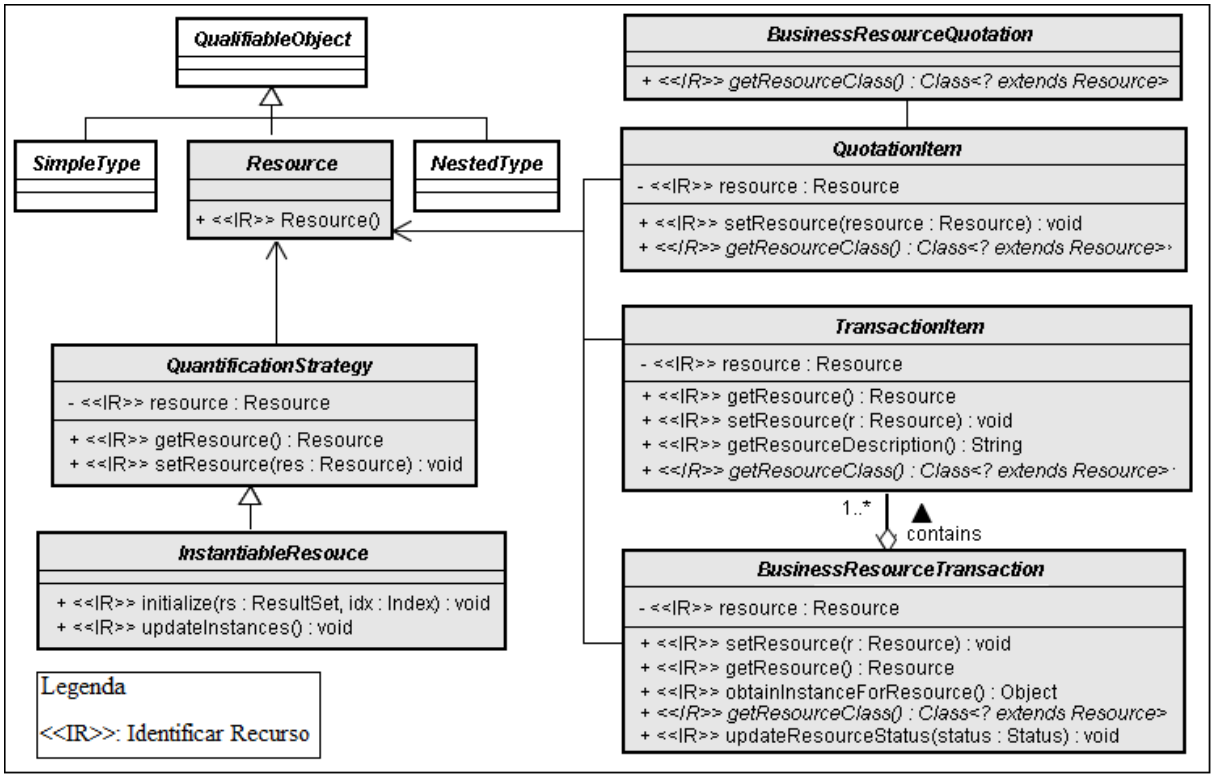

Figura 5.2. Modelo de classes do padrão Identificar Recurso com indícios de espalhamento de interesses.

Na terceira etapa – Identificar Entrelaçamento de Interesses –, o objetivo é analisar cada uma das unidades originais identificadas na primeira etapa em busca de atributos e métodos que estão relacionados a outros padrões ou outros interesses. Com base no conhecimento de domínio, na lista de unidades originais do padrão e nas palavras-chave que representam indícios de interesses relacionados ao padrão (obtidas na etapa 2), cada uma das unidades originais do padrão foi analisada em busca de atributos e operações não relacionados ao padrão. Essa análise se iniciou pela classe `Resource`, em virtude de ser a classe principal do padrão. Foi observada a presença de operações relacionadas ao padrão Quantificar Recurso, como `updateStatus()`, `getMeasureUnityClass()` e `getInstances()` nessa classe, como mostrado na Figura 5.3.

```
01 package grenj.model;
02
03 public abstract class Resource extends QualifiableObject {
04
05    //Padrão Quantificar Recurso
06    public int updateStatus(Status status){ … }
07
08    //Estratégia de Quantificação Recurso Mensurável do padrão
09    //Quantificar Recurso
10    public abstract Class<? extends MeasureUnity>
11          getMeasureUnityClass() { ... }
12
13    //Estratégia de Quantificação Recurso Instanciável do padrão
14    //Quantificar Recurso
15    public List<ResourceInstance> getInstances() { ... }
16
17 }
```

Figura 5.3. Trecho de código da classe Resource com indícios de interesses relacionados ao padrão Quantificar Recurso no GRENJ-OO.

Após a análise do código fonte das unidades originais do padrão em busca de atributos e operações relacionados a outros padrões, o modelo de classes obtido como saída da etapa "Identificar Espalhamento de Interesses" foi refinado, como ilustrado na Figura 5.4. Atributos e operações relacionados a outros padrões que estão na classe `Resource` foram adicionados a esse modelo e marcados com estereótipos UML, como <<QR>>, <<QR-V1>>, <<QR-V2>>, <<QR-V3>> e <<QR-V4>>, que representam o padrão Quantificar Recurso e suas variantes; <<LR>> e <<CR>>, que representam os padrões Locar Recurso e Comercializar Recurso, respectivamente. As classes `SimpleType` e `NestedType`, também pertencem ao padrão Identificar Recurso, mas não possuem atributos e métodos que contribuem para a implementação de outros padrões.

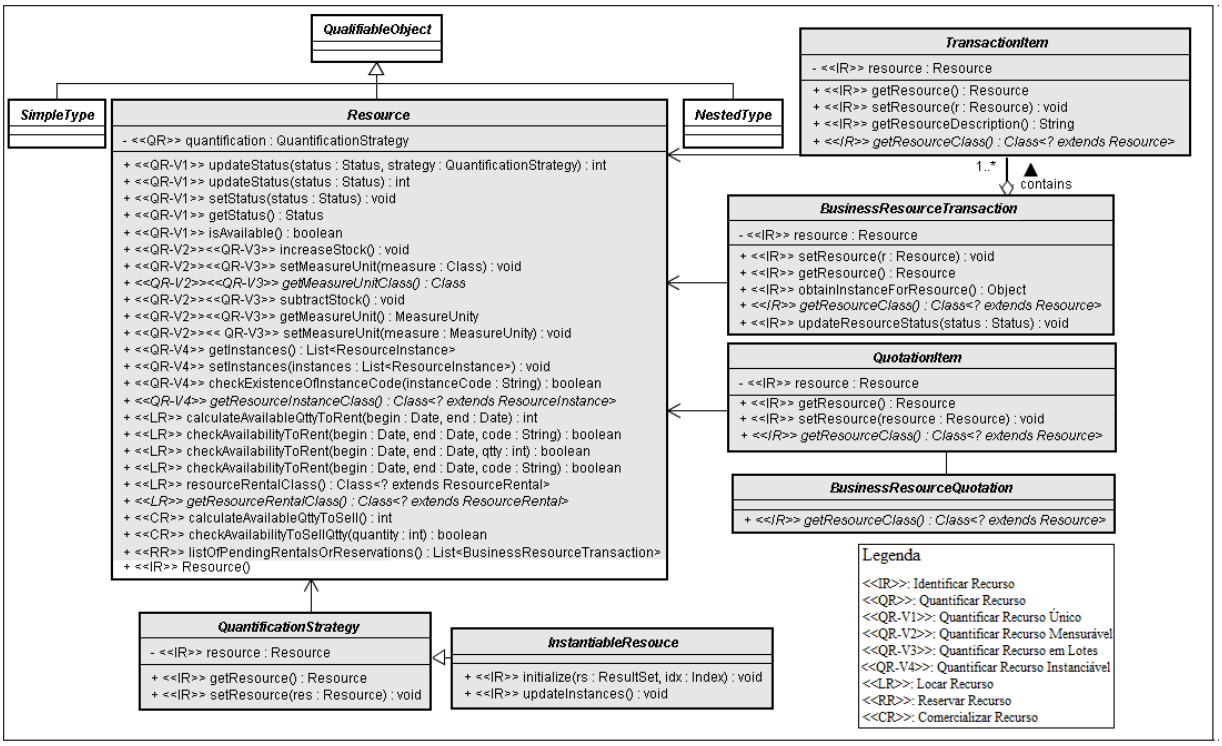

Figura 5.4. Modelo de classes do padrão Identificar Recurso refinado com indícios de entrelaçamento de interesses.

Na quarta etapa do processo – Modularizar com Aspectos –, o objetivo é analisar as marcações que foram feitas nos atributos e métodos das classes do padrão e refatorar o código para aspectos. Para realizar essa etapa, primeiramente foi efetuada a análise do modelo de classes do padrão, gerado na etapa "Identificar Entrelaçamento de Interesses", e do código fonte do *framework*, com o objetivo de identificar o que deve ser modularizado por *inter-type declarations* e *advices* de AspectJ. Em seguida, com base no conhecimento de domínio do *framework*, foi definida uma estratégia para modularizar o padrão com orientação a aspectos. Essa estratégia consiste em criar aspectos para separar o código relacionado ao padrão Identificar Recurso das unidades que implementam outros padrões da GRN e utilizar o idioma *Container Introduction* (HANENBERG *et al.*, 2003) para modularizar os atributos e operações relacionados ao padrão que são comuns às classes `QuantificationStrategy`, `BusinessResourceTransaction`, `QuotationItem` e `TransactionItem`. Os elementos em comum a essas classes são um atributo do tipo `Resource` e os métodos `getResource()` e `setResource()`, como mostrado no modelo de classes da Figura 5.4.

Dessa forma, foi criada uma interface vazia chamada `IIdentifyResource`, que desempenha o papel de *Container* no idioma *Container Introduction*, e um aspecto chamado `IdentifyResourceConcernContainerLoader`, que desempenha o papel de *Container Loader*, como ilustrado na Figura 5.5. Esse aspecto introduz por meio de *inter-type*

declarations, um atributo do tipo `Resource` e os métodos `getResource()` e `setResource()` na interface `IIdentifyResource`.

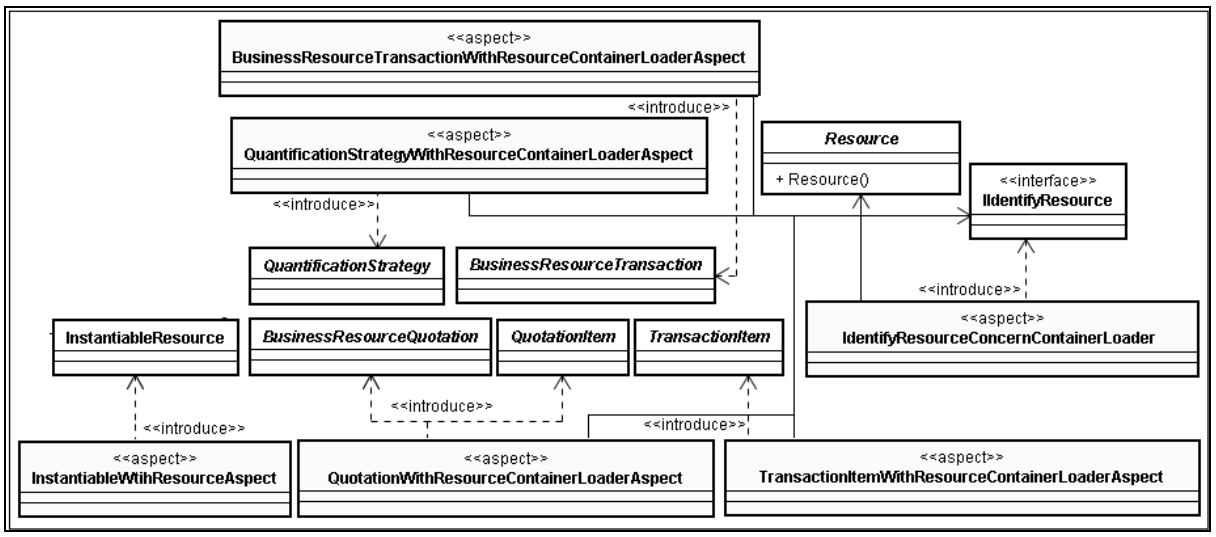

Figura 5.5. Implementação OA do padrão Identificar Recurso.

Os aspectos `BusinessResourceTransactionWithResource-ContainerLoaderAspect`, `QuantificationStrategyWithResourceContainer-LoaderAspect`, `QuotationWithResourceContainerLoaderAspect` e `TransactionItemWithResourceContainerLoaderAspect` desempenham o papel de *Container Connector* no Idioma *Container Introduction*. Esses aspectos vinculam as classes `BusinessResourceTransaction`, `QuantificationStrategy`, `QuotationItem` e `TransactionItem` à interface *Container* (`IIdentifyResource`), por meio da construção "*declare implements*" de AspectJ, de modo que essas classes implementem o comportamento da interface *Container*. Consequentemente, essas classes herdam os atributos e operações relacionados ao padrão Identificar Recurso que o aspecto `IdentifyResourceConcernContainerLoader` introduziu nessa interface.

Seguindo a modularização, os demais atributos e operações relacionados ao padrão Identificar Recurso que estão nas classes `BusinessResourceTransaction`, `BusinessResourceQuotation`, `QuantificationStrategy`, `InstantiableResour-ce`, `QuotationItem` e `TransactionItem` foram removidos das mesmas. Os aspectos `BusinessResourceTransactionWithResourceContainerLoaderAspect`, `Quanti-ficationStrategyWithResourceContainerLoaderAspect`, `QuotationWithRe-sourceContainerLoaderAspect` e `TransactionItemWithResourceContainer-LoaderAspect` encapsulam a implementação desse padrão de todas essas classes.

Para implementar os aspectos procedeu-se da seguinte maneira: foi analisado o quê se torna *crosscutting* estático, em que atributos e operações relacionados ao padrão Identificar Recurso foram removidos de classes do *framework* que implementam outros padrões e novamente introduzidos por aspectos utilizando *inter-type declarations*. Em seguida, foi analisado o quê se torna *crosscutting* dinâmico, ou seja, trechos de código relacionados ao padrão Identificar Recurso dentro de operações que implementam outros padrões. Como por exemplo, nos construtores da classe `TransactionItem`, que implementa o padrão Itemizar Transação do Recurso da GRN, há código relacionado ao Identificar Recurso. Esse código foi removido dessa classe e encapsulado no aspecto `TransactionItemWith-ResourceContainerLoaderAspect`, que adiciona esse comportamento à `TransactionItem` por meio de *advices* de AspectJ.

Com a remoção do espalhamento o próximo passo foi remover o entrelaçamento, ou seja, separar os interesses inerentes aos padrões Quantificar Recurso, Locar Recurso, Reservar Recurso e Comercializar Recurso da classe `Resource`. Os interesses relacionados a esses padrões foram modularizados em outras iterações do processo. Assim, os atributos e operações da classe `Resource`, que estão sombreados na Figura 5.4, foram removidos e alocados em aspectos. Dessa forma, após a modularização a classe `Resource` contém apenas os atributos e operações relacionados ao padrão Identificar Recurso, como ilustrado na Figura 5.5.

5.2.2 Quantificar Recurso

O padrão Quantificar Recurso fornece quatro maneiras de quantificar os recursos de negócio de uma aplicação: a) em que o recurso pode ser tratado como único, em situações em que não é necessário ter o controle de uma instância específica do recurso, como um carro que passa por uma manutenção; b) instanciável, em situações em que é necessário ter o controle de uma instância específica do recurso, como em sistema de biblioteca, em que um livro (o recurso) contém várias cópias; c) mensurável, em situações em que é necessário lidar com quantidades específicas de recurso; ou d) em lotes, em situações em que o recurso necessita ser tratado como lotes (BRAGA *et al.*, 1999).

Na implementação dessas variantes no GRENJ-OO há entrelaçamento de interesses, em que cada uma dessas variantes está altamente acoplada uma com a outra. Outro problema desse padrão no *framework* é que sua implementação está espalhada por unidades que implementam o padrão Identificar Recurso.

Para resolver esses problemas na primeira etapa do processo – Identificar as Unidades Originais do Interesse – foram analisados a descrição textual e o modelo de classes do padrão, extraídos da linguagem de padrões GRN (BRAGA *et al.*, 1999). Com base nessas informações, o código fonte do *framework* foi analisado com o objetivo de identificar as unidades que foram originalmente projetadas para implementar o padrão Quantificar Recurso. Como resultado, foi verificado que as classes `QuantificationStrategy`, `SingleResource`, `MeasurableResource`, `InstantiableResource`, `MeasureUnity` e `ResourceInstance` foram originalmente projetadas para implementar esse padrão. Assim como na modularização do padrão Identificar Recurso, não houve dificuldades na realização desta etapa porque as informações presentes na descrição textual e no modelo de classes do padrão foram suficientes, deixando óbvio que essas classes foram originalmente projetadas para implementá-lo.

Na segunda etapa – Identificar Espalhamento de Interesses – as unidades originais do padrão, a descrição textual e o modelo de classes do padrão foram analisados para extrair um conjunto de palavras-chave que representam indícios de interesses relacionados ao padrão Quantificar Recurso e suas variantes. Com base nessas informações, o código fonte do *framework* foi analisado para identificar atributos e operações de suas classes que contribuem para a implementação do padrão e de suas variantes. Além das unidades originais, foi verificada a presença de código relacionado ao padrão Quantificar Recurso na classe `Resource`, que foi originalmente projetada para implementar o padrão Identificar Recurso.

Durante a realização desta etapa foram encontrados indícios de interesses relacionados a todas as variantes do padrão Quantificar Recurso na classe `QuantificationStrategy`. Isso faz com que as classes `SingleResource`, `MeasurableResource` e `InstantiableResource`, que a estendem, e que respectivamente implementam as variantes Recurso Único, Recurso Mensurável e em Lotes e Recurso Instanciável, implementem métodos relacionados a todas as variantes do padrão, sendo que cada classe foi exclusivamente projetada para implementar uma única variante. Isso denota que as implementações das variantes do padrão Quantificar Recurso estão espalhadas por essas classes.

Após a análise do código fonte do *framework* em busca de indícios relacionados ao padrão Quantificar Recurso, foi elaborado o modelo de classes do padrão a partir das unidades originais e de outras unidades do *framework* que contêm indícios do padrão, como mostrado na Figura 5.6. Os atributos e operações relacionados ao padrão Quantificar Recurso e suas

variantes que estão na classe `Resource`, que implementa o Identificar Recurso, e as classes `QuantificationStrategy`, `SingleResource`, `MeasurableResource` e `InstantiableResource`, `MeasureUnity` e `ResourceInstance`, foram marcados com os estereótipos <<QR>>, <<QR-V1>>, <<QR-V2>>, <<QR-V3>> e <<QR-V4>>, que respectivamente representam o padrão Quantificar Recurso e suas variantes Recurso Único, Recurso Mensurável, Recurso em Lotes e Recurso Instanciável.

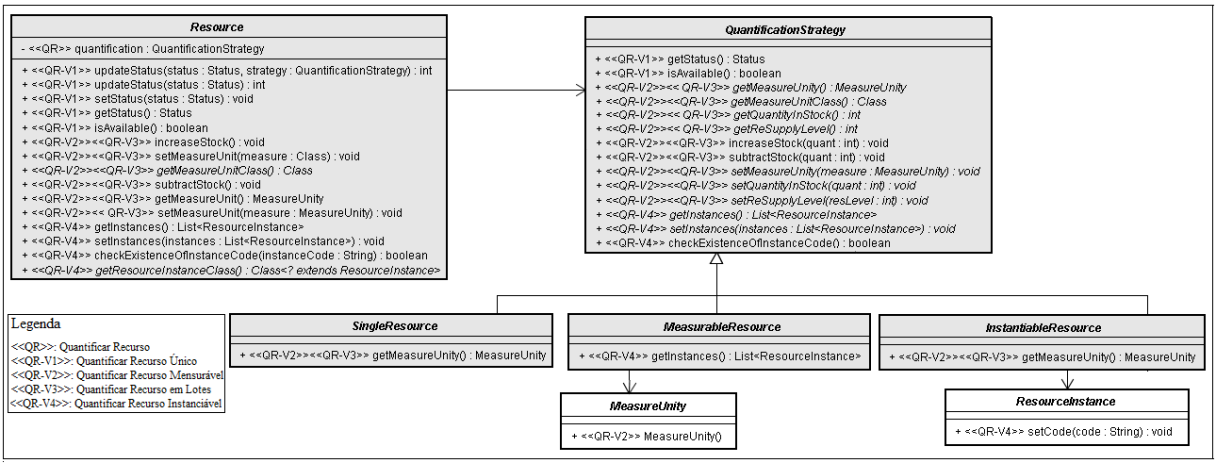

Figura 5.6. Modelo de classes do padrão Quantificar Recurso com indícios de espalhamento de interesses.

Na terceira etapa – Identificar Entrelaçamento de Interesses –, com base no conhecimento de domínio, na lista de unidades originais do padrão e nas palavras-chave que representam indícios de interesses relacionados ao padrão (obtidas na etapa 2), cada uma das unidades originais do padrão foi analisada em busca de atributos e operações não relacionados ao padrão. Essa análise se iniciou pela classe `QuantificationStrategy`, seguida das classes `SingleResource`, `MeasurableResource` e `InstantiableResource`. Foi observado nessas classes a presença de operações relacionadas aos padrões Locar Recurso e Comercializar Recurso, como `calculateAvailabilityToRent()` e `calculateAvailabilityToSell()`.

Após a análise do código fonte das unidades originais do padrão, o modelo de classes obtido como saída da etapa "Identificar Espalhamento de Interesses" foi refinado, como ilustrado na Figura 5.7. Atributos e operações relacionados a outros padrões que estão nas classes `QuantificationStrategy`, `SingleResource`, `MeasurableResource` e `InstantiableResource` foram adicionados a esse modelo e marcados com os estereótipos <<LR>> e <<CR>>, que respectivamente representam os padrões Locar Recurso e Comercializar Recurso. As classes `MeasuryUnity` e `ResourceInstance`, também

pertencem ao padrão Quantificar Recurso, mas não possuem atributos e métodos que contribuem para a implementação de outros padrões.

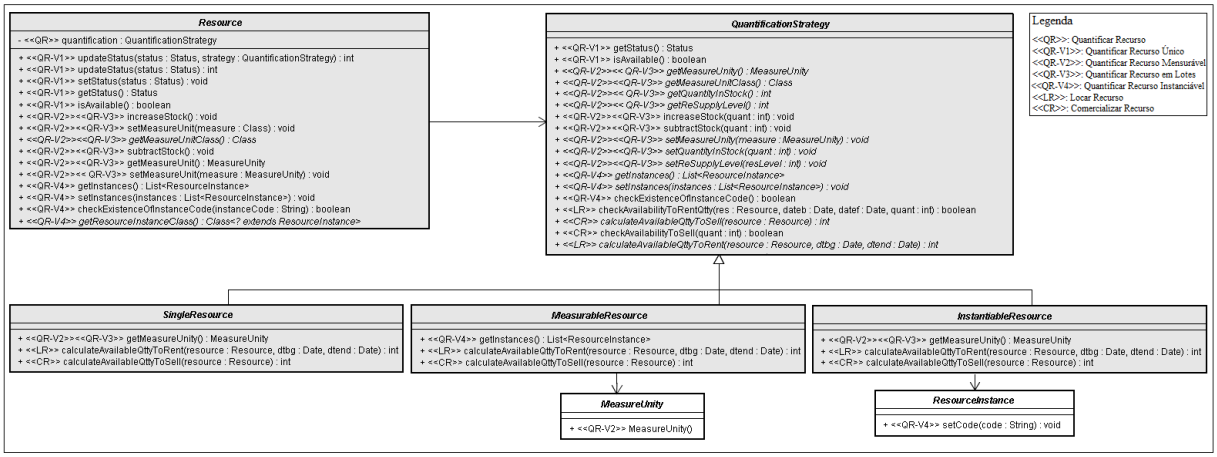

Figura 5.7. Modelo de classes do padrão Quantificar Recurso refinado com indícios de entrelaçamento de interesses.

Na quarta etapa do processo – Modularizar com Aspectos –, foi efetuada a análise do modelo de classes do padrão, gerado na etapa "Identificar Entrelaçamento de Interesses", e do código fonte do *framework*, com objetivo de identificar o que deve ser modularizado por meio de *inter-type declarations* e o que deve ser modularizado com *advices* de AspectJ. Em seguida, com base no conhecimento de domínio do *framework*, foi definida uma estratégia para modularizar o padrão com aspectos. Essa estratégia consiste em criar um aspecto para separar o código relacionado ao padrão Quantificar Recurso da classe `Resource` e criar aspectos para modularizar a implementação de cada variante desse padrão, como mostrado na Figura 5.8.

Para separar o código relacionado ao padrão Quantificar Recurso da classe `Resource`, esse código foi removido dessa classe e encapsulado no aspecto `ResourceWithQuantificationContainerLoaderAspect`, que novamente introduz por *inter-type declarations*, atributos e operações relacionados ao padrão Quantificar Recurso em `Resource`. Esse aspecto também utiliza *advices* para adicionar comportamento relacionado ao padrão Quantificar Recurso nessa classe.

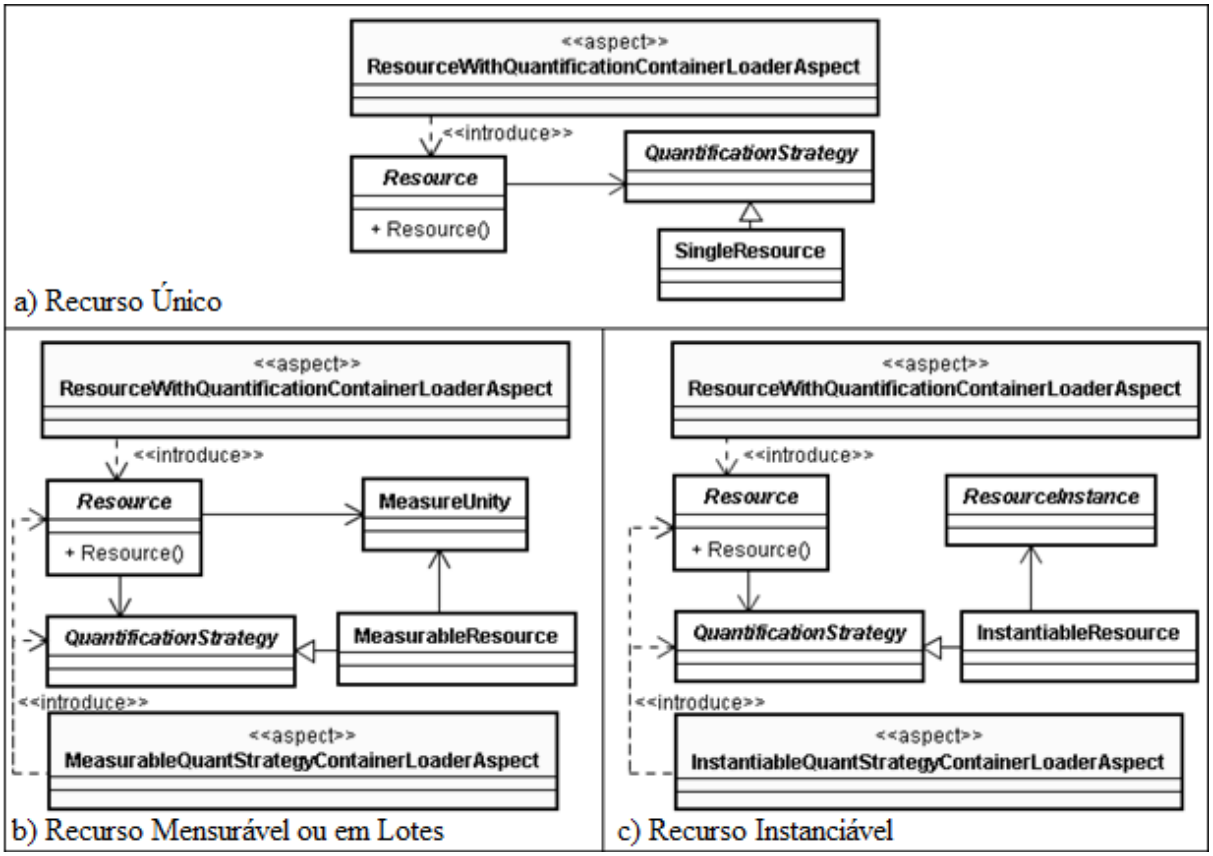

Figura 5.8. Implementação OA do padrão Quantificar Recurso.

Para modularizar as variantes do padrão Quantificar Recurso, foram criados dois aspectos, um para modularizar a implementação da variante Recurso Mensurável ou em Lotes e outro para a variante Recurso Instanciável. Em virtude da variante Recurso Único não possuir métodos que são exclusivamente relacionados à sua implementação não foi necessário criar um aspecto para modularizá-la, como mostrado na Figura 5.8a). O aspecto `ResourceWithQuantificationContainerLoaderAspect`, que encapsula a implementação do padrão Quantificar Recurso da classe `Resource`, é comum a todas as variantes do padrão.

Na implementação OA da variante Recurso Mensurável ou em Lotes, mostrada na Figura 5.8b), os métodos relacionados a essa variante foram removidos das classes `Resource` e `QuantificationStrategy`, e foram encapsulados no aspecto `MeasurableQuant-StrategyContainerLoaderAspect`, que introduz, por meio de *inter-type declarations*, métodos relacionados a essa variante, como `getMeasureUnityClass()` e `getMeasureUnity()`, nessas classes. Isso evita que essa variante fique entrelaçada com a implementação de outros padrões e as demais variantes desse padrão.

Análogo à implementação OA da variante Recurso Mensurável ou em Lotes, na implementação OA da variante Recurso Instanciável (Figura 5.8c)), os métodos relacionados

a essa variante também foram removidos das classes `Resource` e `QuantificationStrategy`, e foram encapsulados no aspecto `InstantiableQuantStrategyContainerLoaderAspect`, que introduz métodos relacionados a essa variante, como `getResourceInstanceClass()` e `getInstances()`, nessas classes.

Os métodos relacionados aos padrões Locar Recurso e Comercializar Recurso que estão nas classes `QuantificationStrategy`, `SingleResource`, `MeasurableResource` e `InstantiableResource`, foram modularizados em outras iterações do processo.

Após a modularização do padrão Quantificar Recurso, somente o código relacionado à variante selecionada é embutido no código da aplicação, evitando a presença de código relacionado a variantes não utilizadas. Com isso, o gerenciamento das variabilidades desse padrão é melhorado.

5.2.3 Locar Recurso

O padrão Locar Recurso fornece uma solução para tratar de transações de aluguel que envolvem recursos de negócio, que podem ser bens emprestados a um cliente por um período ou serviços efetuados por um especialista por determinado período de tempo (BRAGA *et al*, 1999). Na forma que esse padrão está implementado no *framework* GRENJ-OO não há um conjunto coeso de unidades de implementação que o representa. Além disso, suas variantes estão altamente acopladas entre si, fazendo com que o código relacionado a elas seja embutido na aplicação independentemente se são utilizadas ou não.

Para modularizar a implementação desse padrão, na primeira etapa do processo – Identificar as Unidades Originais do Interesse – primeiramente foram analisados a descrição textual e o modelo de classes do padrão, extraídos da linguagem de padrões GRN (BRAGA *et al.*, 1999). Com base nessas informações, foi efetuada a análise do código fonte do *framework* com o objetivo de identificar as unidades que foram originalmente projetadas para implementar o padrão Locar Recurso. Como resultado, foi verificado que as classes `ResourceRental` e `BusinessResourceTransaction` foram projetadas para implementar esse padrão.

Na segunda etapa – Identificar Espalhamento de Interesses – as unidades originais do padrão, a descrição textual e o modelo de classes do padrão foram analisados para extrair um conjunto de palavras-chave que representam indícios de interesses relacionados ao padrão Locar Recurso. Com base nessas informações, o código fonte do *framework* foi analisado para

identificar atributos e operações de suas classes que contribuem para a implementação do padrão. Além das classes `ResourceRental` e `BusinessResourceTransaction`, foi verificada a presença de código relacionado ao Locar Recurso em classes como: `Resource`, `QuantificationStrategy`, `SingleResource`, `MeasurableResource` e `InstantiableResource`, que respectivamente implementam os padrões Identificar Recurso e Quantificar Recurso. Métodos relacionados ao padrão Locar Recurso como `calculateAvailabilityToRent()` e `calculateAvailabilityToRentQtty()` se encontram espalhados por essas classes.

Também foi verificado nesta etapa que na classe `BusinessResourceTransaction`, que é uma classe comum a todos os padrões de transação da GRN, há operações como `getBeginDate()` e `getReturnDate()`, que são exclusivas dos padrões Locar Recurso e Reservar Recurso, ou seja, necessárias somente quando um desses padrões é utilizado. Além disso, analisando a classe `ResourceRental` do padrão Locar Recurso foi constatado que a implementação da variante Locação do Recurso com Multa desse padrão está altamente acoplada a ele. Isso faz com o que o código relacionado a essa variante seja embutido no código da aplicação independentemente se é utilizada ou não. Com isso, para utilizar o padrão, o engenheiro de aplicação necessita fornecer implementações vazias para métodos relacionados a essa variante quando ela não é requerida pela aplicação.

Após a análise do código fonte do *framework* em busca de indícios relacionados ao padrão Locar Recurso, foi elaborado o modelo de classes do padrão a partir das unidades originais e de outras unidades do *framework* que contêm indícios do padrão, como mostrado na Figura 5.9. Os atributos e operações relacionados ao padrão Locar Recurso que estão em classes que implementam outros padrões, como `Resource` e `QuantificationStrategy`, foram marcados com o estereótipo <<LR>> nesse modelo de classes. Em virtude das operações da classe `BusinessResourceTransaction` serem comuns aos padrões Locar e Recurso e Reservar Recurso, foram marcadas com ambos os estereótipos <<LR>> e <<RR>>.

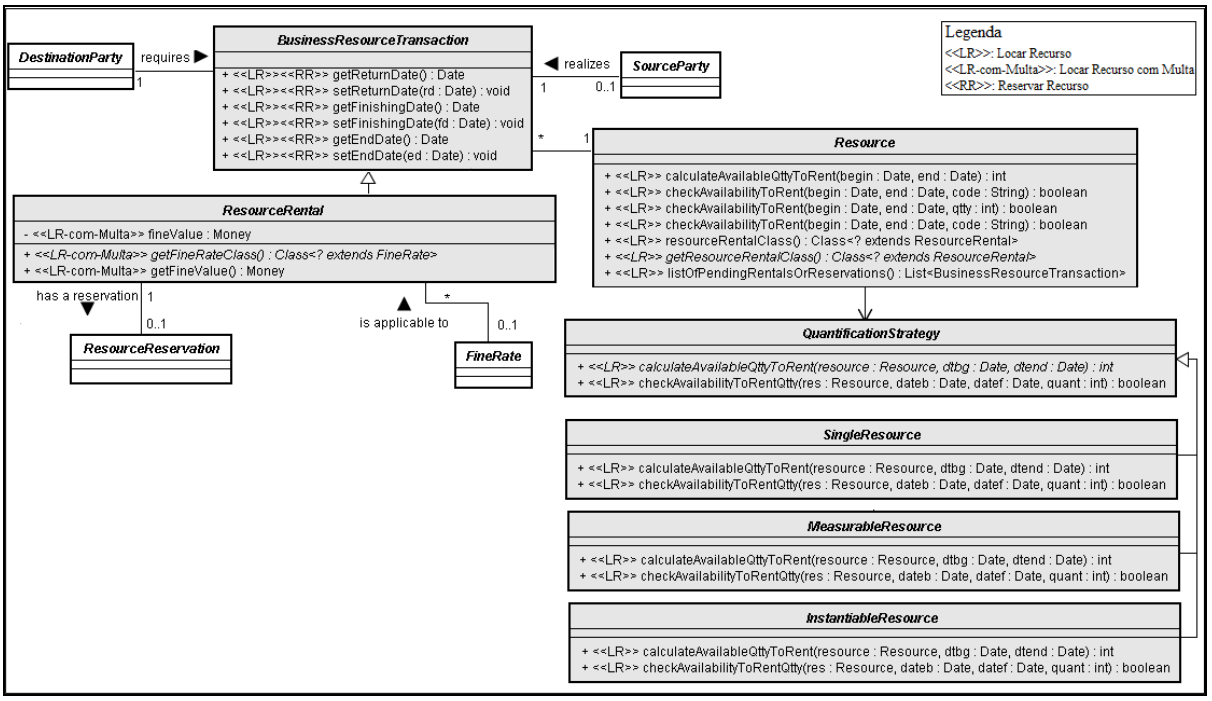

Figura 5.9. Modelo de classes do padrão Locar Recurso com indícios de espalhamento de interesses.

Na terceira etapa – Identificar Entrelaçamento de Interesses –, a partir do conhecimento de domínio, da lista de unidades originais do padrão e das palavras-chave que representam indícios de interesses relacionados ao padrão (obtidas na etapa 2), cada uma das unidades originais do padrão foi analisada em busca de atributos e operações não relacionados ao padrão. Dessa análise foi verificado que a classe `ResourceRental` contém, além dos atributos e operações necessários para implementar o padrão Locar Recurso, atributos e operações relacionados aos padrões Reservar Recurso, Comercializar Recurso e Itemizar Transação do Recurso, como os métodos `getReservationNumber()`, `getSaleNumber()` e `getReturnDateFromItemTransaction()`.

Após a análise do código fonte das unidades originais do padrão, o modelo de classes obtido como saída da etapa "Identificar Espalhamento de Interesses" foi refinado, como ilustrado na Figura 5.10. Atributos e operações relacionados a outros padrões que estão na classe `ResourceRental` foram adicionados a esse modelo e marcados com estereótipos, como <<RR>>, <<CR>> e <<ITR>>, que respectivamente representam os padrões Reservar Recurso, Comercializar Recurso e Itemizar Transação do Recurso.

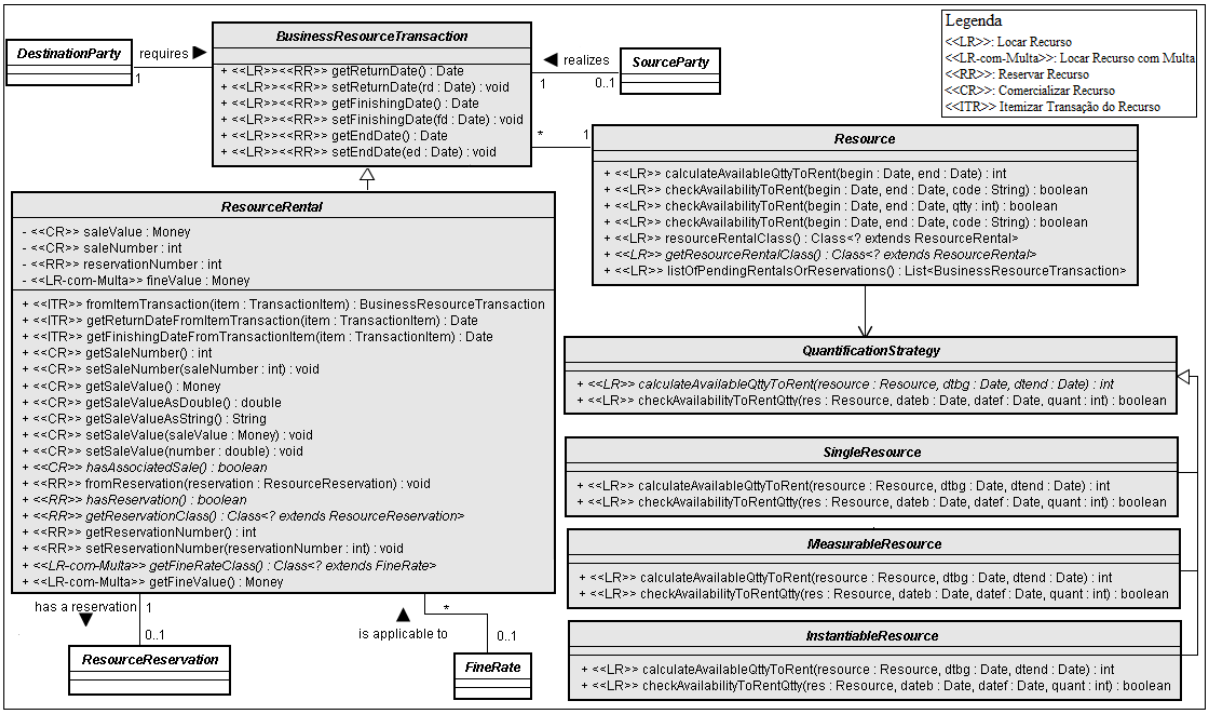

Figura 5.10. Modelo de classes do padrão Locar Recurso refinado com indícios de entrelaçamento de interesses.

Na quarta etapa do processo – Modularizar com Aspectos –, foi efetuada a análise do modelo de classes do padrão, gerado na etapa "Identificar Entrelaçamento de Interesses", e do código fonte do *framework*, com objetivo de identificar o que deve ser modularizado por meio de *inter-type declarations* e o que deve ser modularizado com *advices* de AspectJ. Em seguida, com base no conhecimento de domínio do *framework*, foi definida uma estratégia para modularizar o padrão com aspectos. Essa estratégia consiste em criar um aspecto para cada variante do padrão Quantificar Recurso, para remover o código relacionado a Locar Recurso das variantes desse padrão, um aspecto para separar a implementação do Locar Recurso das classes `Resource` e `QuantificationStrategy`, um aspecto para separar a implementação da variante Locação do Recurso com Multa do padrão Locar Recurso e um aspecto para separar as operações comuns aos padrões Locar Recurso e Reservar Recurso de `BusinessResourceTransaction`, que é uma classe comum a todos os padrões de transação da GRN implementados no GRENJ-OO, como mostrado na Figura 5.11.

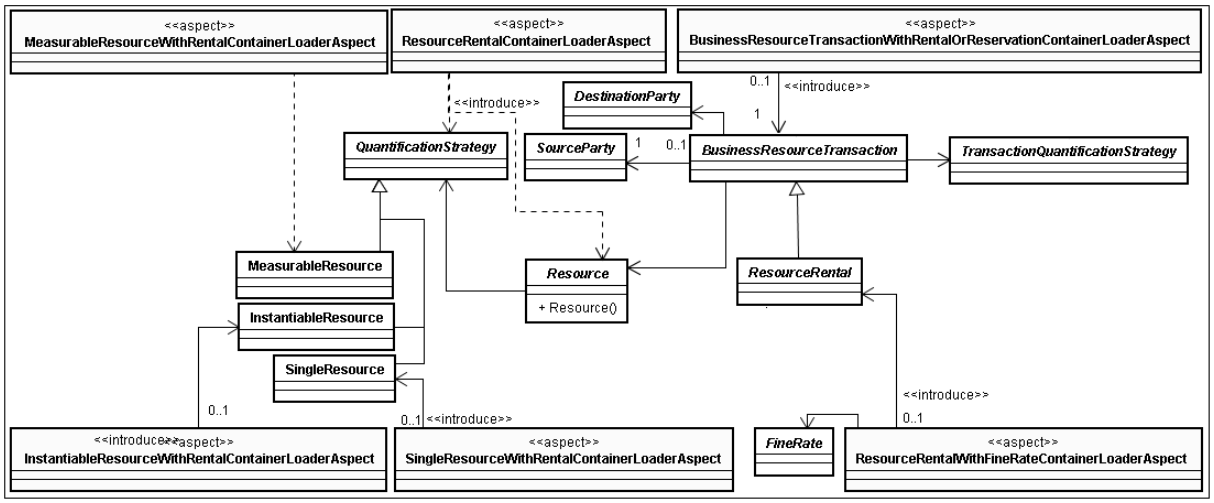

Figura 5.11. Implementação OA do padrão Locar Recurso.

Os aspectos `ResourceRentalContainerLoaderAspect`, `SingleResourceWithRentalContainerLoaderAspect`, `MeasurableResourceWithRentalContainerLoaderAspect` e `InstantiableResourceWihRentalContainerLoaderAspect` introduzem, por meio de *inter-type declarations*, métodos relacionados ao padrão Locar Recurso nas classes `Resource`, `QuantificationStrategy`, `SingleResource`, `MeasurableResource` e `InstantiableResource`, que respectivamente implementam os padrões Identificar Recurso e Quantificar Recurso. Dessa forma, evita-se o espalhamento de interesses relacionados a Locar Recurso por essas classes. Isso contribui para a melhoria da separação de interesses e da coesão das unidades que implementam esses padrões.

O aspecto `ResourceRentalWithFineRateContainerLoaderAspect` encapsula a implementação da variante Locação do Recurso com Multa do padrão Locar Recurso. Esse aspecto introduz por *inter-type declarations* métodos relacionados a essa variante, como `getFineValue()` e `getFineRateClass()`, na classe `ResourceRental`. Dessa forma, a implementação dessa variante foi desacoplada do padrão Locar Recurso. Para utilizá-la, basta que esse aspecto seja adicionado à aplicação.

O aspecto `BusinessResourceTransactionWithRentalOrReservationContainerLoaderAspect` encapsula um conjunto de operações comuns aos padrões Locar Recurso e Reservar Recurso, evitando que essas operações sejam embutidas no código da aplicação sem que nenhum desses padrões seja aplicado.

Com a remoção do espalhamento de interesses relacionados ao padrão Locar Recurso de unidades do *framework* que implementam outros padrões, o próximo passo foi remover o entrelaçamento, ou seja, separar os interesses inerentes aos padrões Reservar Recurso, Comercializar Recurso e Itemizar Transação do Recurso da classe `ResourceRental`. Esses

padrões foram modularizados em outras iterações do processo. Dessa forma, os métodos de `ResourceRental`, que estão sombreados na Figura 5.10, foram removidos e alocados em aspectos.

5.3 Considerações Finais

A forma como os padrões da GRN estavam implementados no *framework* GRENJ-OO apresentava entrelaçamento e espalhamento dos interesses. Há classes do *framework* que contêm código relacionado a vários padrões, o que denota baixa coesão. Além disso, há forte relacionamento de dependência entre essas classes, fazendo com que toda aplicação baseada nesse *framework* carregue todas as suas classes, independentemente de serem utilizadas ou não. Em conseqüência desses problemas, a manutenibilidade, a reusabilidade e o gerenciamento de variabilidades do *framework* são prejudicados.

Com a finalidade de resolver esses problemas, os padrões da GRN implementados no *framework* GRENJ-OO foram modularizados com orientação a aspectos, removendo o entrelaçamento e o espalhamento de interesses relacionados a cada padrão. As variabilidades inerentes a cada padrão também foram modularizadas, sendo encapsuladas em aspectos. Dessa forma, as aplicações construídas com base nesse *framework* não mais necessitam carregar todas as suas *features*, mas somente as necessárias para satisfazer aos requisitos da aplicação.

Na modularização do *framework* GRENJ-OO o uso do mecanismo de *inter-type declarations* de AspectJ teve predominância sobre o uso do mecanismo de *advices*, como mostrado na Tabela 5.1. Isso ocorreu em virtude de que no GRENJ-OO havia grande quantidade de declarações de atributos e operações (*crosscutting* estático) implementadas com entrelaçamento e espalhamento de interesses. A utilização dos *advices* foi menor que a do *inter-type declarations* em conseqüência da menor incidência de *crosscutting* dinâmico.

Tabela 5.1. Números relacionados aos mecanismos utilizados na modularização do *framework* GRENJ-OO.

Mecanismos	*Inter-type Declarations* (*Crosscutting* Estático)	*Advices* (*Crosscutting* Dinâmico)	Total
Valor numérico	306	153	459
Porcentagem	66,67%	33,33%	100%

Nessa nova versão do *framework* GRENJ, denominada GRENJ-OA, cada padrão da GRN e suas variantes estão distribuídos em uma estrutura de pacotes e encapsulados em aspectos. Para utilizar determinado padrão ou variante basta que o pacote correspondente ao mesmo seja adicionado ao projeto. Isso contribui para a melhoria dos níveis de separação de interesses, para a redução do acoplamento e o aumento da coesão das classes do GRENJ. Como conseqüência, há melhoria da manutenibilidade, da reusabilidade e do gerenciamento de variabilidades desse *framework*.

Um ponto negativo do uso da orientação a aspectos na modularização do *framework* GRENJ-OO é quanto ao aumento do número de suas unidades (classes, interfaces e aspectos) e, conseqüentemente, o aumento de sua complexidade. Para facilitar a manutenção e utilização do GRENJ-OA foi elaborado um diagrama de *features* (Apêndice A), que fornece a visão geral de como os padrões da GRN e suas variantes estão implementados. Também foi elaborada uma tabela de mapeamento (Apêndice B) que relaciona cada padrão e suas variantes com as unidades do GRENJ-OA que os implementam.

Para verificar a efetividade do uso de abstrações orientadas a aspectos na modularização dos padrões da GRN e de suas variantes no GRENJ-OO, uma avaliação quantitativa com base em métricas de separação de interesses, acoplamento, coesão e tamanho foi realizada. Os resultados dessa avaliação estão apresentados no próximo capítulo.

Capítulo 6

AVALIAÇÃO QUANTITATIVA DAS VERSÕES GRENJ-OO E GRENJ-OA

6.1 Considerações Iniciais

GRENJ-OA é a versão orientada a aspectos obtida após a modularização do GRENJ-OO (DURELLI, 2008). Para verificar a efetividade da orientação a aspectos nesse contexto é necessário realizar avaliações quantitativas, como as apresentadas por Garcia *et al.* (2005) e Cacho *et al.* (2006), que avaliaram o uso da OA na implementação de padrões de projeto, e Garcia *et al.* (2004), que avaliaram o uso da OA no contexto de sistemas multiagentes.

Para facilitar a compreensão dos resultados obtidos desta avaliação este capítulo está organizado em quatro seções além desta. Na Seção 6.2 os procedimentos de avaliação adotados são descritos. Na Seção 6.3 os resultados da avaliação para cada padrão da GRN são apresentados. Na Seção 6.4 é mostrada a análise dos resultados. Na Seção 6.5 são apresentadas as considerações finais.

6.2 Procedimentos de Avaliação

Para assegurar uma avaliação coerente, as implementações orientadas a objetos e orientadas a aspectos de cada um dos padrões da GRN, nas versões GRENJ-OO e GRENJ-OA, implementam as mesmas funcionalidades. As diferenças existentes entre essas versões

são quanto ao aumento do número de classes, interfaces e a adição de aspectos, alterações na implementação de alguns métodos e a remoção de atributos e métodos.

As métricas de separação de interesses, acoplamento, coesão e tamanho definidas por Sant'Anna *et al.* (2003) foram utilizadas na avaliação quantitativa dos *frameworks* utilizados nesta dissertação por terem sido objeto de estudo em outros trabalhos, como os citados anteriormente e apresentarem resultados significativos.

A coleta dos resultados das métricas de separação de interesses considera cada padrão da GRN como um interesse. Nos diagramas de classes referentes a cada padrão foi feito o sombreamento do que não está relacionado a este padrão. Foi feita a análise de cada diagrama para identificar as classes, interfaces, atributos e métodos que contribuem para a implementação de outros padrões. Por exemplo, no diagrama de classes da versão OO do padrão Identificar Recurso (Figura 5.4), há atributos e métodos da classe `Resource` que implementam outros padrões. Após essa etapa, cada um desses atributos e operações foi associado a um estereótipo da UML, que representa o padrão ou variante que cada um implementa. Isso facilita a visualização do nível de entrelaçamento e espalhamento de interesses de cada um dos padrões da GRN. Para a análise do código fonte das classes do GRENJ-OO, também foi adotado o mesmo procedimento. Em seguida, os dados das métricas de separação de interesses (CDC, CDO e CDLOC) (SANT'ANNA *et al.*, 2003) foram manualmente coletados. Por exemplo, na classe `Resource`, que implementa o padrão Identificar Recurso, há código relacionado a vários padrões, como ilustrado na Figura 6.1. Cada fragmento de código relacionado a outros padrões que não seja o Identificar Recurso está sombreado.

```
01 package grenj.model;
02
03 public abstract class Resource extends QualifiableObject {
04
05    private QuantificationStrategy quantification;
06
07    public Resource() { … }
08
09    public boolean isAvailiable() { … }
10
11    public abstract Class<? extends MeasureUnity>
12        getMeasureUnityClass();
13
14    public abstract Class<? extends ResourceInstance>
15        getResourceInstanceClass();
16
17    public abstract Class<? extends ResourceRental>
18        getResourceRentalClass();
19
20    public boolean checkAvailabilityToSellQtty(int qtty) { … }
21
22    public int save() { … }
23
24 }
```

Figura 6.1. Trecho de código sombreado da versão OO da classe Resource.

Os resultados das métricas de acoplamento, coesão e tamanho foram coletados por meio da análise dos diagramas de classes e do código fonte das classes das versões OO e OA dos padrões da GRN implementados no *framework* GRENJ. Após essa análise, os dados das métricas de acoplamento, coesão e tamanho foram manualmente coletados.

A ferramenta de modelagem Astah (CHANGE VISION, 2010) foi utilizada na elaboração dos modelos de projeto de cada um dos padrões da GRN, aplicando a técnica de sombreamento e o uso de estereótipos da UML para associar classes, interfaces, atributos e operações aos padrões e variantes que implementam. Para a análise de código fonte das classes do GRENJ-OO, a ferramenta Eclipse (THE ECLIPSE FOUNDATION, 2010), juntamente com o AspectJ *plugin* versão 1.5 (ASPECTJ PROJECT, 2010) foram utilizados. Essas ferramentas facilitam a visualização tanto do código OO quanto do código OA das unidades do *framework* GRENJ.

6.3 Resultados

A apresentação dos resultados da aplicação das métricas nas versões OO e OA do *framework* GRENJ está dividida em três partes: na primeira são apresentados os resultados

para as métricas de separação de interesses, acoplamento, coesão e tamanho para os padrões do grupo 1: Identificação do Recurso de Negócio (padrões Identificar Recurso e Quantificar Recurso); na segunda parte, os resultados da aplicação dessas métricas nos padrões do grupo 2: Transações de Negócio (Locar Recurso, Reservar Recurso, Comercializar Recurso, Cotar Recurso, Conferir a Entrega do Recurso, Manter Recurso e Cotar a Manutenção) são apresentados; e na terceira parte são mostrados os resultados da aplicação das métricas nos padrões do grupo 3: Detalhes das Transações de Negócio (Itemizar Transação do Recurso, Pagar pela Transação do Recurso, Identificar o Executor da Transação, Identificar Tarefas de Manutenção e Identificar Peças de Manutenção).

A utilização das métricas de separação de interesses tem o objetivo de verificar a efetividade da orientação a aspectos na separação de interesses relacionados a cada um dos padrões da GRN e de suas variantes. As métricas de acoplamento, coesão e tamanho têm o objetivo de avaliar o impacto do uso da orientação a aspectos nos valores dessas métricas para cada padrão e variante da GRN. Em virtude de o *framework* GRENJ-OO proporcionar um grande número de variabilidades, foi selecionada uma variabilidade associada a cada padrão da GRN para a comparação entre suas versões OO e OA, exceto para os padrões Identificar Recurso e Quantificar Recurso da GRN, em que foram selecionadas para comparação duas variantes do primeiro e três do último.

Gráficos são utilizados na apresentação dos resultados da comparação entre as versões OO e OA das variantes de cada padrão da GRN. O eixo Y de cada gráfico representa os valores absolutos obtidos pela aplicação das métricas. Em cada par de barras, correspondente aos valores obtidos nas versões OO e OA das variantes dos padrões, está anexado um valor de porcentagem, que representa a diferença entre os resultados dessas versões. A porcentagem positiva significa que a versão OA foi superior, enquanto que a negativa significa que a versão OA foi inferior.

6.3.1 Grupo 1: Identificação do Recurso de Negócio

Os resultados das métricas de separação de interesses, acoplamento, coesão e tamanho são apresentados considerando as variantes Recurso com Tipo Simples e Recurso com Tipo Aninhado do padrão Identificar Recurso e as variantes Recurso Único, Recurso Mensurável ou em Lotes e Recurso Instanciável do padrão Quantificar Recurso. A apresentação dos resultados está separada para as métricas de separação de interesses e para as de acoplamento, coesão e tamanho.

6.3.1.1 Métricas de Separação de Interesses

Para as variantes Recurso com Tipo Simples e Recurso com Tipo Aninhado do padrão Identificar Recurso, os resultados para a métrica CDC foram 80% superiores para OA em relação às versões OO dessas variantes, como ilustrado no gráfico da Figura 6.2. Isso ocorre em virtude da remoção de código relacionado a este padrão de classes que implementam os padrões Quantificar Recurso, os padrões de transação e Itemizar Transação do Recurso. Com relação à CDO, o uso de aspectos na implementação dessas variantes foi respectivamente 86,3% e 90% superior às suas versões OO. Isso se deve à remoção do espalhamento de interesses relacionados ao padrão Identificar Recurso pelas classes do GRENJ-OO que implementam outros padrões. Com isso, os resultados de CDLOC para as versões OA das variantes desse padrão foram 100% superiores em relação à OO. Isso ocorre em virtude da ausência de pontos de transição de interesse pelas unidades modulares do GRENJ-OA que implementam esse padrão.

Esses resultados se justificam pela capacidade de quantificação da orientação a aspectos que propiciou a remoção de métodos relacionados ao padrão Identificar Recurso, como `getResource()` e `setResource()`, de unidades que contribuem para a implementação de outros padrões. O uso de orientação a aspectos desacoplou a implementação dos elementos opcionais e alternativos do *framework* GRENJ-OO. Assim, a seleção de qualquer uma das variantes do padrão Identificar Recurso inclui somente as unidades relacionadas ao padrão, ao contrário do que ocorre com a versão OO, que inclui o código fonte de unidades relacionadas a outros padrões que contêm indícios de interesses relacionados ao Identificar Recurso.

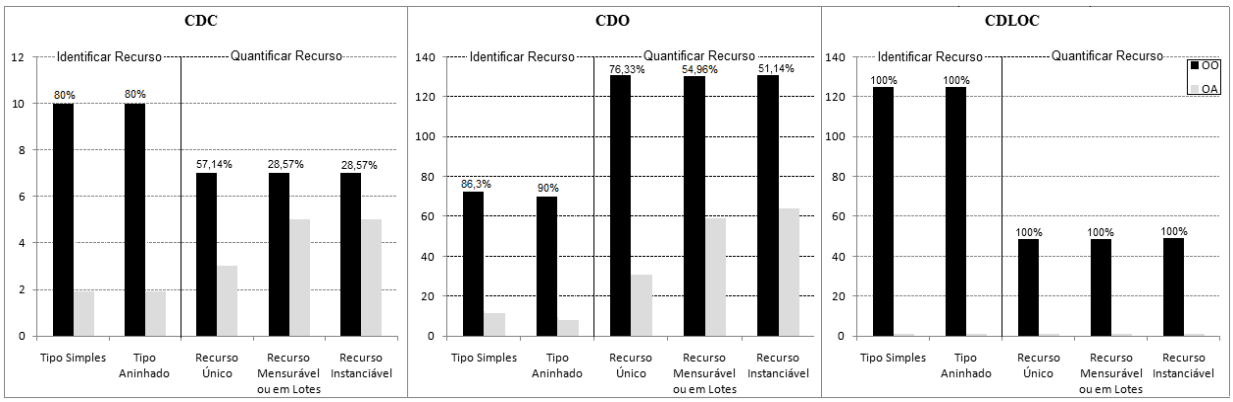

Figura 6.2. Gráfico com os resultados das métricas de separação de interesses para as variantes dos padrões Identificar Recurso e Quantificar Recurso.

Com relação às variantes Recurso Único, Recurso Mensurável ou em Lotes e Recurso Instanciável do padrão Quantificar Recurso, os resultados para a métrica CDC mostraram que

as versões OA dessas variantes foram respectivamente 57,14%, 28,57% e 28,57% superiores em relação às suas versões OO, como ilustrado na Figura 6.2. Isso se justifica pela remoção do entrelaçamento e do espalhamento de interesses relacionados a cada uma dessas variantes. Com isso, há a redução do número de unidades modulares que contribuem para implementação de cada uma dessas variantes. Consequentemente, o número de operações que contribuem para a implementação de cada variante também é reduzido, melhorando os valores de CDO em relação à OO. Para CDLOC, as versões OA dessas variantes foram 100% superiores em relação à OO. Esse resultado é decorrente da remoção do entrelaçamento e do espalhamento de interesses das unidades modulares que implementam cada variante desse padrão.

6.3.1.2 Métricas de Acoplamento, Coesão e Tamanho

Com relação aos resultados das métricas CBC e LCOO, como ilustrado no gráfico da Figura 6.3, o uso da orientação a aspectos na implementação das variantes do padrão Identificar Recurso reduziu em 100% o nível de acoplamento entre suas unidades modulares e aumentou em 100% a coesão das mesmas em relação à OO. Esses resultados se justificam por separar a implementação do padrão Identificar Recurso de unidades que implementam os padrões Quantificar Recurso, os padrões de transação e o padrão Itemizar Transação do Recurso no GRENJ-OO. Como conseqüência, para NOA, DIT, WOC e LOC os resultados também se mostraram favoráveis às versões OA de ambas as variantes desse padrão. As melhorias nos valores dessas métricas se justificam pela remoção de código relacionado ao padrão Identificar Recurso de unidades que foram projetadas para implementar outros padrões. Assim, essas unidades não são mais contabilizadas.

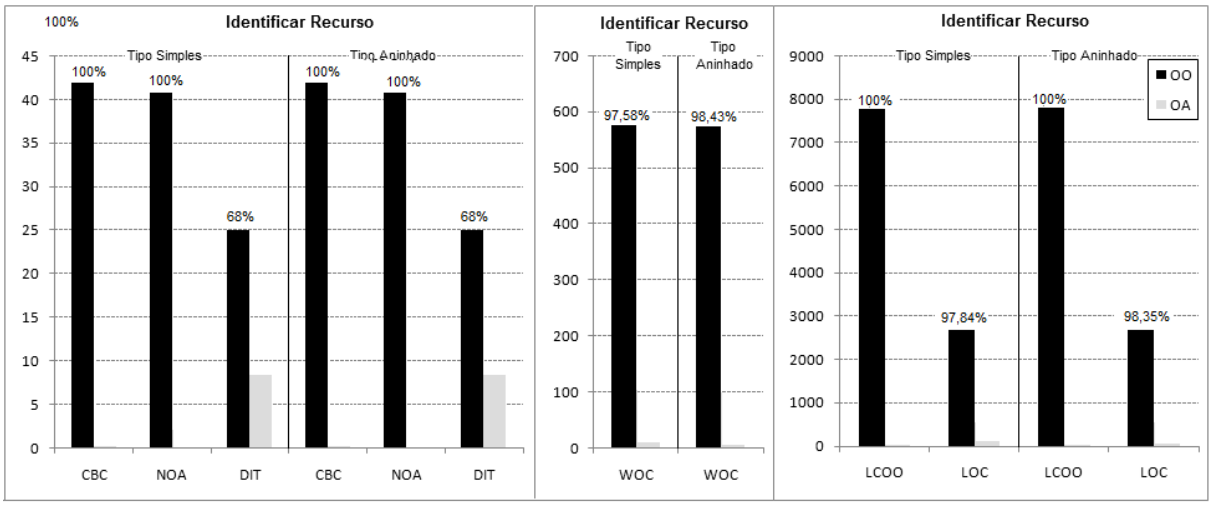

Figura 6.3. Gráfico com os resultados das métricas de acoplamento, coesão e tamanho para as variantes do padrão Identificar Recurso.

Para as variantes do padrão Quantificar Recurso, os resultados da métrica CBC, como ilustrado no gráfico da Figura 6.4, mostraram que suas versões OA foram respectivamente 85,71%, 71,42% e 71,42% superiores à OO. Isso ocorre em virtude da remoção do entrelaçamento e do espalhamento de interesses relacionados a cada variante desse padrão, reduzindo o número de unidades das mesmas. Com isso, a coesão das unidades modulares que implementam cada variante também é melhorada. Com a redução do número de unidades modulares e de operações que contêm código que contribui para a implementação de cada variante na versão OA, os resultados de NOA, DIT e WOC para OA foram superiores em relação às versões OO dessas variantes. Como conseqüência, o número de linhas de código relacionado a cada variante do padrão Quantificar Recurso também foi reduzido, melhorando assim os valores de LOC para cada uma dessas variantes. O código relacionado aos aspectos também foi considerado na medição.

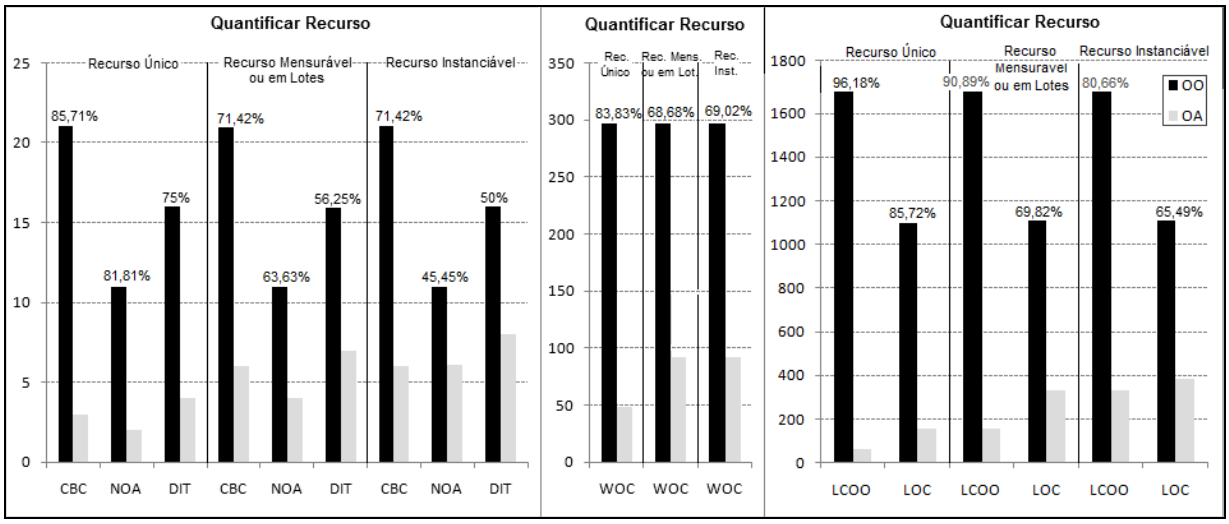

Figura 6.4. Gráfico com os resultados das métricas de acoplamento, coesão e tamanho para as variantes do padrão Quantificar Recurso.

Esses resultados foram obtidos, pois o entrelaçamento e o espalhamento de interesses relacionados a essas variantes foram removidos. Na coleta dos valores das métricas somente as unidades relacionadas a cada variante são contadas e não mais todas as unidades que contêm indícios de interesses relacionados ao padrão Quantificar Recurso, como ocorre no GRENJ-OO.

Os resultados obtidos mostram a efetividade da POA em separar as *features* alternativas do padrão Quantificar Recurso. Dessa forma, somente é carregado o código correspondente à variante necessária para utilizar as funcionalidades fornecidas por esse padrão, facilitando a composição desse padrão com os demais padrões da GRN implementados no *framework*.

6.3.2 Grupo 2: Transações de Negócio

Os resultados das métricas de separação de interesses, acoplamento, coesão e tamanho são apresentados considerando uma variante de cada padrão, exceto para os padrões Manter Recurso e Cotar a Manutenção, em que suas implementações "puras", ou seja, sem variantes, foram consideradas. Por exemplo, na avaliação do padrão Locar Recurso é considerada a variante Locação do Recurso com Multa. Nessa avaliação as implementações OO e OA das variantes desses padrões são comparadas. A apresentação dos resultados está separada para as métricas de separação de interesses e para as de acoplamento, coesão e tamanho.

6.3.2.1 Métricas de Separação de Interesses

A apresentação dos resultados das métricas de separação de interesses está dividida em três partes. Na primeira, são apresentados os resultados dessas métricas para as variantes Locação Instanciável com Multa, do padrão Locar Recurso, Reserva de Recurso Único, do padrão Reservar Recurso e Venda de Recurso Único associada com Locação, do padrão Comercializar Recurso. Na segunda parte são mostrados os resultados para as variantes Cotação do Recurso sem `SourceParty`, do padrão Cotar Recurso e Conferir a Entrega da Venda de Recurso Mensurável, do padrão Conferir a Entrega do Recurso. Na terceira parte, são apresentados os resultados para os padrões Manter Recurso e Cotar a Manutenção sem variantes.

O resultado de CDC para a variante Locação Instanciável com Multa do padrão Locar Recurso mostrou superioridade de 7,69% na versão OA em relação à OO, como mostrado no gráfico da Figura 6.5. Para CDO não houve diferença em relação às versões OO e OA dessa variante. Esse resultado é decorrente de que os atributos e operações relacionados ao padrão Locar Recurso, que estavam espalhados por unidades que implementam outros padrões foram encapsulados em aspectos, ou seja, a remoção de operações relacionadas ao padrão de outras classes do *framework* reduz os valores de CDC e de CDO, mas a adição de aspectos e de operações eleva novamente os valores dessas métricas. Os valores obtidos nesses resultados foram inferiores aos obtidos na análise das variantes dos padrões do grupo 1 em virtude de que o padrão Locar Recurso possui um número menor de variabilidades e um menor nível de entrelaçamento e espalhamento de interesses. Com relação à CDLOC, o uso da orientação a aspectos reduziu em 100% o valor dessa métrica. Isso ocorre pela remoção do código relacionado aos padrões Reservar Recurso e Comercializar Recurso das unidades que implementam o Locar Recurso.

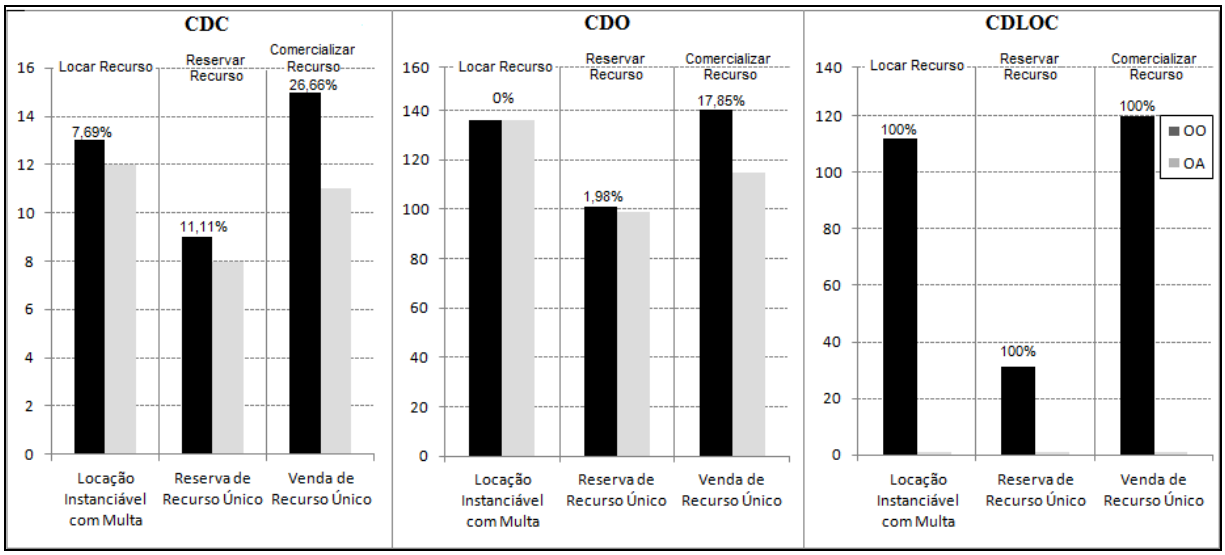

Figura 6.5. Gráfico com os resultados das métricas de separação de interesses para as variantes dos padrões Locar Recurso, Reservar Recurso e Comercializar Recurso.

Para a variante Reserva de Recurso Único do padrão Reservar Recurso, os resultados para as métricas de separação de interesses foram similares aos da variante do padrão Locar Recurso, como ilustrado no gráfico da Figura 6.5. A remoção do espalhamento de interesses na versão OA reduziu de nove para oito o número de unidades que implementam o padrão Reservar Recurso, em que o código relacionado a esse padrão foi removido de classes que implementam o padrão Locar Recurso. Também houve uma redução de três operações para implementar essa variante na versão OA, razão da pequena melhoria no valor de CDO. Operações como `hasReservation()`, que estão relacionadas ao padrão Reservar Recurso, foram removidas de unidades que implementam o padrão Locar Recurso e não foram incorporadas ao código OA. Com relação à CDLOC, a versão OA da variante desse padrão foi 100% superior em relação à solução OO. Isso ocorre em virtude da remoção de código relacionado a outros padrões pelas unidades que implementam o padrão Reservar Recurso.

O uso de OA na implementação da variante Venda de Recurso Único associada com Locação, do padrão Comercializar Recurso, reduziu em 26,66% o número de unidades que contribuem para a sua implementação em relação à OO, como mostrado no gráfico da Figura 6.5. Isso ocorre pela remoção de código relacionado ao padrão Comercializar Recurso de unidades do GRENJ-OO que implementam os padrões Quantificar Recurso e Locar Recurso. Para CDO, a versão OA dessa variante foi 17,85% superior à OO. Isso se justifica por remover operações relacionadas aos padrões Locar Recurso, Reservar Recurso e Conferir a Entrega do Recurso das unidades que implementam as Transações de Negócio e o padrão Comercializar Recurso. Com isso, os valores de CDLOC para a implementação OA dessa

variante foi 100% superior em relação à OO, por remover todos os pontos de transição de interesses das unidades que a implementam.

Para a variante Cotação do Recurso sem `SourceParty` do padrão Cotar Recurso o uso de OA proporcionou uma redução de 20% no número de unidades (CDC) que contribuem para a sua implementação, como ilustrado no gráfico da Figura 6.6. Também reduziu para 7,89% a quantidade de operações (CDO) que contribuem para a sua implementação. Isso ocorre por separar a implementação dessa variante, que é uma *feature* opcional, das unidades que representam o núcleo do padrão Cotar Recurso. Assim, houve o desacoplamento do relacionamento de associação entre o objeto *SourceParty*, que é um elemento opcional desse padrão, com a classe que representa a cotação do recurso. Análogo aos resultados dos demais padrões de transação da GRN, com relação à CDLOC, a versão OA dessa variante do padrão Cotar Recurso também foi 100% superior em relação à sua versão OO. A principal razão para isso é a remoção de código relacionado a esse padrão das unidades que foram projetadas para implementar o padrão Comercializar Recurso.

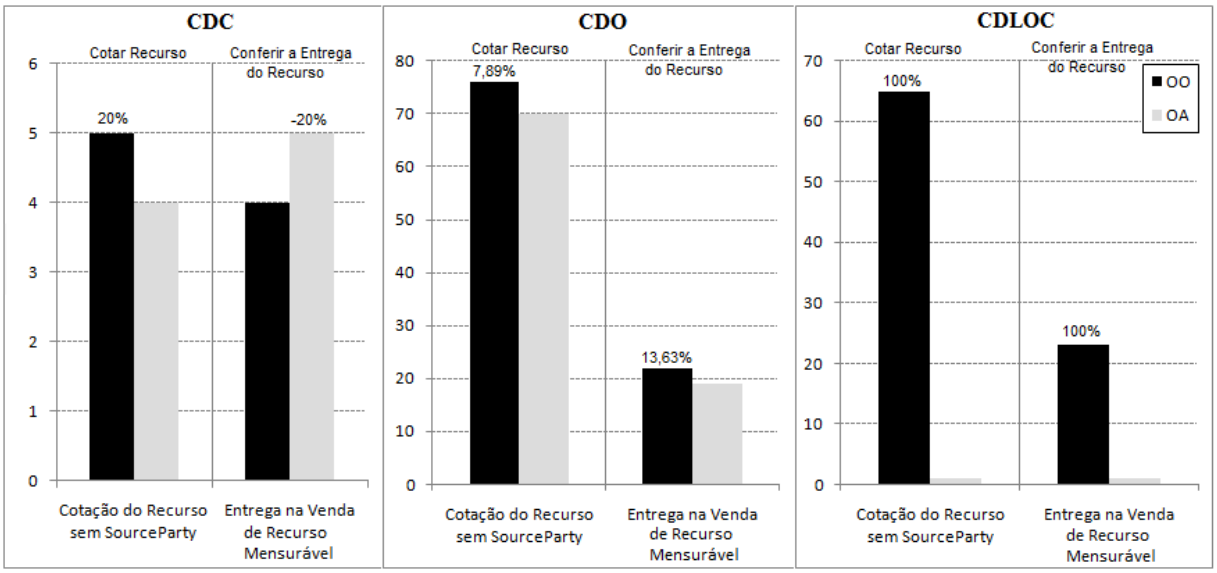

Figura 6.6. Gráfico com os resultados das métricas de separação de interesses para as variantes dos padrões Cotar Recurso e Conferir a Entrega do Recurso.

Diferente dos demais padrões de transação, para a variante Entrega na Venda de Recurso Mensurável, do padrão Conferir a Entrega do Recurso, o resultado da utilização de OA para modularizá-la aumentou o número de unidades que contribuem para implementá-la, sendo o seu CDC 20% inferior em relação à sua implementação OO, como ilustrado no gráfico da Figura 6.6. Isso ocorre em virtude da adição de aspectos para modularizar a implementação dessa variante, aumentando assim o valor de CDC. Ao contrário, para CDO, houve uma redução de 13,63% na quantidade de operações que contribuem para a

implementação da versão OA dessa variante em relação à OO. Isso se deve por encapsular em aspectos as *features* opcionais e alternativas do padrão Conferir a Entrega do Recurso. Na versão OO desse padrão, para utilizar qualquer uma de suas variantes é necessário carregar o código relacionado a todas as outras variantes. Com relação à CDLOC, a versão OA da variante Entrega na Venda de Recurso Mensurável foi 100% superior em relação à sua versão OO. Isso ocorre em virtude da remoção de código relacionado ao padrão Conferir a Entrega do Recurso de unidades que implementam o padrão Comercializar Recurso.

Para o padrão Manter Recurso "puro", ou seja, sem variantes, os resultados de CDC e CDO não apresentaram diferenças entre suas versões OO e OA, como mostrado no gráfico da Figura 6.7. Isso ocorre em virtude da remoção de código relacionado ao padrão Manter Recurso de unidades que implementam os padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção e da criação de aspectos para separar o código relacionado a Manter Recurso dessas unidades. Com relação à CDLOC, a versão OA foi 100% superior em relação à OO, por remover o código relacionado aos padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção (entrelaçamento) das unidades que implementam o padrão Manter Recurso e por remover o código relacionado a esse padrão de unidades que implementam os padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção (espalhamento). Com isso, os pontos de transição de interesse foram removidos das unidades que implementam o Manter Recurso.

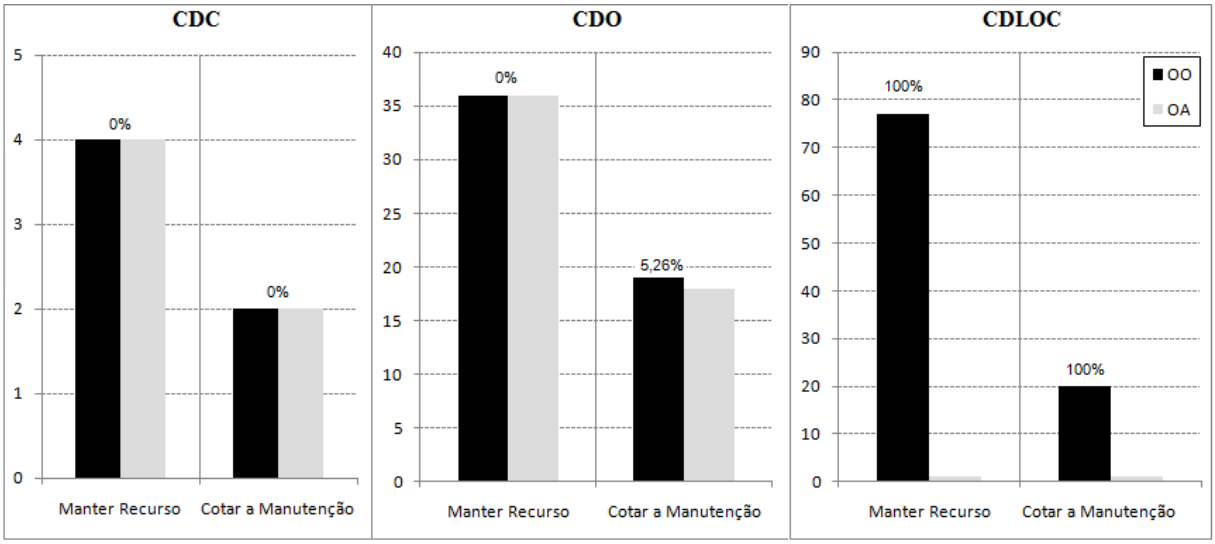

Figura 6.7. Gráfico com os resultados das métricas de separação de interesses para os padrões Manter Recurso e Cotar a Manutenção.

Considerando o padrão Cotar a Manutenção sem variantes, os resultados de CDC também foram os mesmos em ambas as versões, como ilustrado na Figura 6.7. Como esse padrão é opcional, em sua versão OA todo o código relacionado a este foi removido de

unidades do GRENJ-OO que implementam o padrão Manter Recurso e encapsulado em um aspecto, razão pela qual o valor de CDC permanecer o mesmo na versão OA do padrão Cotar a Manutenção. Para CDO, sua versão OA reduziu em 5,26% o número de operações que contribuem para a implementação do padrão. Isso ocorre em virtude da remoção do método `hasQuotation()`, que está relacionado ao padrão Cotar a Manutenção, na versão OA. Em relação à CDLOC, a versão OA desse padrão foi 100% superior à OO. Isso se justifica por remover o código relacionado a Cotar a Manutenção das unidades que implementam o padrão Manter Recurso.

6.3.2.2 Métricas de Acoplamento, Coesão e Tamanho

A apresentação dos resultados das métricas de acoplamento, coesão e tamanho está dividida em três partes. Na primeira, são apresentados os resultados dessas métricas para as variantes Locação Instanciável com Multa, do padrão Locar Recurso, Reserva de Recurso Único, do padrão Reservar Recurso e Venda do Recurso Único associada com Locação, do padrão Comercializar Recurso. Na segunda parte são mostrados os resultados dessas métricas para as variantes Cotação do Recurso sem `SourceParty`, do padrão Cotar Recurso e Conferir a Entrega da Venda de Recurso Mensurável, do padrão Conferir a Entrega do Recurso. Na terceira parte, são apresentados os resultados para os padrões Manter Recurso e Cotar a Manutenção "puros" sem variantes.

Para a variante Locação Instanciável com Multa do padrão Locar Recurso, os resultados das métricas CBC e LCOO, como ilustrado no gráfico da Figura 6.8, mostraram que o uso de OA na implementação dessa variante reduziu em 45% o nível de acoplamento (CBC) entre as unidades modulares que a implementam e aumentou a coesão dessas unidades, sendo 75,1% superior em relação à OO. Esse resultado se justifica pela remoção de código relacionado aos padrões Reservar Recurso e Comercializar Recurso das unidades que implementam o Locar Recurso e por encapsular os elementos opcionais desse padrão em aspectos. Dessa forma, evita-se a presença de código relacionado a essa variante em aplicações que utilizem o padrão Locar Recurso e que não necessitam dessa funcionalidade. Consequentemente, os resultados de NOA, DIT, WOC e LOC também se mostraram favoráveis à versão OA dessa variante, sendo respectivamente 53,12%, 35,48%, 59,06% e 58,17% superior em relação à sua versão no GRENJ-OO, como mostrado no gráfico da Figura 6.8. Isso se deve à redução do número de unidades modulares (classes, interfaces e aspectos) e da quantidade de operações que contribuem para a implementação do padrão

Locar Recurso e à remoção dos relacionamentos de dependência entre essas unidades. O código dos aspectos foi considerado na coleta dos resultados da medição.

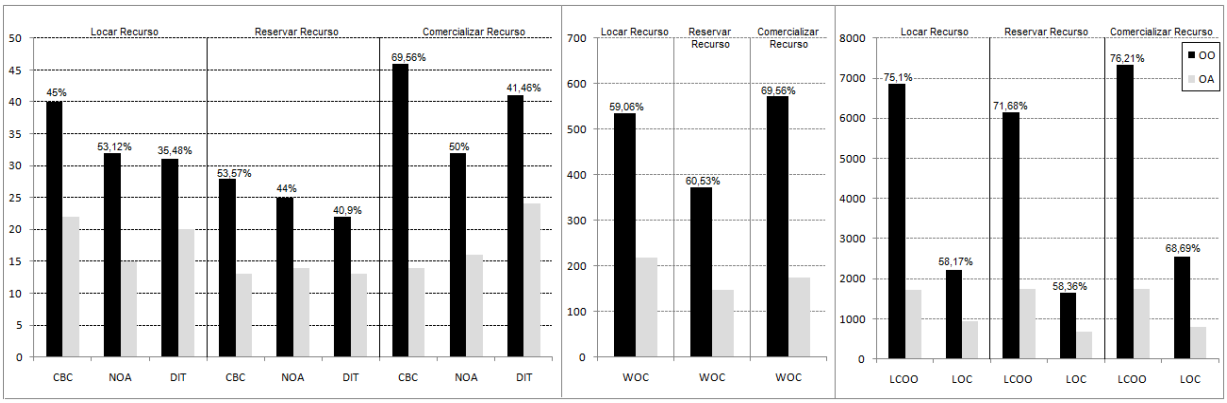

Figura 6.8. Gráfico com os resultados das métricas de acoplamento, coesão e tamanho para as variantes dos padrões Locar Recurso, Reservar Recurso e Comercializar Recurso.

Com relação à variante Reserva de Recurso Único do padrão Reservar Recurso, os resultados de CBC e LCOO mostraram que a versão OA dessa variante foi respectivamente 53,57% e 71,68% superior à sua versão OO, como ilustrado no gráfico da Figura 6.8. Isso ocorre em virtude dos mesmos motivos da versão OA da variante do padrão Locar Recurso. Com a redução do acoplamento e o aumento da coesão das unidades que implementam essa variante, os resultados de NOA, DIT, WOC e LOC respectivamente mostraram uma superioridade de 44%, 40,9%, 60,53% e 58,36% em relação à solução OO.

Considerando a variante Venda de Recurso Único associada com Locação, do padrão Comercializar Recurso, os resultados para as métricas CBC e LCOO foram similares aos obtidos pelas variantes dos padrões Locar Recurso e Reservar Recurso, em que a versão OA dessa variante foi respectivamente 69,56% e 76,21% superior em relação à solução OO, como mostrado no gráfico da Figura 6.8. O mesmo ocorreu com os padrões Locar Recurso e Reservar Recurso em decorrência da remoção do entrelaçamento de interesses das unidades que implementam o padrão Comercializar Recurso. Também por separar os elementos opcionais e alternativos desse padrão, como as variantes Compra e Venda, das unidades que compõem o núcleo do padrão. Como reflexo dessa melhoria, os resultados das métricas de tamanho mostraram que a versão OA da variante do padrão Comercializar Recurso reduziu em 50% os valores de NOA, 41,46% de DIT, 69,56% de WOC e 68,69% os valores de LOC. Esses resultados se justificam pela redução do número de unidades e de operações que contribuem para a implementação do padrão Comercializar Recurso.

Para a variante Cotação do Recurso sem `SourceParty`, do padrão Cotar Recurso, os resultados das métricas CBC e LCOO, como ilustrado no gráfico da Figura 6.9, mostraram

que o uso da orientação a aspectos na implementação dessa variante reduziu em 41,17% o nível de acoplamento entre as unidades modulares que a implementam e aumentou a coesão dessas unidades, sendo 52,23% superior em relação à versão OO. Isso se justifica por separar as variabilidades do padrão Cotar Recurso, em que o código relacionado à Cotação com `SourceParty` foi separado das demais variantes e o código relacionado ao padrão Cotar Recurso foi separado de unidades que implementam o padrão Comercializar Recurso. Dessa forma evita-se a presença de código relacionado a essa variante em aplicações que utilizem o padrão Cotar Recurso e que não necessitam dessa funcionalidade. Como conseqüência, os resultados de NOA, DIT, WOC e LOC também se mostraram favoráveis à versão OA dessa variante, obtendo uma superioridade respectivamente de 20%, 40%, 24,37% e 38,37% em relação à OO, como mostrado no gráfico da Figura 6.9. Isso se deve à redução da quantidade de operações que contribuem para a implementação dessa variante e a remoção dos relacionamentos de dependência entre as unidades que a implementam.

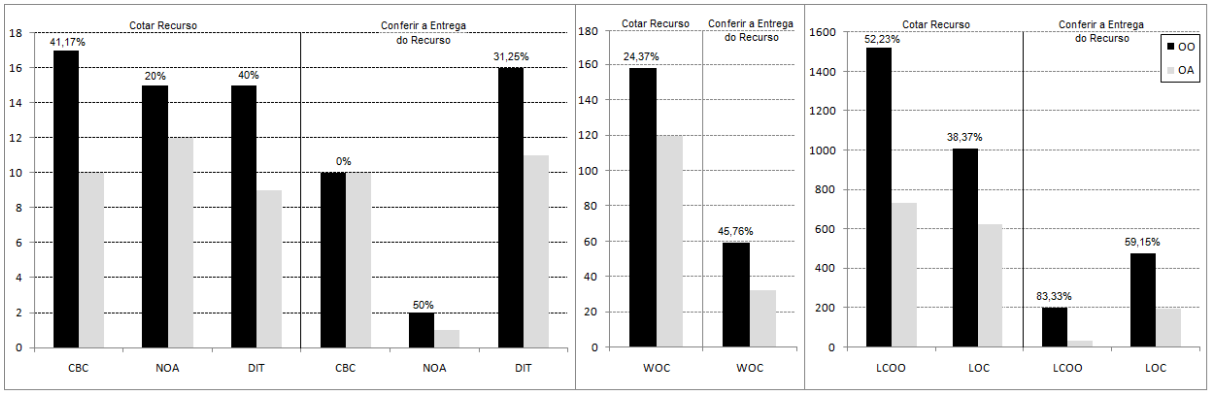

Figura 6.9. Gráfico com os resultados das métricas de acoplamento, coesão e tamanho para as variantes dos padrões Cotar Recurso e Conferir a Entrega do Recurso.

Os resultados de CBC para a variante Entrega da Venda de Recurso Mensurável, do padrão Conferir a Entrega do Recurso, como ilustrado no gráfico da Figura 6.9, mostraram que o uso da orientação a aspectos não influenciou o valor dessa métrica para ambas as versões OA e OO. Isso ocorre em virtude da inversão de acoplamento na implementação dessa variante, em que o código relacionado a esta foi removido de classes que implementam os padrões Comercializar Recurso e Conferir a Entrega do Recurso, mas aspectos foram criados para separar sua implementação desses padrões. Com relação à LCOO, a versão OA dessa variante foi 83,33% superior em relação à sua versão OO, por remover o código relacionado ao padrão Conferir a Entrega do Recurso de unidades que implementam o Comercializar Recurso. Com isso, os resultados de NOA, DIT, WOC e LOC também foram

favoráveis à OA, sendo respectivamente 50%, 31,25%, 45,76% e 59,15% superior em relação à OO.

Para o padrão Manter Recurso sem variantes, ao contrário dos resultados das métricas de separação de interesses, os resultados de CBC e LCOO mostraram que sua versão OA reduziu em 33,33% o nível de acoplamento entre as unidades que contribuem para implementá-lo e aumentou a coesão dessas unidades, sendo 96,13% superior à OO, como mostrado no gráfico da Figura 6.10. Isso se justifica pela remoção de código relacionado ao padrão Manter Recurso de unidades que implementam os padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção. Outro fator que contribuiu para esses resultados foi a remoção de código relacionado ao padrão Cotar a Manutenção de unidades que implementam o Manter Recurso. Essas melhorias se propagaram para as métricas NOA, DIT, WOC e LOC, reduzindo em 81,81% o número de atributos das unidades que implementam o Manter Recurso; em 18,18% os valores de DIT, em 65,05% os valores de WOC, e consequentemente, alcançando uma redução de 73,56% no número de linhas de código do padrão. Esses resultados se justificam pela remoção de código relacionado ao padrão Manter Recurso de unidades que implementam os padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção. Assim, essas unidades não foram contabilizadas na medição da implementação OA do padrão Manter Recurso.

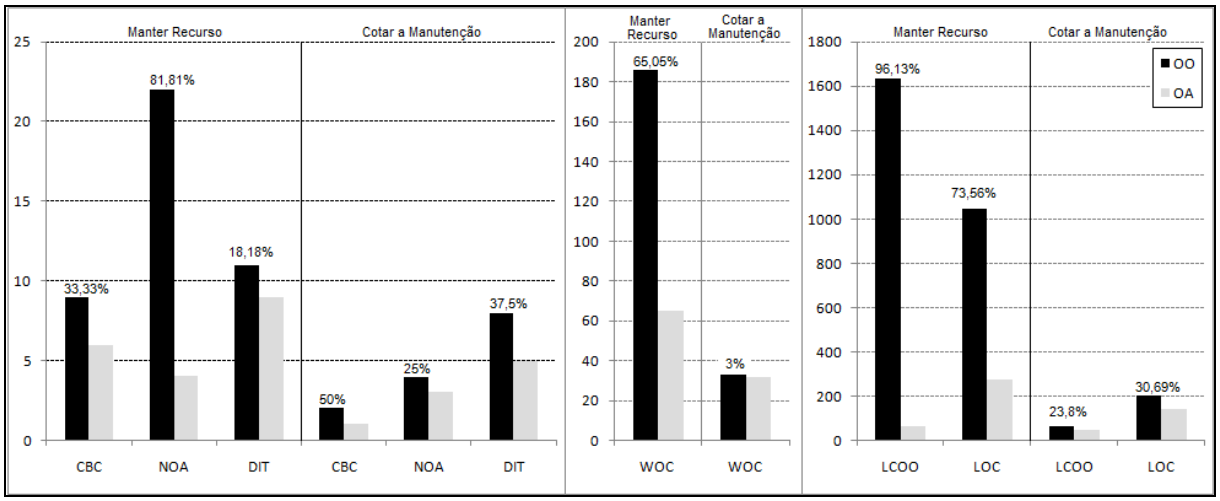

Figura 6.10. Gráfico com os resultados das métricas de acoplamento, coesão e tamanho para os padrões Manter Recurso e Cotar a Manutenção.

Os resultados das medidas de acoplamento, coesão e tamanho considerando o padrão Cotar a Manutenção sem variantes, assim como os do Manter Recurso, também são contrários aos valores das métricas de separação de interesses. Isso ocorre em ambos os padrões pela mesma razão, que consiste no fato de que apesar da remoção do entrelaçamento e do espalhamento de interesses relacionados a esses padrões, aspectos foram criados para

modularizá-los. Para CBC e LCOO, o uso da OA na implementação do padrão Cotar a Manutenção reduziu em 50% o nível de acoplamento entre as unidades que o implementam e elevou para 23,8% a coesão das mesmas, como ilustrado no gráfico da Figura 6.10. Isso ocorre em virtude da remoção de código relacionado ao Cotar a Manutenção de unidades que implementam o Manter Recurso. Como conseqüência, os valores das métricas NOA, DIT, WOC e LOC também foram melhores para OA, uma vez que como não há mais código relacionado ao padrão Cotar a Manutenção nas unidades que implementam o Manter Recurso, essas unidades não são contadas.

6.3.3 Grupo 3: Detalhes das Transações de Negócio

Os resultados das métricas de separação de interesses, acoplamento, coesão e tamanho são apresentados considerando uma variante de cada padrão. Nessa avaliação, as implementações OO e OA das variantes desses padrões são comparadas. A apresentação dos resultados está separada para as métricas de separação de interesses e para as de acoplamento, coesão e tamanho.

6.3.3.1 Métricas de Separação de Interesses

A apresentação dos resultados das métricas de separação de interesses está dividida em três partes. Na primeira, são apresentados os resultados para as variantes Itemizar Locação de Recurso Instanciável, do padrão Itemizar Transação do Recurso, e Pagamento em Dinheiro, do padrão Pagar pela Transação do Recurso. Na segunda são mostrados os resultados para a variante Transação com Executor, do padrão Identificar o Executor da Transação. Na terceira são apresentados os resultados para os padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção sem variantes.

O padrão Itemizar Transação do Recurso é um dos padrões da GRN que possui o maior número de variabilidades, em virtude de que em sua implementação no *framework* GRENJ-OO há código relacionado a ele por unidades que implementam os padrões Identificar Recurso, Locar Recurso, Reservar Recurso, Comercializar Recurso, Cotar Recurso, Conferir a Entrega do Recurso, Manter Recurso, Cotar a Manutenção, Identificar Tarefas de Manutenção e Identificar Peças de Manutenção. A orientação a aspectos foi utilizada para modularizar cada uma dessas variantes, no qual a variante Itemizar Locação de Recurso Instanciável foi escolhida para comparação.

Considerando apenas a variante Itemizar Locação de Recurso Instanciável do padrão Itemizar Transação do Recurso, os resultados de CDC mostraram que sua versão OA reduziu em 60%, o número de unidades que contribuem para a sua implementação, como ilustrado no gráfico da Figura 6.11. Isso ocorre por desacoplar a implementação do padrão Itemizar Transação do Recurso de unidades que foram projetadas para implementar outros padrões, ou seja, o código relacionado ao padrão foi removido da implementação dos demais padrões da GRN no GRENJ-OO. Para CDO, houve uma redução de 40,86% na quantidade de operações que contribuem para a implementação dessa variante, decorrente da melhoria no valor de CDC para essa variante. Por encapsular em aspectos o código relacionado ao padrão Itemizar Transação do Recurso, que estava localizado em unidades que implementam outros padrões, o valor de CDLOC para a variante Itemizar Locação de Recurso Instanciável mostrou que sua versão OA foi 100% superior à OO.

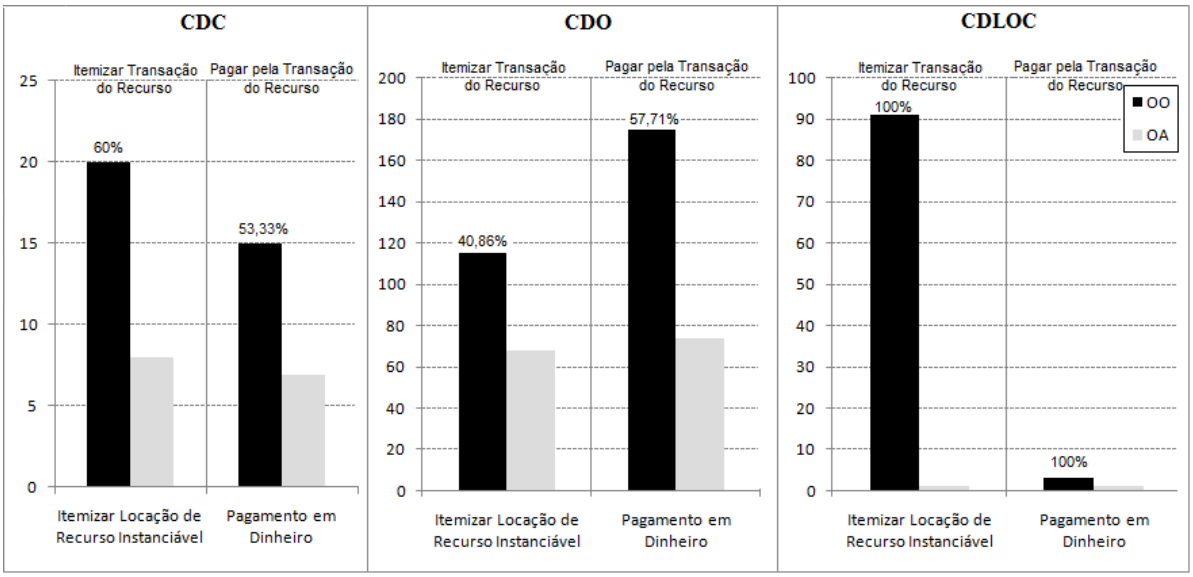

Figura 6.11. Gráfico com os resultados das métricas de separação de interesses para as variantes dos padrões Itemizar Transação do Recurso e Pagar pela Transação do Recurso.

Para a variante Pagamento em Dinheiro do padrão Pagar pela Transação do Recurso, sua versão OA reduziu em 53,33% o número de unidades que contribuem para implementá-la em relação à OO, como ilustrado no gráfico da Figura 6.11. Isso ocorre em virtude da remoção de código relacionado a essa variante das unidades que implementam outras variantes do padrão Pagar pela Transação do Recurso. Com isso, também houve uma redução de 57,71% no número de operações que contribuem para a implementação dessa variante em sua implementação OA comparado com OO. Com relação à CDLOC, a versão OA dessa variante foi 100% superior em relação à OO. Esse resultado se justifica pela remoção do código relacionado a essa variante de unidades que implementam as demais variantes do

padrão, reduzindo os pontos de transição de interesse relacionados a essa variante pelo código do padrão.

A implementação OA da variante Transação com Executor do padrão Identificar o Executor da Transação reduziu em 50% o número de unidades que contribuem para a sua implementação (CDC), como mostrado no gráfico da Figura 6.12. Isso ocorre pela remoção de código relacionado ao padrão Identificar o Executor da Transação de unidades que implementam os padrões de transação e o padrão Identificar Tarefas de Manutenção. Com isso, a quantidade de operações que contribuem para sua implementação também foi reduzida, melhorando em 43,18% o valor de CDO em relação à solução OO. Em virtude da remoção de código relacionado ao padrão Identificar o Executor da Transação de unidades que implementam outros padrões, para CDLOC, a versão OA da variante desse padrão foi 100% superior em relação à OO.

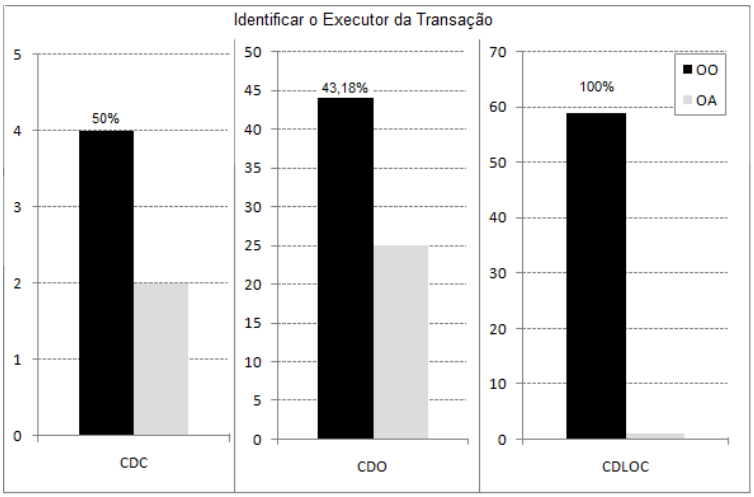

Figura 6.12. Gráfico com os resultados das métricas de separação de interesses para a variante "Transação com Executor" do padrão Identificar o Executor da Transação.

Com relação ao padrão Identificar Tarefas de Manutenção sem variantes, os resultados para as métricas CDC e CDO foram os mesmos para ambas as suas versões OO e OA, como mostrado no gráfico da Figura 6.13. Isso ocorre mesmo com a remoção de código relacionado a esse padrão de unidades que implementam o padrão Manter Recurso, pois um aspecto foi criado para modularizar esse código, não alterando os valores de CDC e de CDO. Esses resultados se justificam pela ocorrência de inversão de acoplamento, em que cada atributo e operação relacionado ao padrão Identificar Tarefas de Manutenção que estava presente em unidades que implementam o Manter Recurso foi removido e encapsulado em um aspecto. Com relação à CDLOC, a versão OA dessa variante foi 100% superior à solução OO. Isso ocorre em virtude da remoção do entrelaçamento e do espalhamento de interesses

relacionados ao padrão Identificar Tarefas de Manutenção, removendo os pontos de transição de interesse das unidades que o implementam.

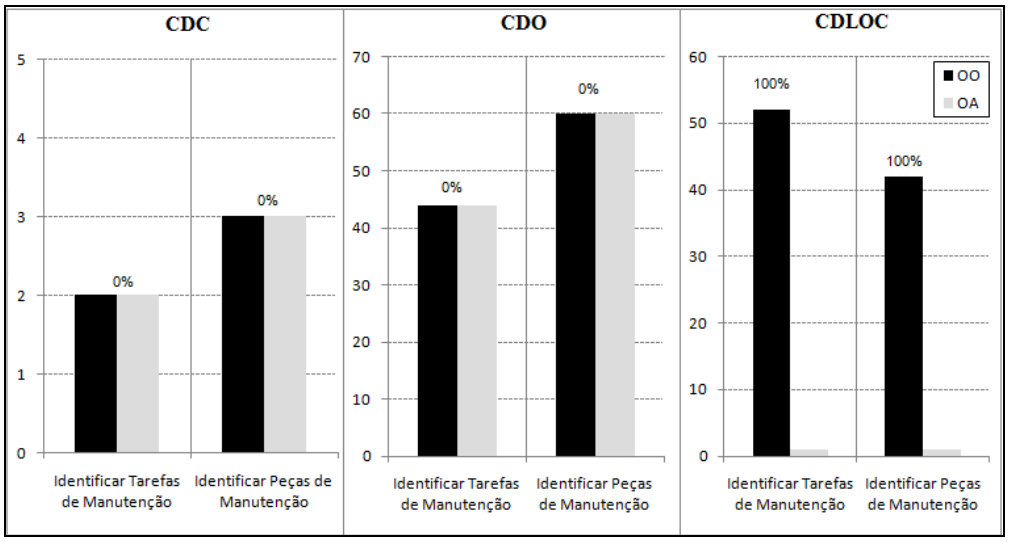

Figura 6.13. Resultados das métricas de separação de interesses para os padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção sem variantes.

Para o padrão Identificar Peças de Manutenção sem variantes, os resultados das métricas de separação de interesses foram os mesmos que os do padrão Identificar Tarefas de Manutenção, como ilustrado no gráfico da Figura 6.13. Assim como no padrão Identificar Tarefas de Manutenção, isso ocorre em virtude da remoção de código relacionado ao padrão Identificar Peças de Manutenção das unidades que implementam o padrão Manter Recurso e da criação de um aspecto para modularizar esse código. Esse aspecto foi criado para separar a implementação do padrão Identificar Peças de Manutenção do padrão Manter Recurso.

6.3.3.2 Métricas de Acoplamento, Coesão e Tamanho

A apresentação dos resultados das métricas de acoplamento, coesão e tamanho está dividida em três partes. Na primeira são apresentados os resultados para as variantes Itemizar Locação de Recurso Instanciável, do padrão Itemizar Transação do Recurso, e Pagamento em Dinheiro, do padrão Pagar pela Transação do Recurso. Na segunda parte são mostrados os resultados para a variante Transação com Executor, do padrão Identificar o Executor da Transação. Na terceira parte, são apresentados os resultados para os padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção sem variantes.

Para a variante Itemizar Locação de Recurso Instanciável do padrão Itemizar Transação do Recurso, os resultados de CBC e LCOO mostraram que a versão OA dessa variante reduziu em 60,86% o nível de acoplamento entre as unidades que a implementam e aumentou em 97,07% a coesão das mesmas, como mostrado no gráfico da Figura 6.14. Isso

ocorre em virtude da remoção do código relacionado ao padrão Itemizar Transação do Recurso de unidades que implementam outros padrões da GRN. Como conseqüência, os resultados das métricas de tamanho NOA, DIT, WOC e LOC também foram melhorados, em que a versão OA dessa variante foi respectivamente 80,76%, 81,81%, 80,97% e 79,95% superior em relação à OO. Esses resultados se justificam uma vez que houve a separação da implementação do padrão Itemizar Transação do Recurso de unidades que implementam padrões como Locar Recurso, Reservar Recurso, Comercializar Recurso, Cotar Recurso e Manter Recurso. Dessa forma, unidades relacionadas a esses padrões não são mais contadas na avaliação da implementação OA da variante Itemizar Locação de Recurso Instanciável.

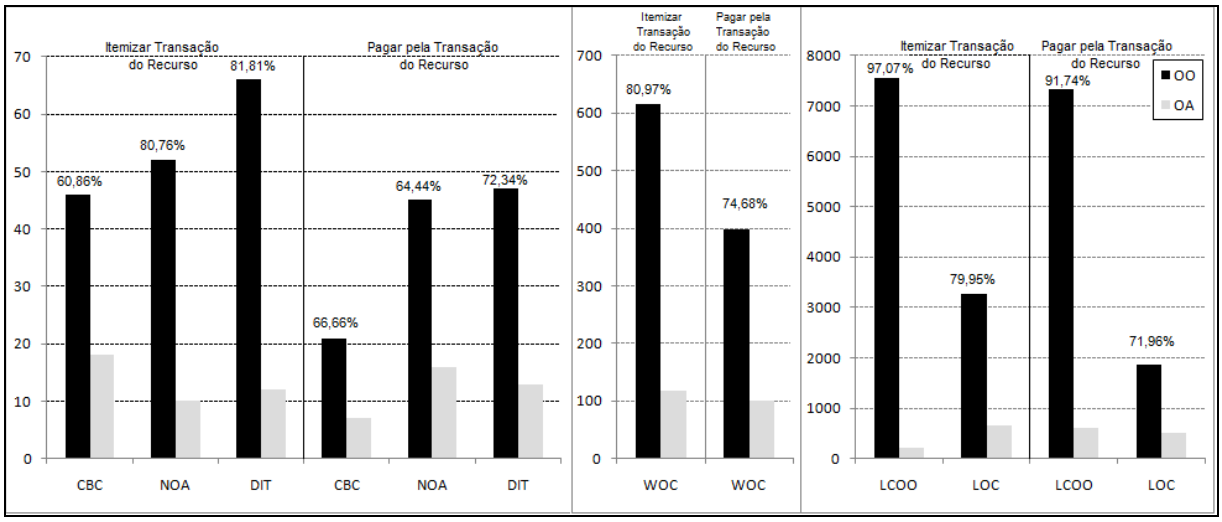

Figura 6.14. Gráfico com os resultados das métricas de acoplamento, coesão e tamanho para as variantes dos padrões Itemizar Transação do Recurso e Pagar pela Transação do Recurso.

O uso de OA na implementação da variante Pagamento em Dinheiro do padrão Pagar pela Transação do Recurso reduziu em 66,66% o acoplamento entre as unidades que a implementam e aumentou em 91,74% a coesão dessas unidades, como ilustrado no gráfico da Figura 6.14. Esses resultados se justificam por isolar a implementação dessa variante das demais variantes do padrão Pagar pela Transação do Recurso. Como conseqüência, os resultados de NOA, DIT, WOC e LOC para a sua versão OA também foram superiores em relação à OO.

Com relação à variante Transação com Executor do padrão Identificar o Executor da Transação, os resultados para as métricas CBC e LCOO, como mostrado no gráfico da Figura 6.15, apontaram uma redução de 87,5% no nível de acoplamento entre as unidades que implementam essa variante em sua versão OA e aumentou em 98,2% a coesão das mesmas. Isso se justifica pela remoção dos relacionamentos de dependência entre as unidades que implementam o padrão Identificar o Executor da Transação e unidades que implementam os

padrões de transação e Identificar Tarefas de Manutenção no GRENJ-OO. Com isso, na versão OA da variante Transação com Executor, houve uma redução de 84,84% no número de atributos (NOA), uma vez que as unidades relacionadas a outros padrões, que tinham código relacionado ao padrão Identificar o Executor da Transação na versão OO, não são mais contabilizadas. O mesmo ocorre para as medidas DIT, WOC e LOC.

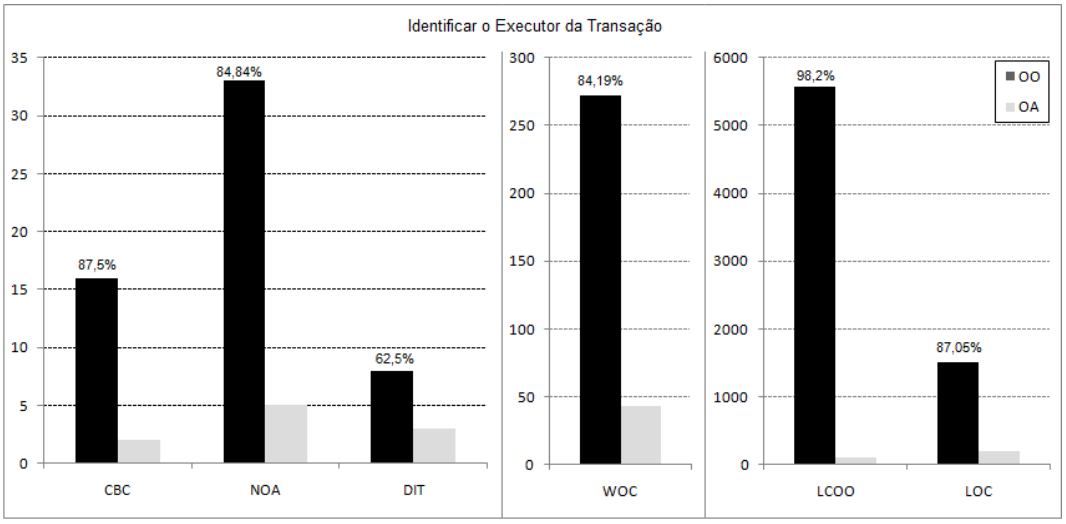

Figura 6.15. Gráfico com os resultados das métricas de acoplamento, coesão e tamanho para a variante "Transação com Executor" do padrão Identificar o Executor da Transação.

Para o padrão Identificar Tarefas de Manutenção sem variantes, os resultados das métricas CBC e LCOO mostraram que sua versão OA reduziu em 50% o nível de acoplamento entre as unidades que o implementam e aumentou em 75,27% a coesão das mesmas, como apresentado no gráfico da Figura 6.16. Essas melhorias são decorrentes da remoção de código relacionado ao padrão de unidades que implementam o padrão Manter Recurso. Com isso, os valores de NOA, DIT, WOC e LOC também foram melhorados, em que a versão OA do padrão Identificar Tarefas de Manutenção foi respectivamente 50%, 40%, 44,18% e 50,73% superior à OO. O valor de WOC representa o grau de complexidade das operações das unidades que contribuem para a implementação do padrão Identificar Tarefas de Manutenção.

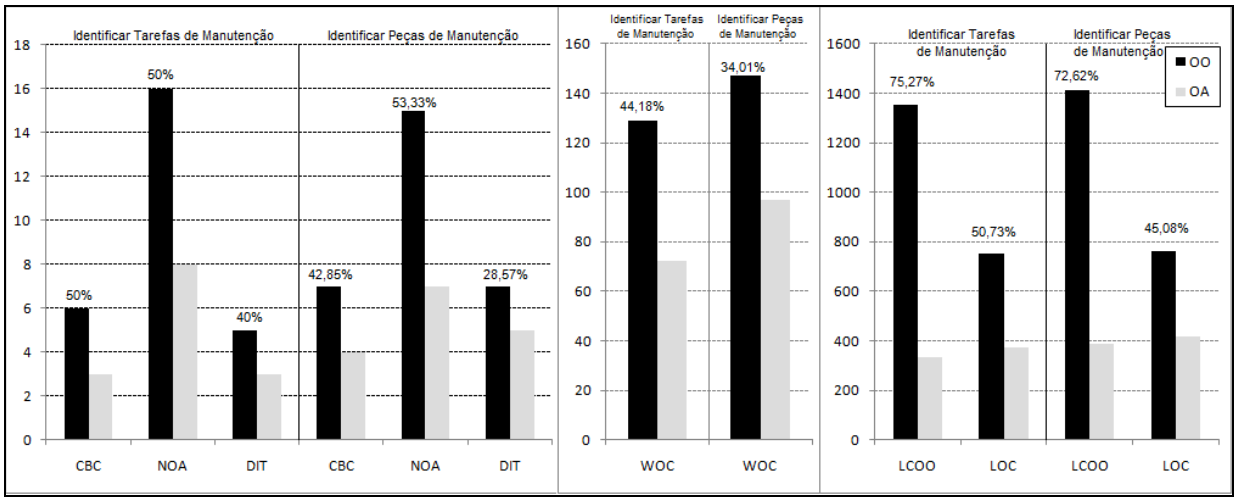

Figura 6.16. Gráfico com os resultados das métricas de acoplamento, coesão e tamanho para os padrões Identificar Tarefas de Manutenção e Identificar Peças de Manutenção sem variantes.

Considerando o padrão Identificar Peças de Manutenção sem variantes, os resultados de CBC e LCOO, mostrados na Figura 6.16, foram similares aos do padrão Identificar Tarefas de Manutenção, em que sua versão OA reduziu em 42,85% o acoplamento entre as unidades que implementam o padrão e aumentou em 72,62% a coesão dessas unidades. Isso ocorre em virtude da remoção de código relacionado ao padrão Identificar Peças de Manutenção de unidades que implementam o padrão Manter Recurso. Essas melhorias também se refletem nos resultados das métricas NOA, DIT, WOC e LOC, em que a versão OA do padrão Identificar Peças de Manutenção foi respectivamente 53,33%, 28,57%, 34,01% e 45,08% superior à sua versão OO.

6.4 Análise dos Resultados

A partir da análise dos valores das métricas de separação de interesses, acoplamento, coesão e tamanho para cada padrão da GRN implementado no GRENJ-OO foi constatado quatro tipos diferentes de resultados: **A)** "Métricas de Separação de Interesses, Acoplamento, Coesão e Métricas de Tamanho favoráveis para OA"; **B)** "Métricas de Separação de Interesses pouco favoráveis para OA e Métricas de Acoplamento, Coesão e Tamanho favoráveis para OA"; **C)** "Métricas de Separação de Interesses favoráveis para OO, Métrica de Acoplamento igual para OO e OA e Métricas de Coesão e Tamanho favoráveis para OA"; e **D)** "Métricas de Separação de Interesses iguais para OO e OA e Métricas de Acoplamento, Coesão e Tamanho favoráveis a OA". Na Tabela 6.1 é apresentado o agrupamento dos padrões da GRN em cada uma dessas categorias de resultados.

Tabela 6.1 – Tipos de resultados e os padrões da GRN.

Tipo de Resultados	Padrões da GRN
A	Identificar Recurso, Quantificar Recurso, Comercializar Recurso, Itemizar Transação do Recurso, Pagar pela Transação do Recurso e Identificar o Executor da Transação.
B	Locar Recurso, Reservar Recurso e Cotar Recurso.
C	Conferir a Entrega do Recurso.
D	Manter Recurso, Cotar a Manutenção, Identificar Tarefas de Manutenção e Identificar Peças de Manutenção.

A) Métricas de Separação de Interesses, Acoplamento, Coesão e de Tamanho favoráveis para OA: A justificativa para a ocorrência desses resultados para este grupo de padrões consiste no fato de que suas versões OO estão entrelaçadas e espalhadas por várias unidades do *framework* GRENJ-OO. Outro fator a ser considerado é a existência de um grande número de variabilidades inerentes a cada um desses padrões que estão altamente acopladas nesse *framework*. Dessa forma, o uso de OA na modularização dessas variabilidades favoreceu os resultados das medidas de acoplamento, coesão e tamanho.

B) Métricas de Separação de Interesses pouco favoráveis para OA e métricas de Acoplamento, Coesão e Tamanho favoráveis para OA: Isso ocorre para os padrões desse grupo em virtude de possuírem um número menor de variabilidades quando comparado com os padrões do grupo A). Assim nas versões OA de suas variantes, o espalhamento de interesses relacionados a cada uma delas foi removido das unidades do GRENJ-OO que não foram projetadas para implementá-las, mas aspectos foram criados para modularizá-las, resultando em melhorias discretas para os valores de CDC e CDO. Ao contrário, os resultados das métricas de acoplamento, coesão e tamanho mostraram que suas versões OA foram superiores em relação às OO. Isso se justifica pela efetividade da OA em separar as *features* opcionais e alternativas dos padrões Locar Recurso, Reservar Recurso e Cotar Recurso.

C) Métricas de Separação de Interesses favoráveis para OO, Métrica de Acoplamento igual para OO e OA e Métricas de Coesão e Tamanho favoráveis para OA: esses resultados foram obtidos somente na avaliação da variante Entrega de Venda de Recurso Mensurável, do padrão Conferir a Entrega do Recurso. A razão da versão OA dessa variante ter um valor inferior para CDC em relação à OO é decorrente do nível de granularidade fina que essa variante possui, ou seja, sua implementação afeta o núcleo do padrão (seus elementos base) e outras variantes desse padrão. Com isso, aspectos foram criados para encapsular a implementação dessa variante em cada um desses níveis de

granularidade. Com relação a acoplamento, esses resultados ocorrem porque houve uma inversão de acoplamento na implementação dessa variante, em que o código relacionado a ela foi removido de unidades que implementam outras variantes, mas um aspecto foi criado para modularizar a implementação dessa variante. Para coesão e de tamanho, os resultados foram favoráveis para OA em virtude dos interesses relacionados a essa variante estarem separados dos elementos base do padrão Conferir a Entrega do Recurso e de outras variantes desse padrão.

D) Métricas de Separação de Interesses iguais para OO e OA e Métricas de Acoplamento, Coesão e Tamanho favoráveis a OA: Os resultados das métricas de separação de interesses para esses padrões se justificam pela remoção do entrelaçamento e do espalhamento de interesses relacionados a esses padrões das unidades do *framework* GRENJ-OO que implementam outros padrões. Com isso, um aspecto foi criado para cada unidade desse *framework* que contém código relacionado a esses padrões. Dessa forma, os valores de CDC não se alteraram em ambas as versões OO e OA desses padrões. Ao contrário, os resultados das métricas de acoplamento, coesão e tamanho foram favoráveis para OA. Isso ocorre pela efetividade da OA em separar a implementação dos padrões desse grupo, que são *features* opcionais do *framework* GRENJ-OO, dos demais padrões da GRN implementados nesse *framework*.

6.5 Considerações Finais

Neste capítulo foi apresentada uma avaliação quantitativa para comparar as versões OO e OA dos padrões da linguagem GRN no *framework* GRENJ-OO. A realização dessa avaliação teve o objetivo de verificar a efetividade do uso de abstrações orientadas a aspectos na modularização de *frameworks* desenvolvidos com linguagens de padrões de análise. Notou-se que com o uso de orientação a aspectos houve a redução do acoplamento e aumento da coesão na maioria dos padrões da GRN e de suas variantes implementadas no *framework* GRENJ-OO. Dessa forma, a manutenibilidade, a reusabilidade e o gerenciamento de variabilidades desse *framework* podem ser obtidos mais facilmente.

Com o uso de OA houve aumento do número de linhas de código, número de operações e outras medidas de tamanho relacionadas ao GRENJ-OO. Na Tabela 6.2 estão apresentados os valores encontrados para as versões GRENJ-OO e GRENJ-OA referentes ao

número de unidades, ao número de métodos, ao número de classes abstratas, ao número de métodos abstratos e linhas de código, bem como as diferenças entre esses valores. A porcentagem positiva significa que a versão OA apresentou resultados melhores em relação à OO, o valor zero indica que não houve alteração entre as versões e a negativa que a versão OO apresentou resultados melhores em relação à versão OA.

Tabela 6.2. Medidas de tamanho dos *frameworks* GRENJ-OO e GRENJ-OA.

Frameworks/Medidas de Tamanho	Linhas de Código (LOC)	Número de Unidades	Número de Métodos/Advices	Classes Abstratas	Métodos Abstratos
GRENJ-OO	18424	71	1117	40	83
GRENJ-OA	20297	152	1222	40	71
Diferença OO e OA	-10,16%	-114,08%	-9,4%	0%	+14,45%

Durante a realização da avaliação quantitativa foi verificado que o uso de orientação a aspectos para modularizar variabilidades de *frameworks* construídos com base em linguagens de padrões, aumenta a quantidade de suas unidades em determinadas situações, como ocorreu na modularização do GRENJ-OO. Isso pode conduzir a falsos negativos quando se utiliza métricas para avaliá-los. Por exemplo, ao considerar o padrão como um todo na avaliação, ou seja, o padrão com todas as suas variantes, pode-se concluir que uso de OA prejudica a separação de interesses, aumenta o acoplamento e reduz a coesão da implementação do padrão. Porém, quando se analisa cada variante isoladamente, é possível verificar as melhorias que OA proporciona ao separar a implementação de cada variante do padrão.

Com base nos resultados obtidos neste estudo pôde-se concluir que a orientação a aspectos é uma solução adequada para modularizar variabilidades em *frameworks* desenvolvidos com base em linguagens de padrões análise. A descrição e a condução de um experimento são descritas no próximo capítulo.

Capítulo 7

EXPERIMENTO DE MANUTENÇÃO

7.1 Considerações Iniciais

Experimentos de software são conduzidos com a finalidade de controlar e manipular soluções de projeto, com base no ambiente e em um conjunto de objetivos, para estabelecer evidências empíricas do que funciona e o que não funciona sobre determinadas condições (BASILI *et al.*, 1986).

Neste capítulo é relatado um experimento conduzido com o objetivo de comparar as versões GRENJ-OO e GRENJ-OA quanto à compreensão, à manutenibilidade e à reusabilidade. A motivação para realizar esse experimento consistiu da necessidade de se mensurar os benefícios do uso da orientação a aspectos na modularização do *framework* GRENJ-OO, bem como suas limitações, com a finalidade de verificar o impacto da OA na arquitetura do *framework* e em seu potencial de manutenção e reutilização.

Para a apresentação do experimento realizado o formato de descrição sugerido por Wohlin *et al.* (2000) foi utilizado. O experimento deve ser definido, planejado, executado e dados devem ser coletados e analisados para apresentação dos resultados obtidos. Assim, na Seção 7.2 são descritos a definição e o planejamento do experimento. Na Seção 7.3 são apresentados os dados coletados durante a realização do experimento. Na Seção 7.4 é mostrada a análise dos dados coletados. Na Seção 7.5 os resultados obtidos do experimento são apresentados. Na Seção 7.6 as ameaças a validade do experimento são apresentadas. Na Seção 7.7 são expostas as considerações finais.

7.2 Definição e Planejamento do Experimento

O objetivo do experimento conduzido foi comparar as versões GRENJ-OO e GRENJ-OA quanto à compreensão, à manutenibilidade e à reusabilidade, do ponto de vista de desenvolvedores de software no contexto de alunos de pós-graduação.

O planejamento do experimento constou de a) seu contexto, que corresponde ao ambiente no qual foi aplicado; b) a definição das hipóteses que corresponde a uma previsão dos resultados que se pretende obter; c) seleção de variáveis controladas pelo experimento, ou seja, o que se pretende controlar; d) o critério de seleção dos participantes; e) o projeto experimental, que consiste no planejamento das atividades do experimento; f) tipo de projeto, que se refere à forma de distribuição dos membros para executar essas atividades, e g) instrumentação, relacionada aos artefatos (formulários, diagramas de classes, versões de *frameworks*) que foram utilizados durante a realização do experimento.

a) Seleção do contexto. O estudo foi realizado com estudantes de mestrado em Ciência da Computação da Universidade Federal de São Carlos, que cursaram a disciplina Engenharia de Software, no primeiro semestre de 2010. Foram elaborados questionários relacionados às versões GRENJ-OO e GRENJ-OA, cenários de manutenção e cenários de utilização no domínio desse *framework.*

b) Definição de hipóteses. A hipótese a ser testada é: "a versão orientada a aspectos do *framework* GRENJ-OO facilita a compreensão de suas funcionalidades, sua reutilização, manutenção e evolução em relação à sua versão orientada a objetos". Essa hipótese possui como base o estudo quantitativo baseado em métricas de separação de interesses, acoplamento, coesão e de tamanho apresentado no Capítulo 6. Os resultados obtidos no estudo quantitativo indicaram que o uso de orientação a aspectos melhorou os níveis de separação de interesses, reduziu o acoplamento e aumentou a coesão das unidades que implementam cada padrão e variante da linguagem de padrões GRN no *framework* GRENJ-OO.

c) Seleção de variáveis. As variáveis independentes são todas aquelas que são manipuladas e controladas durante o estudo. Neste estudo, as versões GRENJ-OO e GRENJ-OA são as variáveis independentes. As variáveis dependentes são aquelas que estão sob análise (WOHLIN *et al.*, 2000). Nesse experimento as variáveis dependentes são: as respostas dos questionários relacionados às arquiteturas desses *frameworks*, que indicam o número de acertos de cada participante ao analisar as versões OO e OA desse *framework*, o tempo de

resposta dos questionários, o tempo para realizar manutenções e o tempo para instanciar aplicações utilizando as versões do *framework*. Essas variáveis são influenciadas pelas versões OO e OA do *framework*.

d) Critério de seleção dos participantes. Os participantes do estudo foram selecionados por meio de amostragem não-probabilística por conveniência, ou seja, a probabilidade de todos os elementos da população pertencer à mesma amostra é desconhecida.

e) Projeto experimental. O experimento foi planejado seguindo uma estrutura de blocos e foi balanceado para assegurar que os tratamentos tenham o mesmo número de participantes, para possibilitar a comparação dos efeitos das variáveis independentes. O experimento foi dividido em três etapas: Compreensão, Manutenção e Reutilização. A etapa de "Compreensão" teve o objetivo de verificar a facilidade dos participantes envolvidos em localizar as unidades que implementam as funcionalidades fornecidas pelo *framework*. As etapas de "Manutenção" e "Reutilização" envolveram os mesmos participantes e tiveram como objetivo comparar a manutenibilidade e a reusabilidade das versões GRENJ-OO e GRENJ-OA do *framework*, por meio de dois cenários de manutenção e dois de instanciação. Ambos os cenários de manutenção e de reutilização foram aplicados às versões OO e OA. Cada uma das etapas do experimento é composta por duas fases nas quais os grupos estão organizados como ilustrado na Tabela 7.1. Para evitar que a ordem das variáveis independentes exercesse influência sobre os resultados do experimento, em cada etapa do experimento houve a preocupação em definir de maneira homogênea a ordem de utilização de cada uma dessas variáveis. Assim, na Etapa 1 - Compreensão, Fase 1, o grupo 1 utilizou a versão GRENJ-OO e o grupo 2 a versão GRENJ-OA. Na Fase 2 houve inversão das versões anteriormente utilizadas pelos grupos.

Tabela 7.1 – Etapas do experimento.

Treinamento: Apresentação dos conceitos de programação orientada a aspectos, linguagem de padrões GRN, *framework* GRENJ-OO e a realização do piloto do experimento.		
Etapa 1 – Compreensão		
Grupo	**Fase 1**	**Fase 2**
Grupo 1	Questionário 1 Framework OO	Questionário 2 Framework OA
Grupo 2	Questionário 1 Framework OA	Questionário 2 Framework OO
Etapa 2 – Manutenção		
Grupo	**Fase 1**	**Fase 2**
Grupo 1	Estratégia de Recurso Instanciável Framework OA	Itemizar Transação do Recurso Framework OO
Grupo 2	Estratégia de Recurso Instanciável Framework OO	Itemizar Transação do Recurso Framework OA
Etapa 3 – Reutilização		
Grupo	**Fase 1**	**Fase 2**
Grupo 1	Biblioteca Framework OA	Aplicação Hotel - Framework OO
Grupo 2	Biblioteca Framework OO	Aplicação Hotel - Framework OA

f) Tipo de projeto. O experimento possui três fatores e três tratamentos aplicáveis a esses fatores (3*3 fatorial). Esse experimento foi realizado com doze participantes, que foram organizados em dois grupos com seis membros cada. A distribuição dos participantes nos grupos colocou a mesma quantidade de participantes experientes em cada grupo. O nível de experiência de cada participante foi avaliado com base na análise de questionários de caracterização de perfil (Apêndice C) respondidos por cada um. O questionário constou de um conjunto de questões relacionadas aos conceitos e técnicas necessárias para a realização do experimento (orientação a objetos, orientação a aspectos, linguagens Java e AspectJ, linguagem de padrões GRN, *frameworks* GRENJ-OO e GRENJ-OA e ambiente Eclipse). Os resultados obtidos com a aplicação desse questionário são exibidos no gráfico da Figura 7.1, tendo no eixo `x` os participantes e no eixo `y` o grau de acerto. Com base nas informações obtidas dos questionários, os participantes P2, P5, P8, P9 e P11 foram considerados como os mais experientes.

g) Instrumentação. Os documentos utilizados pelos participantes durante o experimento foram: questionário de caracterização de perfil; formulário de consentimento; questionários sobre as arquiteturas dos *frameworks* GRENJ-OO e GRENJ-OA; roteiros para a execução dos cenários de manutenção; modelos de classes das aplicações que foram instanciadas durante a realização do experimento; manuais de instanciação das versões GRENJ-OO e GRENJ-OA; tabelas para registro do tempo de execução das manutenções e das instanciações de aplicações nas versões utilizadas do GRENJ, os casos de teste das duas manutenções e instanciações e *scripts* de bases de dados. No Apêndice D estão questionários relacionados às arquiteturas das versões GRENJ-OO e GRENJ-OA. No Apêndice E estão o modelo de formulário de consentimento do experimento e exemplos de tabelas utilizadas para registrar os tempos para realizar manutenções e instanciar aplicações com as versões GRENJ-OO e GRENJ-OA. No Apêndice F estão os modelos de classes das aplicações, os manuais de instanciação das versões do *framework* e um modelo de roteiro para a execução de cenários de manutenção. No Apêndice G podem ser encontrados exemplos de casos de teste utilizados nas manutenções e instanciações realizadas e um exemplo de *script* de base de dados. Após a descrição e o planejamento do experimento passou-se à fase de execução como descrito na seção a seguir.

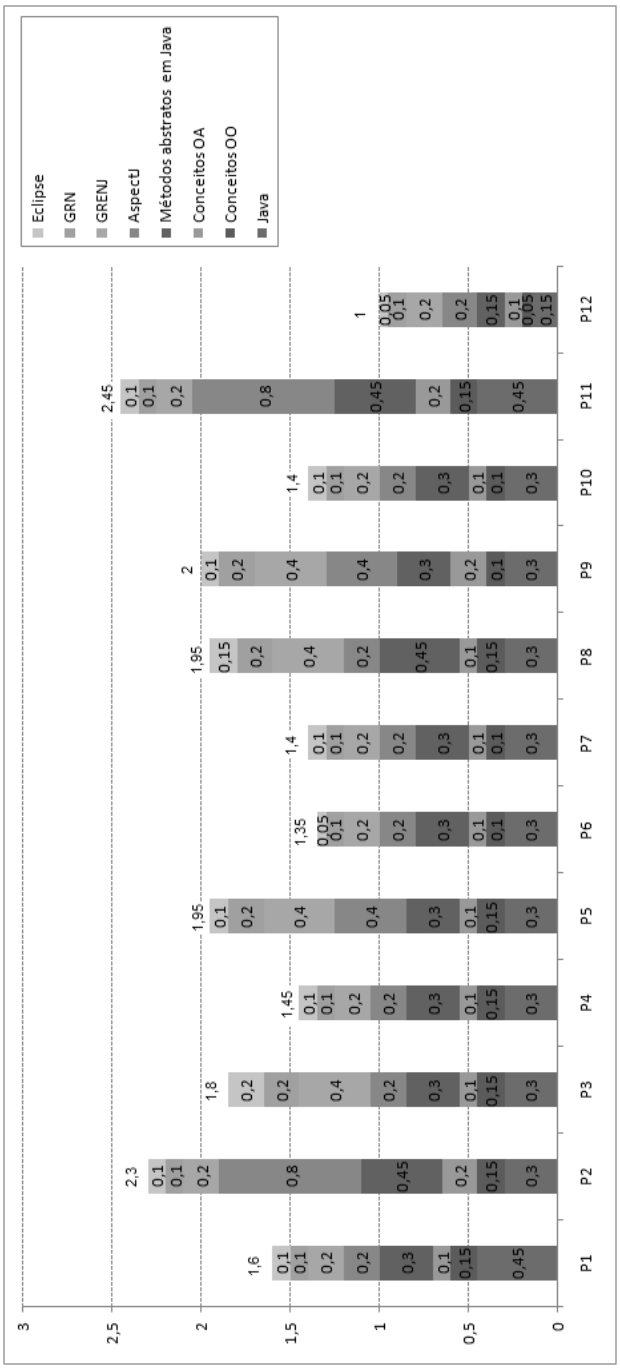

Figura 7.1. Experiência dos participantes do experimento.

7.2.1 Execução

Para a execução do experimento, inicialmente os participantes foram divididos em dois grupos, com seis membros cada um. Os participantes receberam um treinamento no qual foram apresentados todos os conceitos necessários, com o objetivo de nivelar o grau de conhecimento dos participantes nos conceitos e técnicas necessários para a realização do experimento. Em seguida, foi realizado o piloto do experimento, uma simulação do experimento real, para que todos os participantes tivessem conhecimento de sua estrutura para

que pudessem executá-lo de maneira correta. O experimento foi organizado em três etapas: "Compreensão", "Manutenção" e "Reutilização", sendo que cada etapa possui duas fases: uso da versão OO e uso da versão OA, que serão descritas a seguir.

Na etapa "Compreensão" o objetivo foi verificar a facilidade dos participantes em encontrar determinadas funções (padrões e variantes da GRN) no código fonte ao responderem aos questionários cujas questões se relacionavam às arquiteturas das versões OO e OA do *framework* e marcaram o tempo gasto para respondê-lo. Enquanto um grupo respondia as questões relacionadas à versão OO, o outro respondia as questões da versão OA.

Na etapa "Manutenção" o objetivo foi registrar o tempo gasto pelos participantes para a realização de cenários de manutenção nas versões OO e OA do *framework*. Como ocorreu na etapa "Compreensão", enquanto um grupo realizava manutenção na versão OO o outro realizava na versão OA.

Na etapa "Reutilização" o objetivo foi registrar o tempo gasto dos participantes para instanciar os *frameworks* para aplicações no domínio de biblioteca e hotel. Aqui também todos os participantes tiveram contato com as duas versões do *framework*.

Os cenários de manutenção e de instanciação das aplicações utilizados no experimento possuem graus de complexidade similares, para que esses fatores não influenciassem os resultados.

7.3 Dados Coletados

Nesta seção são apresentadas as informações coletadas após a realização do experimento que correspondem aos índices de acerto dos participantes em relação aos questionários sobre a arquitetura das versões OO e OA do *framework* GRENJ e o tempo de resposta a esses questionários, o tempo gasto por cada participante na execução dos cenários de manutenção e o tempo gasto por cada participante na instanciação das aplicações de Biblioteca e Hotel.

Na Tabela 7.2 são mostrados os dados coletados da primeira etapa do experimento (Compreensão), que correspondem aos índices de acerto e os tempos de cada participante do experimento ao responder os Questionários 1 e 2 (APÊNDICE D). Como já mostrado na Tabela 7.1, os grupos responderam às questões das duas versões do *framework*. Enquanto o

Grupo 1 respondia às questões sobre a versão OO o Grupo 2 respondia sobre OA, e vice versa.

Tabela 7.2. Etapa 1 do Experimento – Compreensão

Fase 1: Questionário 1														
	Framework OO Grupo 1							**Framework OA Grupo 2**						
Partici-pante	P1	P2	P3	P4	P5	P6	Média	P7	P8	P9	P10	P11	P12	Média
Núme-ro de acertos	1,3	2,8	2,4	1,5	2,2	2	2,03	2	4	3,5	2,6	2,6	0,3	2,5
Tempo	13:50	08:29	09:35	12:57	04:35	11:13	10:06	12:02	09:29	07:31	13:30	11:54	13:53	11:23

Fase 2: Questionário 2														
	Framework OA Grupo 1							**Framework OO Grupo 2**						
Partici-pante	P1	P2	P3	P4	P5	P6	Média	P7	P8	P9	P10	P11	P12	Média
Núme-ro de acertos	1,5	3	1,3	0,9	2,8	1,7	1,86	2,3	3,3	2,1	0,75	2,5	0,8	1,95
Tempo	15:00	11:22	13:08	15:00	12:25	12:32	13:14	15:00	08:54	08:18	12:00	13:00	14:58	12:01

Cada um desses questionários possui quatro questões relacionadas à arquitetura do *framework* GRENJ. Para cada uma dessas questões foi atribuído o mesmo peso (1). Analisando a tabela, em cada fase do experimento há o campo "Média", que refere-se às médias de acertos por grupo de participantes que responderam os questionários. Por exemplo, na Fase 1 da Etapa 1 do experimento, o grupo 1 respondeu ao Questionário 1 analisando a versão OO do *framework* GRENJ, obtendo uma média de 2,03 pontos de acerto em uma escala máxima de 4, já que cada questão possui peso 1.

Com relação à segunda etapa do experimento (Manutenção), foram elaborados dois cenários de manutenção, para os quais os Grupos 1 e 2 utilizaram as versões OO e OA do GRENJ para executá-los em cada fase desta etapa. Na Tabela 7.3 estão apresentados os tempos de execução das manutenções de cada participante em cada uma das fases desta etapa do experimento. Também é apresentada a média de tempo que cada grupo levou para executar as tarefas de cada cenário de manutenção.

Tabela 7.3. Etapa 2 do Experimento – Manutenção

Fase 1: Estratégia de Recurso Instanciável														
	Framework OA Grupo 1							**Framework OO Grupo 2**						
Participante	P1	P2	P3	P4	P5	P6	**Média**	P7	P8	P9	P10	P11	P12	**Média**
Tempo	04:06	04:52	12:55	15:00	04:16	15:00	09:21	14:00	05:52	05:40	06:30	06:43	13:10	08:39

Fase 2: Itemizar Transação do Recurso														
	Framework OO Grupo 1							**Framework OA Grupo 2**						
Participante	P1	P2	P3	P4	P5	P6	**Média**	P7	P8	P9	P10	P11	P12	**Média**
Tempo	06:22	05:39	06:00	07:00	03:58	15:00	07:19	09:00	06:00	04:18	07:00	04:13	15:00	07:35

Durante a realização das etapas 2 e 3 do experimento, os participantes foram instruídos a parar a cronometragem do tempo de execução da manutenção, caso encontrassem

problemas no ambiente de desenvolvimento. A retomada de cronometragem do tempo era realizada após a solução desse problema. Isso foi feito com o objetivo de evitar que fatores externos influenciassem os resultados do experimento.

Na terceira etapa do experimento (Reutilização) foram fornecidos modelos de classes de duas aplicações diferentes, uma de Hotel e outra de Biblioteca, as quais foram instanciadas com as versões OO e OA do GRENJ. Na Fase 1 desta etapa, o Grupo 1 utilizou o *framework* OA para instanciar a aplicação de Biblioteca, enquanto o Grupo 2 utilizou o *framework* OO para instanciar a mesma aplicação. Na Fase 2 esse processo se inverteu. Nesta etapa foram coletados os tempos de execução das instanciações de cada participante do experimento, como mostrado na Tabela 7.4, bem como a média dos tempos gasta por cada grupo para a conclusão da tarefa.

Tabela 7.4. Etapa 3 do Experimento – Reutilização

Fase 1: Aplicação Biblioteca														
	Framework OA Grupo 1							**Framework OO Grupo 2**						
Participante	P1	P2	P3	P4	P5	P6	**Média**	P7	P8	P9	P10	P11	P12	**Média**
Tempo	74:35	33:08	98:00	87:00	35:25	73:00	66:51	40:00	30:32	24:58	43:14	48:08	35:40	37:05

Fase 2: Aplicação Hotel														
	Framework OO Grupo 1							**Framework OA Grupo 2**						
Participante	P1	P2	P3	P4	P5	P6	**Média**	P7	P8	P9	P10	P11	P12	**Média**
Tempo	60:20	21:42	77:00	37:51	14:08	50:00	43:30	28:10	36:00	24:24	27:00	38:19	40:43	32:26

7.4 Análise dos Dados

Nesta seção é apresentada a análise dos dados de cada etapa do experimento, que correspondem às informações relacionadas à quantidade de acertos e os tempos gastos por participante para responder aos questionários da primeira etapa do experimento, os tempos em relação à execução dos cenários de manutenção e instanciações das aplicações nas segunda e terceira etapas do experimento. Na Seção 7.4.1 é apresentada a análise dos dados da "Etapa 1 – Compreensão" do experimento. Nas Seções 7.4.2 e 7.4.3 a análise dos dados das etapas "Manutenção" e "Reutilização" é descrita.

7.4.1 Primeira Etapa – Compreensão

Nesta etapa, cinco participantes que responderam aos questionários analisando a versão OA do *framework* obtiveram resultados inferiores em relação à versão OO. Isso pode ter ocorrido em conseqüência do baixo conhecimento dos participantes quanto a linguagem AspectJ e a versão OA do *framework* GRENJ. Também foi observado que sete participantes obtiveram maior número de acertos nas questões relativas à versão OA do *framework* GRENJ em relação a sua versão OO, como ilustrado na Figura 7.2. Isso se deve ao fato de que na versão OA, as funcionalidades do GRENJ estão separadas em aspectos e em pacotes, o que facilita a localização das mesmas.

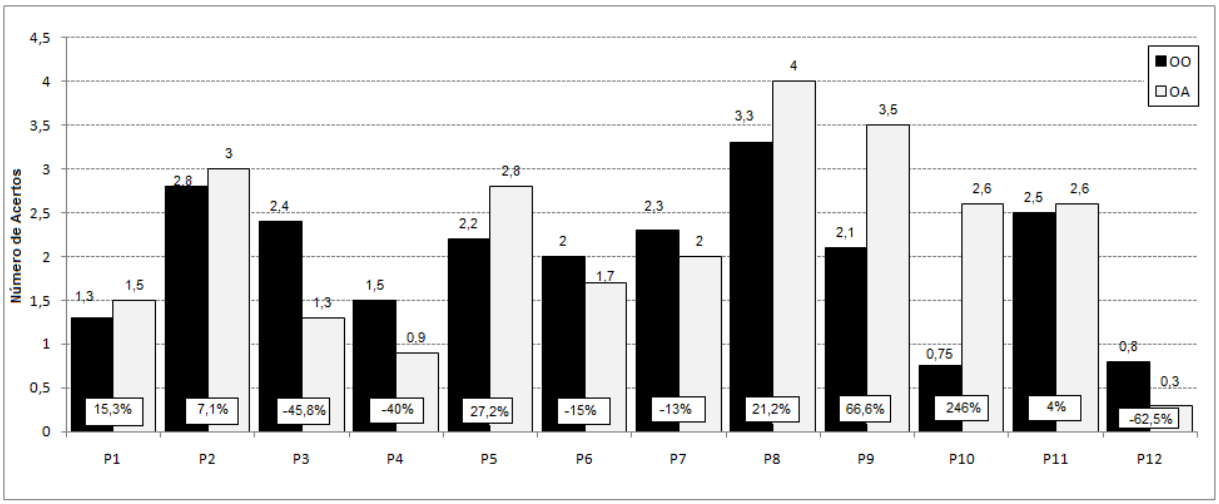

Figura 7.2. Número de acertos, por participante, nos questionários das versões OO e OA do GRENJ.

A média de acertos para a versão OO foi de 1,99 pontos e na versão OA foi de 2,18, o que representa uma superioridade de 9,5%. Desconsiderando os valores para os participantes com menos experiência em AspectJ e na versão OA do GRENJ, os resultados mostraram que a média de acertos para OA foi 23,2% superior à OO. Esse resultado fornece evidências que podem comprovar a hipótese de que o uso da orientação a aspectos na modularização do GRENJ-OO facilitou a compreensão de suas funcionalidades.

Com relação ao tempo despendido por cada participante ao responder aos questionários da primeira etapa, os resultados mostraram que o tempo de resposta aos questionários da versão OA foi maior para oito participantes em relação ao *framework* OO, como mostrado na Figura 7.3. Isso ocorre até mesmo para os participantes que possuem maior nível de conhecimento em relação do *framework* GRENJ-OO (P8 e P9), mas com diferenças mínimas entre os tempos de OO e OA, que não devem ser consideradas. Apenas quatro participantes (P7, P9, P11 e P12) levaram menos tempo para responder aos questionários do

framework OA. Um caso interessante é que apesar do tempo do participante P5 ter sido menor para responder ao questionário do *framework* OO o seu índice de acertos, como mostrado na Figura 7.2, foi 27,2% superior para OA em relação à OO. Isso mostra que apesar dos tempos para OA serem maiores em relação à OO, na maioria dos casos, na versão OA do GRENJ as funcionalidades estão mais bem localizadas, o que facilita a compreensão do *framework*.

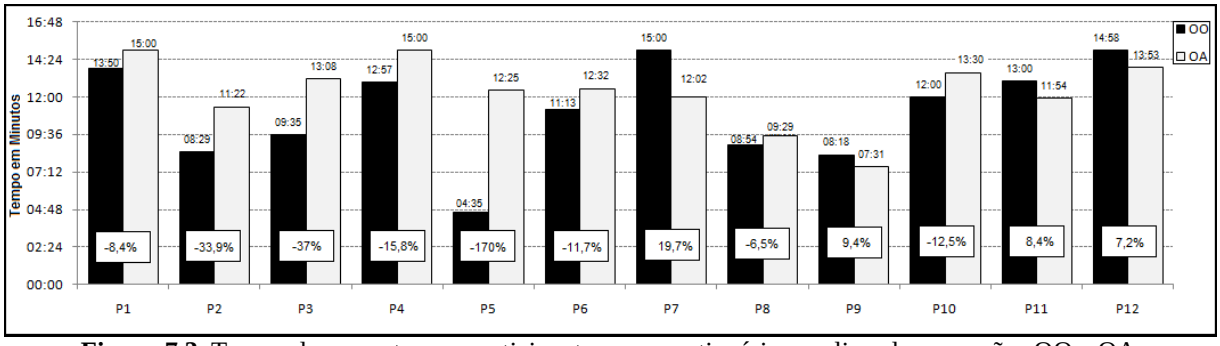

Figura 7.3. Tempo de resposta, por participante, aos questionários analisando as versões OO e OA do GRENJ.

7.4.2 Segunda Etapa - Manutenção

Nesta etapa foram coletados os tempos que cada participante levou para implementar os cenários de manutenção. A partir da análise desses dados foi verificado que cinco participantes (P1, P2, P7, P9 e P11) realizaram a manutenção em menor tempo na versão OA, como mostrado na Figura 7.4. Isso pode ser justificado pelo fato das funcionalidades do *framework* estarem implementadas em módulos coesos e com baixo acoplamento. Os resultados dos tempos para os participantes P5, P6, P8 e P10 mostraram uma diferença que varia de 0% a 7,6% a favor do *framework* OO em relação ao *framework* OA. Essa diferença não é suficiente para afirmar que o uso da orientação a aspectos dificulta a manutenção do GRENJ, mas com base nessas informações também não se pode afirmar o contrário. Os dados coletados também mostraram que para três participantes (P3, P4 e P12), os tempos para realizar manutenções no *framework* OA foram respectivamente 115,2%, 114,2% e 13,9% maiores (piores) em relação aos tempos para realizar manutenções no *framework* OO, o que é uma diferença significativa. Isso se justifica pelo fato dos participantes P3, P4 e P12 possuírem pouco conhecimento na linguagem AspectJ, como ilustrado no gráfico da Figura 7.1, que é a linguagem utilizada na versão OA.

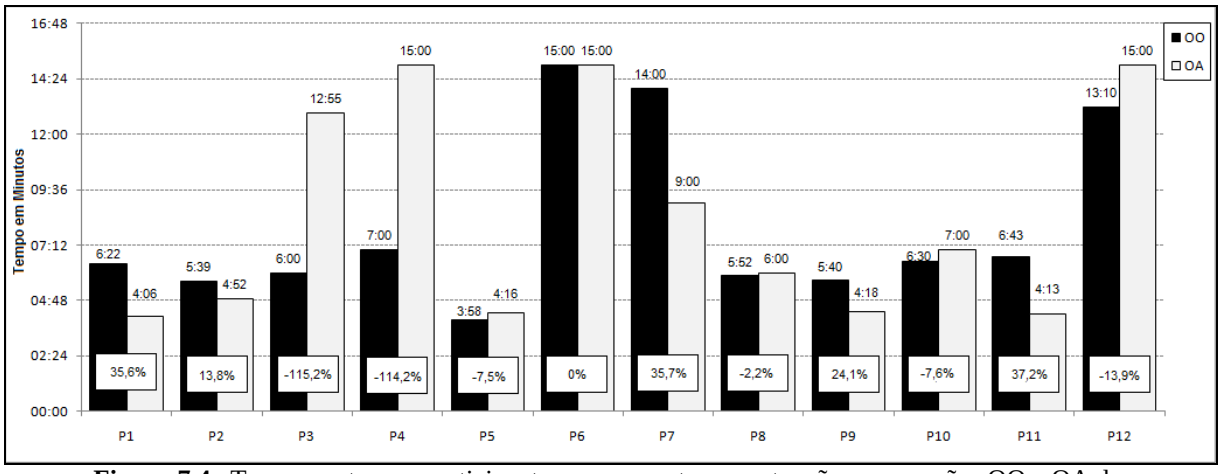

Figura 7.4. Tempo gasto, por participante, para executar manutenções nas versões OO e OA do GRENJ.

A média dos tempos dos participantes para executarem as manutenções na versão OO do *framework* GRENJ foi de 7 minutos e 59 segundos, enquanto que em sua versão OA foi de 8 minutos e 28 segundos. Esse resultado mostra que o tempo médio de manutenção na versão OO do GRENJ foi 6% melhor em relação à sua versão OA. Com essa diferença não é possível fazer qualquer afirmação em relação à manutenibilidade das versões GRENJ-OO e GRENJ-OA.

Considerando apenas os participantes mais experientes (P2, P5, P8, P9 e P11), os resultados mostraram que três participantes (P2, P9 e P11) gastaram menos tempo para realizar as manutenções no *framework* OA do que no OO. Para os participantes P5 e P8, os tempos de manutenção na versão OA do GRENJ apresentaram diferenças de 7,5% e 2,2%, respectivamente, em favor da OO. A média dos tempos entre os participantes mais experientes mostrou que o tempo para realizar manutenções utilizando o *framework* OA foi 15,1% menor em relação ao *framework* OO. Esses resultados evidenciam que a hipótese de que a modularização do *framework* GRENJ com orientação a aspectos melhorou sua manutenibilidade pode ser comprovada. Porém, para sua comprovação é necessário que esse experimento seja replicado em outras populações.

7.4.3 Terceira Etapa – Reutilização (Instanciação)

Nesta etapa os participantes instanciaram aplicações utilizando as versões OO e OA do *framework* GRENJ, sendo coletados os tempos. A partir da análise dos dados foi constatado que oito participantes (P1, P2, P3, P4, P5, P6, P8 e P12) levaram mais tempo para instanciar aplicações utilizando a versão OA do que a versão OO, como ilustrado na Figura 7.5. Somente quatro participantes obtiveram melhor tempo na versão OA. Isso ocorre em

virtude de dois fatores: a pouca experiência dos participantes na versão OA do GRENJ, como P3, P4, P6, e P12, e a alta experiência dos participantes P5, P8 e P9 na versão OO do GRENJ.

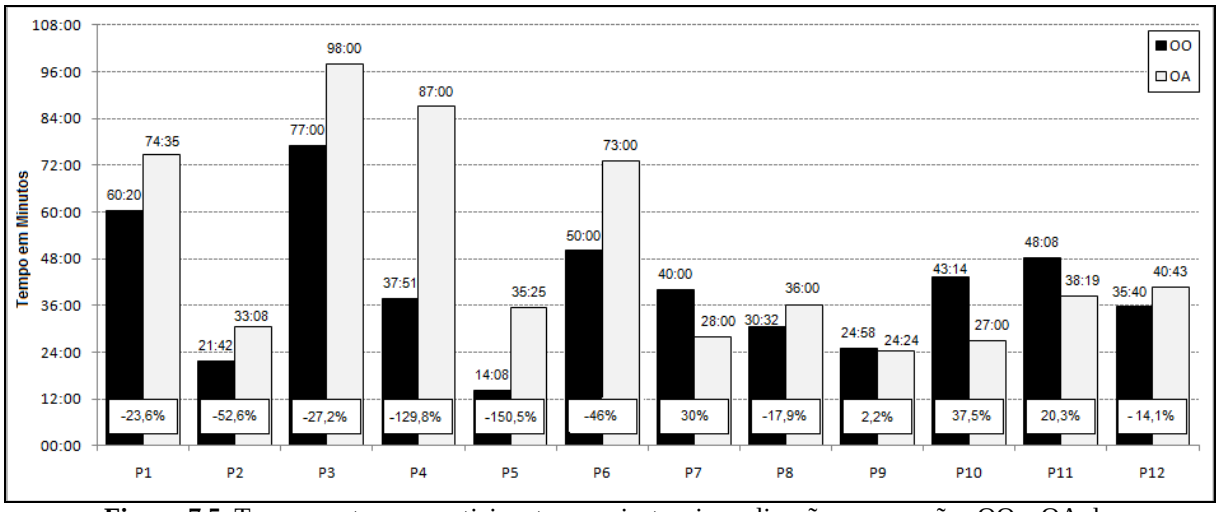

Figura 7.5. Tempo gasto, por participante, para instanciar aplicações nas versões OO e OA do GRENJ.

Um fato interessante é que mesmo os participantes P7 e P10 sendo menos experientes, como mostrado no gráfico da Figura 7.1, obtiveram tempos melhores para instanciar aplicações utilizando a versão OA em relação à OO. Apesar do resultado geral, que envolve todos os participantes do experimento, não ser favorável para OA, isso mostra, parcialmente, que a versão OA do GRENJ facilita o processo de instanciação de aplicações, por evitar que o engenheiro de aplicação tenha que fornecer implementações vazias para métodos não necessários aos requisitos da aplicação. Portanto, não há evidências suficientes para refutar a hipótese de que a modularização do *framework* GRENJ-OO com orientação a aspectos melhora a reusabilidade desse *framework*, mas também não há evidências suficientes para comprová-la.

Por exemplo, no GRENJ-OO para instanciar uma aplicação que contenha apenas o padrão Identificar Recurso e uma variante do padrão Quantificar Recurso é necessário fornecer implementações vazias para três métodos relacionados ao padrão Locar Recurso (`resourceRentalClass()`, `getResourceRentalClass()` e `getDestinationPartyClass()`) e para métodos relacionados a todas as variantes do Quantificar Recurso (`getMeasureUnityClass()` e `getResourceInstanceClass()`). Para instanciar essa mesma aplicação utilizando o GRENJ-OA não é necessário fornecer implementações para métodos relacionados ao padrão Locar Recurso e nem para métodos de todas as variantes do Quantificar Recurso, que deve implementar

`getMeasureUnityClass()` ou `getResourceInstanceClass()`, de acordo com a variante selecionada.

7.5 Resultados

A apresentação dos resultados obtidos com o experimento está dividida em três categorias: as médias obtidas em cada etapa do experimento considerando todos os participantes; as médias obtidas em cada etapa do experimento considerando os participantes mais experientes e as obtidas considerando os participantes menos experientes.

Considerando todos os participantes do experimento, os resultados mostraram evidências de que o uso da orientação a aspectos na modularização do *framework* GRENJ-OO melhorou a compreensão de suas funcionalidades. Essa afirmativa decorre da média geral de acertos dos questionários da versão OA ser 9,54% superior à OO, na etapa de "Compreensão" como mostrado na Tabela 7.5. Assim, há evidências de que a OA proporciona melhor localização das funcionalidades do *framework*. Com relação ao tempo para responder aos questionários, o tempo médio dos participantes que responderam aos questionários analisando o *framework* OA foi 11,14% inferior à OO. A razão para esse resultado se deve ao fato de que a arquitetura da versão GRENJ-OA está subdividida em vários pacotes e aspectos, para separar a implementação de suas *features* opcionais e alternativas. Apesar da facilidade de localização dessas *features* isso pode ter influenciado no tempo dos participantes. Esses resultados evidenciam que a hipótese "o uso da orientação a aspectos para modularizar o GRENJ-OO facilita a compreensão de suas funcionalidades" pode ser comprovada. Porém, é necessário que esse experimento seja replicado em outras populações para realmente comprovar a validade dessa hipótese.

Com relação à etapa "Manutenção" do experimento, as médias dos tempos considerando todos os participantes mostraram uma pequena diferença a favor da OO, como mostrado na Tabela 7.5. O tempo médio para realizar as tarefas de manutenção no GRENJ-OO foi 6% menor em relação ao tempo médio para a versão OA. Em virtude dessa pequena diferença, esse resultado não é suficiente para evidenciar a refutação da hipótese de que o uso da orientação a aspectos na modularização do *framework* GRENJ-OO melhorou sua manutenibilidade.

Tabela 7.5 - Resultados obtidos em cada etapa do experimento para todos os participantes.

Etapa 1 – Compreensão													
OO													
Participante	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	Média
Número de acertos	1,3	2,8	2,4	1,5	2,2	2	2,3	3,3	2,1	0,75	2,5	0,8	1,99
Tempo	13:50	08:29	09:35	12:57	04:35	11:13	15:00	08:54	08:18	12:00	13:00	14:58	11:04
OA													
Participante	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	Média
Número de acertos	1,5	3	1,3	0,9	2,8	1,7	2	4	3,5	2,6	2,6	0,3	2,18
Tempo	15:00	11:22	13:08	15:00	12:25	12:32	12:02	09:29	07:31	13:30	11:54	13:53	12:18
Etapa 2 – Manutenção													
OO													
Participante	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	Média
Tempo	06:22	05:39	06:00	07:00	03:58	15:00	14:00	05:52	05:40	06:30	06:43	13:10	07:59
OA													
Participante	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	Média
Tempo	04:06	04:52	12:55	15:00	04:16	15:00	09:00	06:00	04:18	07:00	04:13	15:00	08:28
Etapa 3 – Reutilização													
OO													
Participante	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	Média
Tempo	60:20	21:42	77:00	37:51	14:08	50:00	40:00	30:32	24:58	43:14	48:08	35:40	40:17
OA													
Participante	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	Média
Tempo	74:35	33:08	98:00	87:00	35:25	73:00	28:10	36:00	24:24	27:00	38:19	40:43	49:38

Para a etapa “Reutilização” o tempo médio dos participantes para instanciar aplicações utilizando a versão OA do *framework* GRENJ foi 23,21% maior em relação ao uso do GRENJ-OO. Esse resultado se justifica pelo fato de boa parte dos participantes não estar familiarizada com a nova estrutura de organização das funcionalidades na versão GRENJ-OA. Outro fator que contribuiu para esse resultado foi o alto nível de conhecimento dos participantes P5, P8 e P9 em relação ao GRENJ-OO e esses não estarem familiarizados com a arquitetura do GRENJ-OA. Em decorrência desses fatores, mesmo com a superioridade do uso do *framework* OO, acredita-se que não se pode evidenciar que a hipótese “o uso da orientação a aspectos para modularizar o *framework* GRENJ-OO melhora o seu potencial de reúso” deve ser refutada.

A análise dos resultados obtidos nas etapas do experimento considerando somente os participantes mais experientes fornece evidências de que o uso da orientação a aspectos na modularização do *framework* GRENJ-OO facilita a compreensão de suas funcionalidades e melhora sua manutenibilidade, como ilustrado na Tabela 7.6. Com relação à etapa “Compreensão”, a média de acertos dos questionários respondidos pelos participantes mais experientes analisando a versão OA do *framework* GRENJ foi 23,25% superior à OO, obtendo resultados melhores em relação à média de acertos considerando todos os participantes do experimento (Tabela 7.5). Esse resultado fornece evidências de que o uso da orientação a aspectos na modularização desse *framework* facilita a compreensão de suas

funcionalidades. Porém, o tempo médio que os participantes experientes levaram para responder os questionários analisando a versão OA do GRENJ foi 24,16% maior em relação ao GRENJ-OO. A justificativa para esse resultado é que esses participantes não estão familiarizados com a arquitetura da versão OA do GRENJ.

Tabela 7.6 - Resultados em cada etapa do experimento considerando os participantes mais experientes.

Etapa 1 – Compreensão						
OO						
Participante	P2	P5	P8	P9	P11	Média
Número de acertos	2,8	2,2	3,3	2,1	2,5	2,58
Tempo	08:29	04:35	08:54	08:18	13:00	08:39
OA						
Participante	P2	P5	P8	P9	P11	Média
Número de acertos	3	2,8	4	3,5	2,6	3,18
Tempo	11:22	12:25	09:29	07:31	11:54	10:32
Etapa 2 – Manutenção						
OO						
Participante	P2	P5	P8	P9	P11	Média
Tempo	05:39	03:58	05:52	05:40	06:43	05:34
OA						
Participante	P2	P5	P8	P9	P11	Média
Tempo	04:52	04:16	06:00	04:18	04:13	04:43
Etapa 3 – Reutilização						
OO						
Participante	P2	P5	P8	P9	P11	Média
Tempo	21:42	14:08	30:32	24:58	48:08	27:53
OA						
Participante	P2	P5	P8	P9	P11	Média
Tempo	33:08	35:25	36:00	24:24	38:19	33:15

Para a etapa "Manutenção" o tempo médio dos participantes experientes ao executarem cenários de manutenção na versão OA do GRENJ foi 15,26% menor em relação à OO. Isso ocorre em virtude do bom nível de conhecimento desses participantes na linguagem AspectJ. Esses resultados fornecem evidências de que a hipótese "o uso da orientação a aspectos na modularização do GRENJ-OO melhorou sua manutenibilidade" pode ser comprovada.

Apesar da versão OA do *framework* GRENJ facilitar a localização de suas funcionalidades e melhorar sua manutenibilidade, o mesmo não ocorre com relação à reusabilidade. Os resultados da etapa "Reutilização" do experimento considerando os participantes mais experientes mostraram que o tempo médio para instanciar aplicações utilizando a versão OA do GRENJ foi 19,96% maior em relação ao uso do GRENJ-OO para instanciar essas aplicações. A ocorrência desse resultado se deve ao fato de que os participantes não estão familiarizados com a organização em estrutura de pacotes das *features* que representam cada padrão e variante da GRN na versão OA desse *framework*. Esses

resultados fornecem evidências que apontam para a refutação da hipótese "o uso da orientação a aspectos na modularização do GRENJ-OO aumenta o seu potencial de reutilização". Porém, para comprovar ou refutar estatisticamente essa hipótese é necessária a replicação desse experimento em uma população que esteja familiarizada com a arquitetura de ambas as versões OO e OA do *framework* GRENJ.

Considerando os participantes menos experientes, os resultados para todas as etapas do experimento mostraram que o uso da orientação a aspectos na modularização do *framework* GRENJ-OO foi inferior quanto à facilidade de compreensão de suas funcionalidades, sua manutenibilidade e reusabilidade em relação ao GRENJ-OO, como mostrado na Tabela 7.7. Como mostrado nesta tabela, o índice de acertos da maioria dos participantes que responderam aos questionários analisando a versão GRENJ-OA foi inferior em relação à sua versão OO, exceto para os participantes P1 e P10, em que seus índices de acertos foram superiores para OA. Os resultados para esses dois participantes podem ser considerados um desvio no padrão em relação aos demais participantes. Em virtude desses resultados a média de acertos dos participantes considerando o *framework* OA foi 6,36% inferior em relação à OO. Com a ocorrência desse resultado e dos desvios acredita-se que não se pode comprovar ou refutar estatisticamente a hipótese de que "o uso da orientação a aspectos na modularização do GRENJ-OO facilita a localização de suas funcionalidades", sendo necessária a replicação desse experimento.

Com relação ao tempo médio que os participantes menos experientes do experimento levaram para responder os questionários analisando as versões OO e OA do GRENJ, os resultados mostraram que OA foi 6,25% inferior à OO, como ilustrado na Tabela 7.7. Comparando esse resultado com o obtido pelos participantes mais experientes (Tabela 7.6), pode-se concluir que os resultados para OA com relação ao tempo médio foram melhores para o grupo de participantes menos experientes. Porém, a média de acertos para OA dos participantes mais experientes foi melhor que a dos menos experientes. Com isso pode-se concluir que nem sempre o menor tempo indica uma melhor compreensão da arquitetura do *framework.*

Tabela 7.7 - Resultados em cada etapa do experimento considerando os participantes menos experientes.

Etapa 1 – Compreensão								
OO								
Participante	P1	P3	P4	P6	P7	P10	P12	Média
Número de acertos	1,3	2,4	1,5	2	2,3	0,75	0,8	1,57
Tempo	13:50	09:35	12:57	11:13	15:00	12:00	14:58	12:47
OA								
Participante	P1	P3	P4	P6	P7	P10	P12	Média
Número de acertos	1,5	1,3	0,9	1,7	2	2,6	0,3	1,47
Tempo	15:00	13:08	15:00	12:32	12:02	13:30	13:53	13:35
Etapa 2 – Manutenção								
OO								
Participante	P1	P3	P4	P6	P7	P10	P12	Média
Tempo	06:22	06:00	07:00	15:00	14:00	06:30	13:10	09:43
OA								
Participante	P1	P3	P4	P6	P7	P10	P12	Média
Tempo	04:06	12:55	15:00	15:00	09:00	07:00	15:00	11:08
Etapa 3 – Reutilização								
OO								
Participante	P1	P3	P4	P6	P7	P10	P12	Média
Tempo	60:20	77:00	37:51	50:00	40:00	43:14	35:40	49:09
OA								
Participante	P1	P3	P4	P6	P7	P10	P12	Média
Tempo	74:35	98:00	87:00	73:00	28:10	27:00	40:43	61:12

Os resultados para a etapa "Manutenção" do experimento considerando os participantes menos experientes mostraram que o tempo para realizar tarefas de manutenção na versão OA do *framework* GRENJ para quatro participantes foi maior em relação à versão OO, como mostrado na Tabela 7.7. Como ocorre na etapa "Compreensão", os tempos dos participantes P1 e P7 foram menores para OA, o que pode ser considerado um desvio. Mesmo com esse desvio, o tempo médio dos participantes para realizar tarefas de manutenção no GRENJ-OO foi 14,57% menor em relação à OA. Mesmo com esse resultado favorável para OO acredita-se que não se pode refutar a hipótese de que o uso da orientação a aspectos na modularização do *framework* GRENJ-OO melhora sua manutenibilidade, uma vez que os participantes menos experientes possuem pouco conhecimento na linguagem AspectJ, que é essencial para implementar cenários manutenção na versão OA desse *framework.* Em virtude dos desvios nos tempos dos participantes, também não se pode comprovar estatisticamente essa hipótese, sendo necessária a replicação desse experimento.

Para a etapa "Reutilização" do experimento considerando os participantes menos experientes, os resultados mostraram que o tempo para instanciar aplicações utilizando a versão OA do *framework* GRENJ para a maioria dos participantes foi maior em relação ao uso do GREN-OO para instanciar essas aplicações, exceto para os participantes P7 e P10, o que pode ser considerado um desvio no padrão. Apesar desse desvio, o tempo médio dos

participantes que utilizaram o GREN-OO para instanciar aplicações foi 24,51% menor em relação ao *framework* OA. Como comentado anteriormente nos resultados considerando todos os participantes, isso se deve ao fato dos participantes não estarem familiarizados com a arquitetura do *framework* GRENJ-OA. Isso evidencia que a hipótese "o uso da orientação a aspectos na modularização do GRENJ-OO melhora sua reusabilidade" não pode ser refutada. Porém, em virtude dos desvios, essa hipótese também não pode ser comprovada estatisticamente.

7.6 Ameaças à Validade do Estudo

Pelo fato do experimento ter sido aplicado a estudantes de pós-graduação e não a desenvolvedores de software, os resultados obtidos podem ter sido influenciados. A amostra de participantes foi pequena e se o mesmo experimento for aplicado com maior número de participantes pode-se obter resultados diferentes dos aqui apresentados.

Apesar da realização de um treinamento para nivelamento dos participantes em relação aos conceitos e técnicas utilizados no experimento, não há garantias de que todos os participantes possuem o mesmo nível de conhecimento e esse fator também pode ter influenciado nos resultados apresentados.

7.7 Considerações Finais

Neste capítulo foi descrito um experimento de manutenção realizado para comparar o *framework* GRENJ-OO com sua versão OA, modularizada durante a realização do estudo apresentado nesta dissertação. O objetivo do experimento foi analisar as versões OO e OA do *framework* GRENJ quanto à facilidade de compreensão de suas funcionalidades, à manutenibilidade e à reusabilidade. Como resultado foi possível evidenciar que o uso da orientação a aspectos para modularizar esse *framework* facilita a compreensão de suas funcionalidades, em virtude de proporcionar uma melhor localização das mesmas, e melhora a sua manutenibilidade, por reduzir o tempo para realizar tarefas de manutenção. Porém, com relação à reusabilidade os resultados mostraram que o uso da versão GRENJ-OA eleva o

tempo para instanciar aplicações em relação à GRENJ-OO. Isso se deve ao fato de os participantes não estarem familiarizados com a forma que as funcionalidades do GRENJ estão organizadas em sua versão OA.

Após a análise dos resultados obtidos com o experimento foi verificada a presença de desvios nos resultados de cada etapa para os participantes menos experientes, em que apresentaram melhores resultados para o GRENJ-OA em relação ao GRENJ-OO. Por essa razão não se pode comprovar estatisticamente as hipóteses de que o uso da orientação a aspectos na modularização do *framework* GRENJ-OO facilita a compreensão de suas funcionalidades, melhora sua manutenibilidade e reusabilidade. Para tanto é necessário que esse experimento seja replicado para garantir resultados mais concretos em relação à comprovação ou a refutação dessas hipóteses. No próximo capítulo são apresentadas as conclusões obtidas com a realização deste projeto, seus pontos positivos e negativos bem como sugestões para trabalhos futuros.

Capítulo 8

CONCLUSÕES

8.1 Considerações Finais

A modularização do *framework* de aplicação GRENJ-OO, que foi construído com base na linguagem de padrões GRN, utilizou orientação a aspectos para modularizar cada um desses padrões e suas variantes. Com o objetivo de desacoplar a implementação do núcleo do *framework* de seus elementos opcionais e alternativos, cada padrão e variante foi considerado como uma *feature*, gerando uma nova versão do *framework*, denominada GRENJ-OA. As aplicações baseadas no GRENJ-OA apresentam como ponto positivo que somente o código relacionado às *features* necessárias está presente na arquitetura da aplicação por ele instanciada.

A partir dessa modularização foi extraído um processo de modularização de *frameworks*, que os transformam em uma Linha de Produtos de *Frameworks*. Esse processo é aplicável na modularização de *frameworks* desenvolvidos com linguagens de padrões de análise. O conceito de Linha de Produtos de *Frameworks* consiste em uma linha de produtos na qual seus membros são *frameworks*, ao invés de aplicações de software. Os membros da linha são *frameworks* específicos que encapsulam um subconjunto de *features* do domínio da linha de produtos. Aplicando-se esse conceito ao GRENJ-OO foi possível gerar várias configurações específicas a partir da combinação de suas *features*.

Uma avaliação quantitativa indicou que a orientação a aspectos é satisfatória para modularizar a implementação de padrões em nível de análise, pela comparação do GRENJ-OO com o GRENJ-OA. Essa avaliação teve com base um conjunto de métricas de separação de interesses, acoplamento, coesão e tamanho (SANT'ANNA *et al.*, 2003). Os resultados dessa comparação mostraram que a utilização de OA melhorou os níveis de separação de

interesses, reduziu o acoplamento e aumentou a coesão da maioria dos padrões da GRN implementados no GRENJ-OO. Isso ocorreu, pois foram removidos o entrelaçamento e o espalhamento de interesses relacionados a cada um desses padrões pelas unidades do *framework*. Outro fator que contribuiu para esses resultados foi a separação das *features* opcionais e alternativas do núcleo do GRENJ-OO.

Na Seção 8.2 é apresentada uma comparação entre alguns dos trabalhos existentes na literatura que se relacionam com o que foi desenvolvido e aqui apresentado. Na Seção 8.3 são listadas as contribuições deste projeto e as limitações estão comentadas na Seção 8.4. Finalmente, na Seção 8.5 são listadas algumas sugestões para continuidade desta pesquisa.

8.2 Comparação com Trabalhos Relacionados

Alguns estudos encontrados na literatura se relacionam com o que foi apresentado nesta dissertação e evidenciam o estado da arte na utilização de orientação a aspectos na manutenção de *frameworks* e linhas de produtos. Avaliações quantitativas envolvendo orientação a aspectos também foram encontradas. Assim, nas Tabelas 8.1 e 8.2 é mostrada uma comparação dos resultados obtidos com o projeto realizado e os existentes na literatura especializada. A coluna 1 dessas tabelas tem referência ao trabalho a ser comparado, na coluna 2 há uma breve descrição sobre o conteúdo tratado nesses trabalhos e na coluna 3 encontra-se a comparação deles com o projeto desenvolvido.

Tabela 8.1 – Manutenção de Frameworks e Linha de Produtos de Software.

Trabalho	**Descrição**	**Semelhanças e diferenças entre os conceitos apresentados pelos autores e os utilizados nesta dissertação**
Kellens *et al.* (2008)	Uso da orientação a aspectos na implementação de regras de negócio de um sistema orientados a objetos.	**Semelhança**: utiliza orientação a aspectos para implementar requisitos funcionais. **Diferença**: o enfoque na modularização de padrões em nível de análise.
Zhang e Jacobsen (2004)	Cinco princípios para refatorar *frameworks* de modo a desacoplar a implementação de interesses transversais de seu núcleo. Esses princípios constituem a base do conceito de Decomposição Horizontal.	**Semelhanças**: o enfoque em separar *features* obrigatórias das demais funcionalidades do *framework* e o *refactoring* incremental de *features*. **Diferença**: maior enfoque na modularização de *features* opcionais e alternativas de *frameworks*.

Tabela 8.1 – Manutenção de Frameworks e Linha de Produtos de Software.

Godil e Jacobsen (2005)	Refactoring de um *framework* de persistência seguindo os princípios da Decomposição Horizontal.	**Semelhança:** trata da modularização das *features* de *frameworks* com orientação a aspectos. **Diferenças:** utilização da OA para separar a implementação de *features* opcionais e alternativas em frameworks construídos com linguagem de padrões de análise.
Camargo e Masiero (2008)	Propuseram uma abordagem para a criação de uma família de *frameworks* transversais.	**Semelhanças**: o enfoque em desacoplar a implementação de *features* opcionais e alternativas de *frameworks*. **Diferenças**: o enfoque na criação de uma linha de produtos de *frameworks* de aplicação enquanto os autores tratam da criação de uma família de *frameworks* transversais.

A manutenção do GRENJ-OO apresentou resultado satisfatório com o uso de OA para refatorar *frameworks* construídos com base em linguagens de padrões de análise. Essa utilização difere das encontradas na literatura e apresentadas na Tabela 8.1. Na Tabela 8.2 é apresentada a comparação tendo o enfoque em avaliação quantitativa.

Tabela 8.2 – Avaliação Quantitativa.

Trabalho	**Descrição**	**Semelhanças e diferenças do estudo desenvolvido nesta dissertação**
Oliveira *et al*. (2008) e Oliveira *et al.* (2008a)	Realizaram estudos quantitativos para avaliar as implementações OA do padrão *Data Access Object*.	**Semelhanças**: o enfoque na avaliação quantitativa. **Diferenças**: o enfoque na modularização de padrões de análise.
Oliveira *et al.* (2008b)	Propuseram uma arquitetura de software para o desenvolvimento de aplicações de ambiente multiplataforma e apresentaram um estudo comparativo com base em métricas para validá-la.	**Semelhanças**: o enfoque na avaliação quantitativa. **Diferenças**: o enfoque na modularização de padrões de análise.
Cacho *et al.* (2006)	Efetuaram uma avaliação quantitativa do uso da OA na modularização de composição de padrões de projeto.	**Semelhanças**: também trata com a composição de padrões. Possui o enfoque na avaliação quantitativa. **Diferença**: o enfoque na modularização de padrões de análise.
Garcia *et al.* (2005)	Realizaram uma avaliação quantitativa das implementações OA dos padrões de projeto GoF (GAMMA *et al*., 1995).	**Semelhanças**: a utilização das mesmas métricas para realizar a avaliação quantitativa. **Diferenças**: o enfoque na modularização de padrões de análise.

8.3 Contribuições

As contribuições desta dissertação foram:

- A constatação da efetividade do uso de OA para modularização de *frameworks* desenvolvidos com base em linguagem de padrões de análise, gerando o *framework* GRENJ-OA. Assim, houve melhoria na separação de interesses, redução do acoplamento e aumento da coesão das unidades do *framework*. Desse modo, houve melhoria na manutenibilidade e na reusabilidade do *framework*.
- A avaliação quantitativa da implementação de padrões de análise nas versões OO e OA dos padrões da GRN implementados no *framework*.
- O conceito de Linha de Produtos de *Frameworks* na qual seus membros são *frameworks* ao invés de aplicações de software. Esse conceito possibilita a obtenção de várias configurações específicas de um *framework* a partir da composição de suas *features*. Cada uma dessas configurações é um *framework* específico que inclui um subconjunto de *features*.
- A redução do código fonte na arquitetura final de aplicações construídas com Linha de Produtos de *Frameworks*. Somente as *features* necessárias para satisfazer os requisitos de uma determinada aplicação são incluídas evitando a presença de código não utilizado na arquitetura final da aplicação.
- A evolução incremental tanto da Linha de Produtos de *Frameworks* quanto das aplicações construídas com o seu apoio. A incorporação de novas *features* à linha de produtos de *frameworks* e aos *frameworks* específicos é possível, sem prejudicar a arquitetura da LPF, de seus *frameworks* específicos e de aplicações construídas com o apoio desses *frameworks*.
- A apresentação de um processo para modularização de *frameworks* construídos com base em uma linguagem de padrões de análise e transformá-los em uma Linha de Produtos de *Frameworks*. O uso desse processo possibilita a evolução de um *framework* de modo a facilitar a composição de suas unidades, desacoplando a implementação de suas *features* obrigatórias, opcionais e alternativas.

8.4 Limitações

As limitações identificadas neste projeto são:

- O aumento do número de unidades do *framework* GRENJ-OO provocado pelo uso de orientação a aspectos na sua modularização. Esse aumento provocou também o aumento de sua complexidade, dificultando a compreensão de suas funcionalidades. Para amenizar essa limitação, foi elaborado o diagrama de *features* do GRENJ-OA (APÊNDICE A) e uma tabela de mapeamento de suas variabilidades e as unidades que as implementam (APÊNDICE B).
- A inclusão de um passo a mais para a instanciação de aplicações utilizando o conceito de LPF aqui apresentado. Deve-se selecionar as *features* da LPF que serão utilizadas para desenvolver a aplicação, para depois criar as classes de aplicação que estendem as classes relacionadas a essas *features*. No GRENJ-OA, o engenheiro de aplicação necessita selecionar manualmente as *features* que serão utilizadas para construir a aplicação.
- A ausência de um mecanismo automatizado para a seleção das *features* do GRENJ-OA é outra limitação deste trabalho.
- A falta de integração das Camadas de Controle e de Interface Gráfica desenvolvidas por Viana (2009), considerando que o enfoque deste trabalho foi a modularização da Camada de Negócios do *framework* GRENJ-OO. Com isso, é possível reutilizar somente as classes da Camada de Negócios desse *framework*.

8.5 Sugestões de Trabalhos Futuros

Alguns trabalhos que podem dar continuidade ao aqui iniciado são:

- A integração da Camada de Negócios do GRENJ-OA com as Camadas de Controle e de Interface Gráfica existentes no GRENJ-OO.
- A implementação de um mecanismo que automatize o processo de seleção de *features* no GRENJ-OA para facilitar o trabalho do engenheiro de aplicação na geração de uma configuração mais específica desse *framework*.

- A realização de um estudo que investigue a composição de diferentes Linhas de Produtos de *Frameworks*. Isso consiste na possibilidade de combinar *features* de duas ou mais diferentes linhas de produtos de *frameworks* para gerar *frameworks* específicos que contenham *features* de ambas as linhas.
- A condução de estudos comparativos quanto à implementação dos padrões da GRN no *framework* GRENJ-OO utilizando a linguagem AspectJ e linguagens orientadas a *features.* Exemplos de linguagens e ferramentas orientadas a *features* são: CaesarJ (MEZINI e OSTERMANN, 2003), AHEAD (BATORY *et al*., 2004) e Feature C++ (APEL *et al.*, 2005). Esses estudos possibilitam verificar a efetividade, as vantagens e desvantagens do uso dessas linguagens e ferramentas na modularização de *frameworks* construídos com base em linguagens de padrões.
- A formalização de um processo de desenvolvimento de Linhas de Produtos de *Frameworks*. Esse processo tem como objetivo apresentar um conjunto de atividades que devem ser executadas para projetar uma Linha de Produtos de *Frameworks* a partir de especificações de requisitos e do conhecimento de domínio.

REFERÊNCIAS

APEL, S.; KASTNER, C.; LENGAUER, C. FEATUREHOUSE: Language-independent, automated software composition. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE'09), 31st, 2009, Vancouver, Canadá. **Proceedings…** 2009. p. 221-231.

APEL, S. et al. FeatureC++: on the symbiosis of feature-oriented and aspect-oriented programming. In: INTERNATIONAL CONFERENCE ON GENERATIVE PROGRAMMING AND COMPONENT ENGINEERING (GPCE'05), 4th, 2005, Tallinn, Estonia. **Proceedings…** Springer, Heidelberg, 2005. v. 3676, p. 125-140.

ASPECTJ PROJECT. **The AspectJ project**. 2010. Disponível em: <http://www.eclipse.org/aspectj/>. Acesso em: 18 jul. 2010.

BASILI, V. R.; SELBY, R. W.; HUTCHENS, D. H. Experimentation in software engineering. **IEEE Transactions on Software Engineering**, v. 12, n. 7, p. 733-743, 1986.

BATORY D. S.; SARVELA, J. N.; RAUSCHMAYER, A. Scaling step-wise refinement. **IEEE Transactions on Software Engineering**, v. 30, n. 6, p. 355-371, 2004.

BECK, K. et al. Industrial experience with design patterns. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE'96), 18th, 1996, Berlin, Alemanha. **Proceedings…** IEEE Computer Society, Washington, 1996. p. 103–114.

BOOCH, G.; RUMBAUNGH, J.; JACOBSON, I. **Unified modeling language user guide**. 2nd ed. Addison-Wesley Professional, 2005. 496 p.

BRAGA, R. T. V. **Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico**. 216 p. Tese (Doutorado em Ciência da Computação) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2002. Disponível em: < http://www.teses.usp.br/teses/disponiveis/55/55134/tde-12052003-102000/publico/TeseFinal Corrigida.pdf>. Acesso em: 07 jun. 2010.

BRAGA, R. T. V.; MASIERO, P. The role of pattern languages in the instantiation of white-box object-oriented frameworks. **Cadernos de Computação do Instituto de Ciências Matemáticas e de Computação**, São Carlos, v. 3, p. 119–145, 2002.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. A pattern language for business resource management. In: ANNUAL CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS (PLOP'99), 6th, 1999, Monticello, Illinois, EUA. **Proceedings…** 1999. v. 7, p. 1-33.

BRUGALI, D.; SYCARA, K. Frameworks and pattern languages: an intriguing relationship. **ACM Computing Surveys**, Nova York, v. 32, n. 2, 2000.

CACHO, N. et al. Composing design patterns: a scalability study of aspect-oriented programming. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT (AOSD'06), 5th, 2006, Bonn, Alemanha. **Proceedings…** 2006. p. 109-121.

CAMARGO, V. V.; MASIERO, P. C. An approach to design crosscutting framework families. In: 2008 AOSD WORKSHOP ON ASPECTS, COMPONENTS, AND PATTERNS FOR INFRASTRUCTURE SOFTWARE (ACP4IS'08), 2008, Bruxelas, Bélgica. **Proceedings...** 2008. p. 1-6.

CAMARGO, V. V. de; NINA, E.; MALDONADO, J. C. Um estudo comparativo do tempo de composição de um framework orientado a aspectos de persistência e de um framework orientado a objetos de persistência. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES'06), 20, 2006, Florianópolis, Brasil. **Anais...** 2006. p. 193-208**.**

CAMARGO, V. V. **Frameworks transversais: definições, classificações, arquitetura e utilização em um processo de desenvolvimento de software.** 256 p. Tese (Doutorado em Ciência da Computação) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos-SP, 2006. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-01112006-142356/pt-br.php>. Acesso em: 14 jul. 2010.

CHANGE VISION. **Astah UML modeling tool**. 2010. Disponível em < http://jude.change-vision.com/jude-web/index.html>. Acesso em: 15 ago. 2010.

CHAVEZ, C.; GARCIA, A.; LUCENA, C. Desenvolvimento de software orientado a Aspectos. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES'03), 17, 2003, Manaus, Amazonas. **Anais...** Universidade Federal do Amazonas, 2003.

CHIDAMBER, S.; KEMERER, C. A metrics suite for object oriented design. **IEEE Transactions on Software Engineering**, v. 20, n. 6, p. 476-493, 1994.

COSTA, H. A. et al. Recovering class models stereotyped with crosscutting concerns. In: WORKING CONFERENCE ON REVERSE ENGINEERING (WCRE'09), 16th, 2009, Lille, França. **Proceedings...** IEEE Computer Society, Washington, DC, 2009. p. 311-312.

DURELLI, V. **Um processo de reengenharia ágil de frameworks orientados a objetos**. 128 p. Dissertação (Mestrado em Ciência da Computação) - Departamento de Computação, Universidade Federal de São Carlos, 2008.

FENTON, N.; PFLEEGER, S. **Software metrics: a rigorous and practical approach**. 1 ed., London, PWS Publishing Co, 1997. 638 p.

FIGUEIREDO, E. et al. Evolving software product lines with aspects: an empirical study on design stability. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE'09), 30th, 2008, Leipzig, Alemanha. **Proceedings…** ACM Nova York, 2008. p. 261-270.

GAMMA, E. et al. **Design patterns: elements of reusable object-oriented software**. Addison-Wesley Professional Computing Series, 1995. 416 p.

GARCIA, A. F. et al. Modularizing design patterns with aspects: a quantitative study. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT (AOSD'05), 4th, 2005, Chicago, USA. **Proceedings…** ACM Press, 2005. p. 3-14.

GARCIA, A. F. et al. Separation of concerns in multi-agent systems: an empirical study. In: LUCENA, C.; GARCIA, A.; ROMANOVSKY, A.; CASTRO, J.; ALENCAR, P. (Org.). **Software engineering for multi-agent systems II - research issues and practical applications**. Berlin-Heidelberg, Springer-Verlag, 2004, v. 2940, p. 49-72.

GODIL, I.; JACOBSEN, H. Horizontal decomposition of Prevayler. In: 2005 CONFERENCE OF THE CENTRE FOR ADVANCED STUDIES ON COLLABORATIVE RESEARCH, 2005, Toronto, Ontario, Canadá. **Proceedings…** IBM Press, 2005. p. 83-100.

GOMAA, H. **Designing software product lines with UML – from use cases to pattern-based software architectures**. 1 ed. Addison Wesley Longman Publishing Co, 2004. 736 p.

GOSLING, J. et al. **The Java language specification**. 2 ed. Prentice Hall, 2000, 544 p.

GRISS, M. L. Implementing product-line features with component reuse. In: INTERNATIONAL CONFERENCE ON SOFTWARE REUSE: ADVANCES IN

SOFTWARE REUSABILITY, 6th, 2000, Vienna, Áustria. **Proceedings…** Springer-Verlag, 2000. v. 1844, p. 137-152.

HANNEMANN, J.; KICZALES, G. Design pattern implementation in Java and AspectJ. In: ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA'02), 17th, 2002, Seattle, EUA. **Proceedings…** ACM, Nova York, EUA, 2002. v. 37, n. 11, p. 161-173.

HANENBERG, S.; SCHMIDMEIER, A.; UNLAND, R. AspectJ idioms for aspect-oriented software construction. In: EUROPEAN CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS (EUROPLOP 2003), 8th, 2003, Irsee, Alemanha. **Proceedings…** 2003.

HUNT, J. M.; MCGREGOR, J. D. Software product lines: a pedagogical application. **Journal of Computing Sciences in Colleges**. v. 22, n. 2, p. 295-302, 2006.

JOHN, I., MUTHIG, D. Product line modeling with generic use cases. In: WORKSHOP ON TECHNIQUES FOR EXPLOITING COMMONALITY THROUGH VARIABILITY MANAGEMENT, 1st, 2002, San Diego. **Proceedings of the Second Software Product Line Conference.** 2002.

KANG, K. C., LEE, J. DONOHOE, P. Feature-oriented product line engineering. **IEEE Software**. v. 19, n. 4, p. 58-65, 2002.

KANG, K. C. et al. **Feature Oriented Domain Analysis (FODA) feasibility study**. Software Engineering Institute, Carnegie Mellon University, 1990. 148 p. Relatório Técnico. Disponível em: < http://www.sei.cmu.edu/reports/90tr021.pdf >. Acesso em: 10 ago. 2010.

KELLENS, A et al. Experiences in modularizing business rules into aspects. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE (ICSM'08). 24th, 2008, Beijing, China. **Proceedings…** IEEE Computer Society, 2008. p. 448-451.

KICZALES, G. et al. An overview of AspectJ. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, 15th, 2001, Budapeste, Hungria. **Proceedings…** Springer-Verlag, Londres, 2001. p. 327-354.

KICZALES, G. et al. Aspect-oriented programming. In: EUROPEAN CONFERENCE OBJECT ORIENTED PROGRAMMING (ECOOP'97). 11th, Jyväskylä, Finlândia, 1997. **Proceedings…** Springer-Verlag, 1997. p. 220-242.

LADDAD, R. **AspectJ in action: practical aspect-oriented programming.** Manning Publications, 2003, 512 p.

MEYER, B. **Object–oriented software construction**. 2 ed. Prentice–Hall, 1997, 1254 p.

MEZINI, M.; OSTERMANN, K. Conquering aspects with Caesar. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT (AOSD '03), 2nd, 2003, Boston, Massachusetts. **Proceedings…** ACM, Nova York, 2003. p. 90-99.

MYSQL. **MySQL: the world's most popular open source database**. 2010. Disponível em: <http://www.mysql.com>. Acesso em: 05 mai. 2010.

OLIVEIRA, A. L. de; PENTEADO, R. A. D. **Modularização dos padrões da linguagem GRN utilizando orientação a aspectos**. Departamento de Computação, Universidade Federal de São Carlos, 2010, 91 p. Documento de Trabalho. Disponível em: <http://gbd.dc.ufscar.br/~andre/modularizacao.oliveira-penteado.pdf>. Acesso em: 02 ago. 2010.

OLIVEIRA, A. L. de; CAMARGO, V. V. de; PENTEADO, R. A. D. Manutenção de frameworks orientados a objetos utilizando orientação a aspectos. In: WORKSHOP DE MANUTENÇÃO DE SOFTWARE MODERNA (WMSWM'10), 7, 2010, Belém, Pará, Brasil. **Anais do IX Simpósio Brasileiro de Qualidade de Software (SBQS)**. Universidade Federal do Pará, 2010.

OLIVEIRA, A. L. de., MENOLLI, A. L. A., COELHO, R. G. Implementing the Data Access Object pattern using AspectJ. In: INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS, 10th, 2008, Barcelona, Espanha. **Proceedings…** 2008. p. 523-530.

OLIVEIRA, A. L. de. et al. Um estudo quantitativo das implementações orientadas a aspectos do padrão Data Access Object. In: LATIN AMERICAN WORKSHOP ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT (LA-WASP'08), 2, 2008a, Campinas, São Paulo, Brasil, **Anais do XXII Simpósio Brasileiro de Engenharia de Software (SBES)**. Universidade Estadual de Campinas (UNICAMP), 2008. p. 80-89.

OLIVEIRA, A. L. de.; MENOLLI, A. L. A.; COELHO, R. G. A proposal of software architecture for multiplatform environment applications development, a quantitative study. In: INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS, 10th, 2008b. Barcelona, Espanha. **Proceedings…** 2008. p. 397-404.

OSSHER, H.; TARR, P. Using subject–oriented programming to overcome common problems in object–oriented software development/evolution. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE'99), 21st, 1999, Los Angeles, EUA. **Proceedings…** ACM, Nova York, 1999. p. 687–688.

PACIOS, S. F. **Uma abordagem orientada a aspectos para desenvolvimento de linhas de produtos de software**. Dissertação (Mestrado em Ciência da Computação) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2006. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08052007-171603/pt-br.php >. Acesso em 02 ago. 2010.

PARREIRA JÚNIOR, P. et al. Comscid - um apoio computacional customizável para a identificação de interesses transversais em sistemas legados orientados a objetos. In: WORKSHOP DE MANUTENÇÃO DE SOFTWARE MODERNA (WMSWM'10), 7, 2010, Belém, Pará, Brasil. **Anais do IX Simpósio Brasileiro de Qualidade de Software (SBQS)**. Universidade Federal do Pará, 2010.

PREVAYLER. **Prevayler framework**. 2010. Disponível em: < http://www.prevayler.org/>. Acesso em: 17 ago. 2010.

ROSS, D. T. Structured Analysis (SA): A language for communicating ideas. **IEEE Transactions on Software Engineering**. Piscataway, Nova Jersey, EUA. v. 3, n. 1, p. 16-34. 1977.

SANT'ANNA, C. et al. On the reuse and maintenance of aspect-oriented software: an assessment framework. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES'03), 17, 2003, Manaus, Amazonas, Brasil. **Anais…** 2003. p. 19-34.

SCHMID, K. A comprehensive product line scoping approach and its validation. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE'02), 24th, 2002, Orlando, Florida, EUA. **Proceedings…** ACM, Nova York, EUA, 2002. p. 593-603.

SEI/CMU. **A framework for software product line practice, version 5.0**. Software Engineering Institute, Carnegie Mellon University. Disponível em: < http://www.sei.cmu.edu/productlines/framework.html#framework_toc>. Acesso em: 05 ago. 2010.

SHALLOWAY, A.; TROTT, J. **Design patterns explained: a new perspective on object-oriented design**. 1 ed. Addison-Wesley, 2001, 368 p.

SMALLTALK DOT ORG. **Smalltalk dot org: community and industry meet inventing the future.** 2010. Disponível em: <http://www.smalltalk.org/main>. Acesso em: 10 jul. 2010.

SOARES, S., BORBA, P. AspectJ: programação orientada a aspectos em Java. In: SIMPÓSIO BRASILEIRO DE LINGUAGENS DE PROGRAMAÇÃO (SBLP'02), 6, 2002, Rio de Janeiro. **Anais...** 2002.

SUN MICROSYSTEMS. **Core J2EE patterns: Data Access Object**. 2002. Disponível em: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>. Acesso em: 10 jul. 2010.

THE ECLIPSE FOUNDATION. **Eclipse tool**. 2010. Disponível em: <www.eclipse.org>. Acesso em 15 mai. 2010.

VAN DEURSEN, A.; KLINT, P. Domain-specific language design requires feature descriptions. **Journal of Computing and Information Technology**. v. 10 n. 1. p. 1-17, 2001.

VIANA, M. C. **Construção da camada de interface gráfica e de um wizard para o framework GRENJ**. 117 p. Dissertação (Mestrado em Ciência da Computação) – Departamento de Computação, Universidade Federal de São Carlos, 2009.

WOHLIN, C et al. **Experimentation in software engineering: an introduction**. Kluwer Academic Publishers, 2000, 212 p.

ZANON, I. B. **Uso de um framework transversal na camada de persistência do GRENJ**. 93 p. Dissertação (Mestrado em Ciência da Computação) – Departamento de Computação, Universidade Federal de São Carlos, 2009.

ZHANG, C.; JACOBSEN, H. Resolving feature convolution in middleware systems. In: ANNUAL ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA '04), 19th, 2004, Vancouver, Canadá. **Proceedings...** ACM, Nova York, EUA, 2004. p. 188-205.

Apêndice A

DIAGRAMA DE FEATURES DO FRAMEWORK GRENJ-OA

Neste documento é apresentado o diagrama de *features* da versão orientada a aspectos do *framework* GRENJ. Em virtude da grande quantidade de *features* presentes na versão orientada a aspectos do *framework* GRENJ, a apresentação desse diagrama está dividida em seis figuras. Esse diagrama segue a notação de modelagem de *features* baseada em UML proposta por Gomaa (2004). Na Figura 1 são apresentadas as *features* que compõem o núcleo do *framework* GRENJ, juntamente com as *features* relacionadas aos padrões Identificar Recurso e Quantificar Recurso, que pertencem ao grupo de Identificação do Recurso de Negócio da GRN.

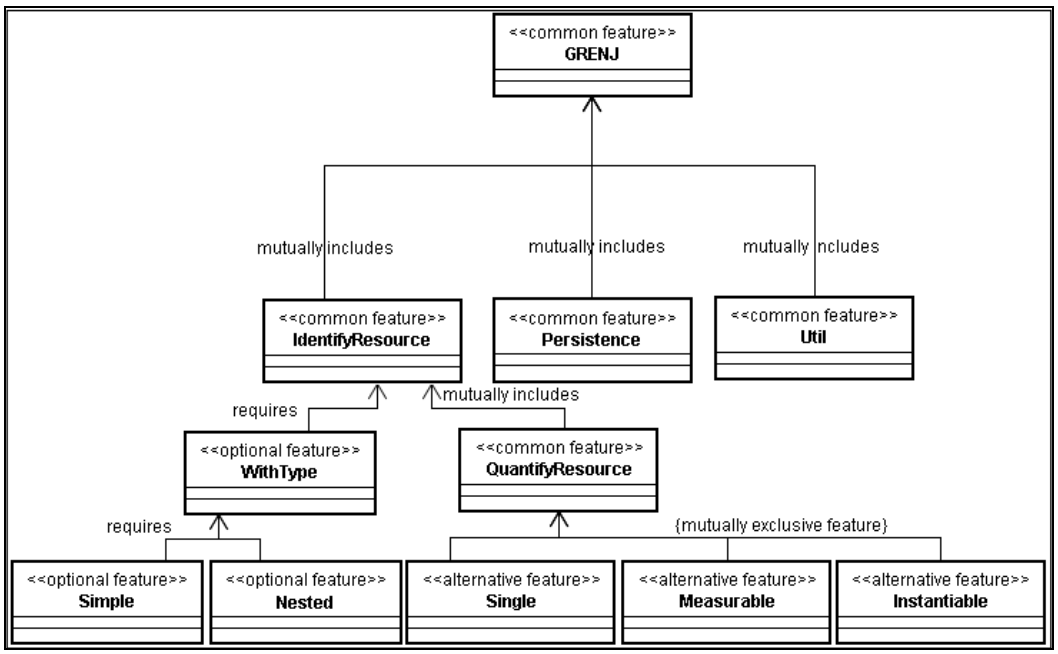

Figura 1. *Features* relacionadas ao núcleo do *framework* GRENJ e aos padrões Identificar Recurso e Quantificar Recurso da GRN.

Na Figura 2 as *features* relacionadas aos padrões Locar Recurso e Reservar Recurso, do grupo Transações do Negócio da GRN, são apresentadas.

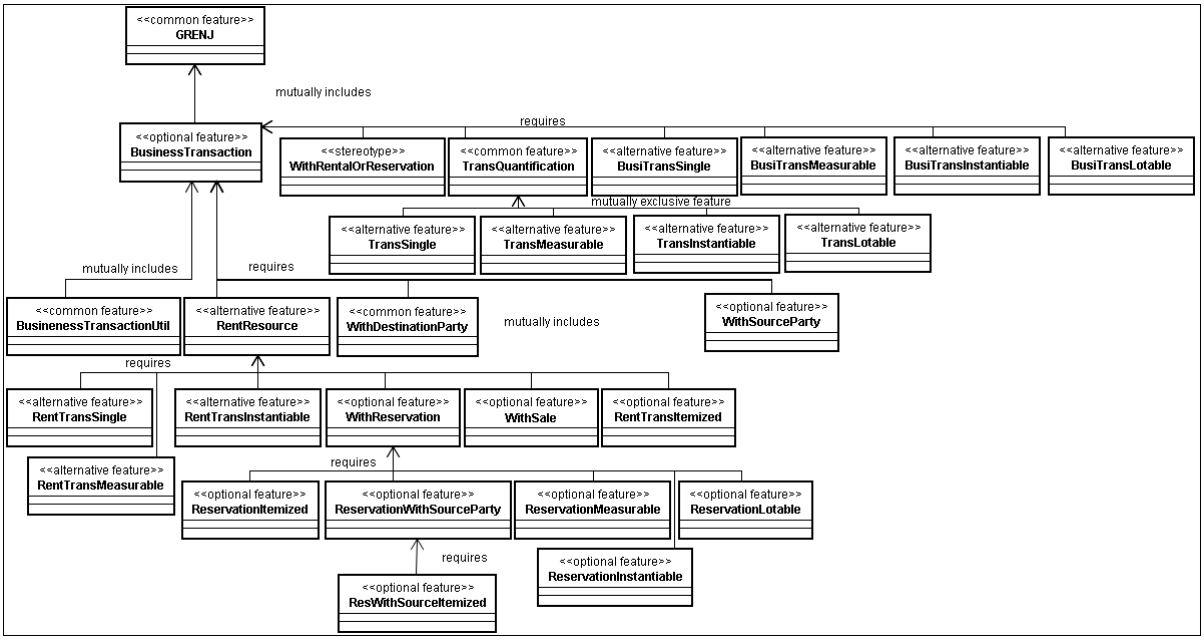

Figura 2. *Features* relacionadas aos padrões Locar Recurso e Reservar Recurso da GRN.

Na Figura 3 são apresentadas as *features* relacionadas aos padrões Comercializar Recurso e Conferir a Entrega do Recurso, que pertencem ao grupo Transações do Negócio da GRN.

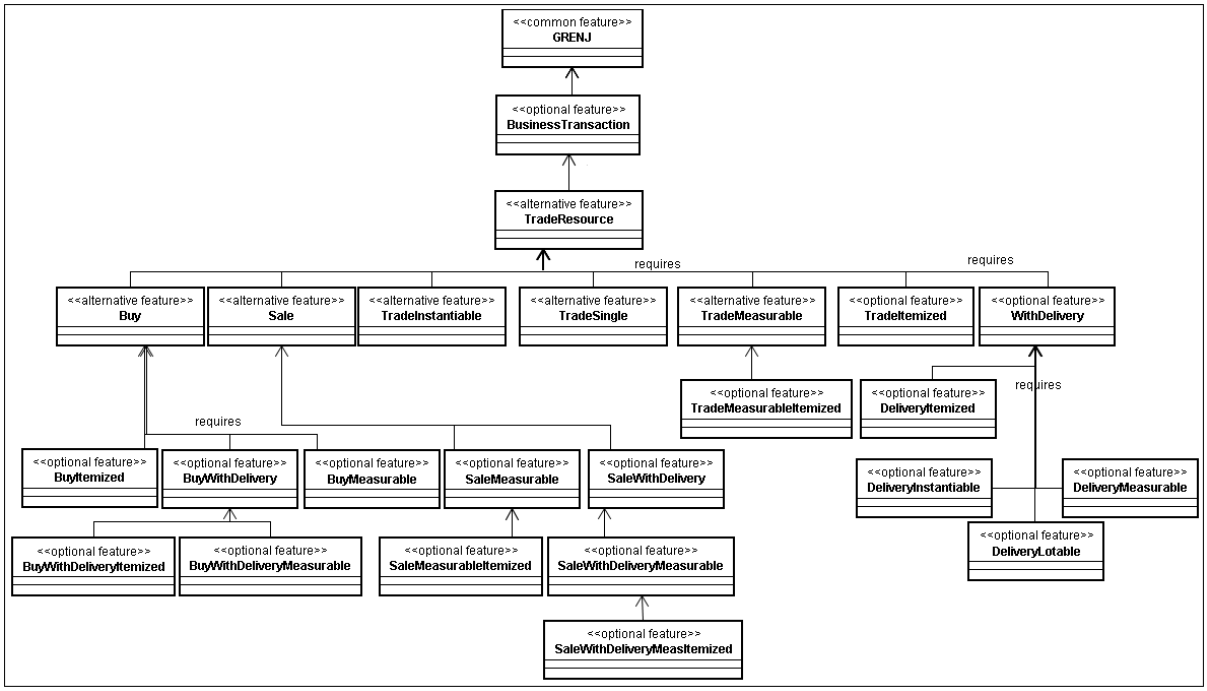

Figura 3. *Features* relacionadas aos padrões Comercializar Recurso e Conferir a Entrega do Recurso da GRN.

Na Figura 4 são apresentadas as *features* relacionadas aos padrões Cotar Recurso, Manter Recurso e Cotar a Manutenção do Recurso, que pertencem ao grupo Transações do Negócio da GRN, Identificar Peças de Manutenção e Identificar Tarefas de Manutenção, do grupo Detalhes das Transações do Negócio da GRN.

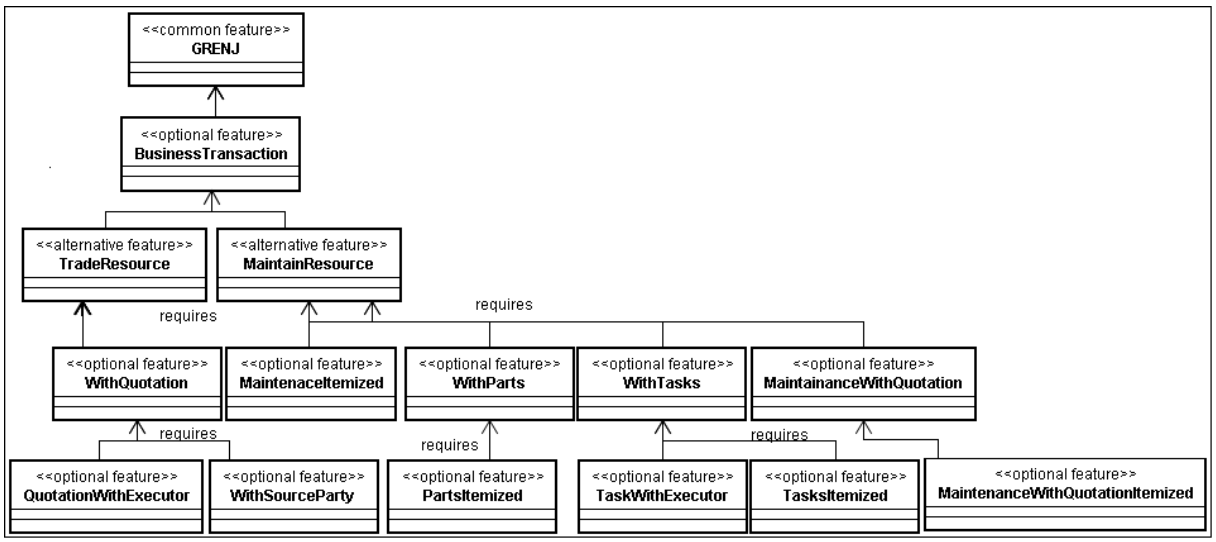

Figura 4. *Features* relacionadas aos padrões Cotar Recurso, Manter Recurso, Cotar a Manutenção do Recurso, Identificar Peças de Manutenção e Identificar Tarefas de Manutenção da GRN.

Na Figura 5 são apresentadas as *features* relacionadas aos padrões Itemizar a Transação do Recurso e Identificar o Executor da Transação, que pertencem ao grupo Detalhes das Transações do Negócio da GRN. Nesse diagrama também são apresentadas as *features* comuns entre os padrões Locar Recurso e Pagar pela Transação do Recurso da GRN.

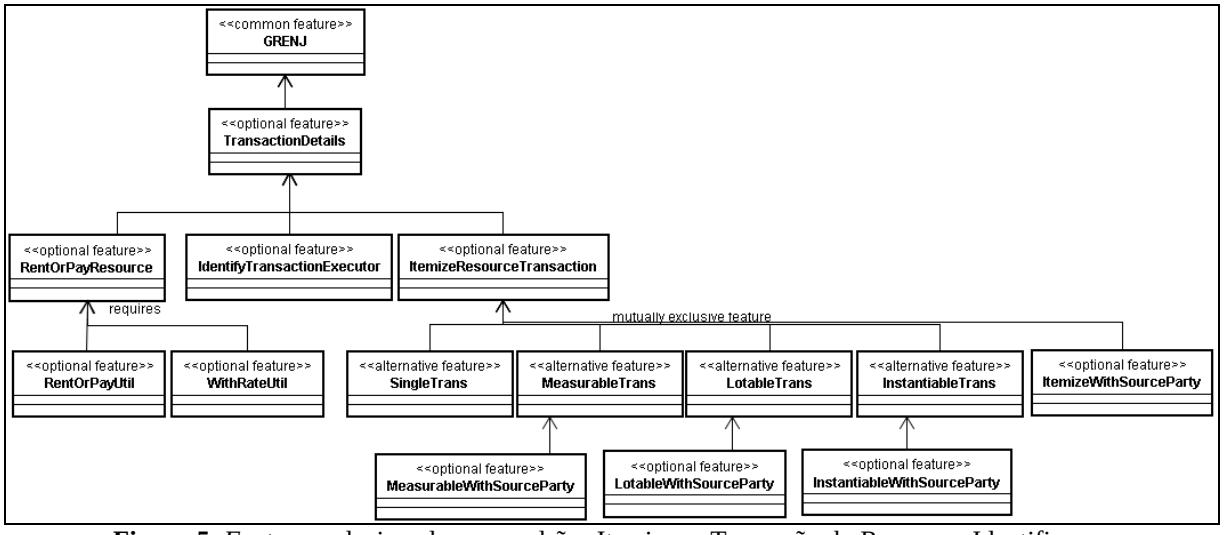

Figura 5. *Features* relacionadas aos padrões Itemizar a Transação do Recurso e Identificar o Executor da Transação da GRN.

Na Figura 6 são apresentadas as *features* relacionadas ao padrão Pagar pela Transação do Recurso, que pertence ao grupo Detalhes das Transações do Negócio da GRN.

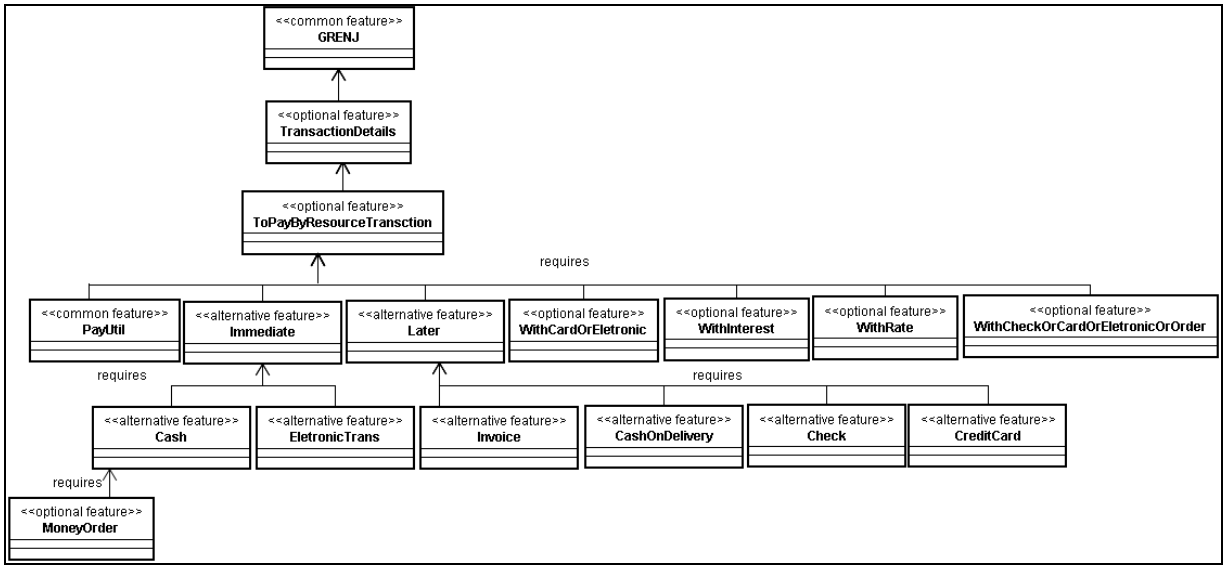

Figura 6. *Features* relacionadas ao padrão Pagar pela Transação do Recurso da GRN.

REFERÊNCIAS

GOMAA, H. **Designing software product lines with UML – from use cases to pattern-based software architectures**. 1 ed. Addison Wesley Longman Publishing Co, 2004. 736 p.

Apêndice B

TABELA DE MAPEAMENTO DOS PADRÕES E VARIANTES DA GRN PARA UNIDADES DO GRENJ-OA

Padrão/ Variante	Pacote (s)	Classe (s)
Core da Linha de Produtos	grenj.model; grenj.persistence; grenj.util	QualifiableObject, StaticObject, ConnectionManager, PersistentObject, RowNotFoundException, SQLCommand, Index, Money, Period, Status
Identificar Recurso	grenj.model.identifyresource;	Resource, IdentifyResourceConcernContainerLoaderAspect, IIdentifyResource,
Identificar Recurso variante recurso com tipo simples	grenj.model.identifyresource.withtype.simple	SimpleType
Identificar Recurso variante recurso com titipo aninhado	grenj.model.identifyresource.withtype.nested	NestedType
Quantificar Recurso	grenj.model.quantifyresource	QuantificationStrategy, ResourceWithQuantificationContainerLoaderAspect, QuantificationStrategyWithResourceContainerLoaderAspect
Quantificar Recurso Único	grenj.model.quantifyresource.singleresource	SingleResource
Quantificar Recurso Mensurável	grenj.model.quantifyresource.measurableresource	MeasurableResource, MeasureUnity, MeasurableQuantStrategyContainerLoaderAspect
Quantificar Recurso Instanciável	grenj.model.quantifyresource.instantiableresource	InstantiableResource, ResourceInstance, InstantiableQuantStrategyContainerLoaderAspect, InstantiableWithResourceAspect
Armazenar Recurso	grenj.model.withstore	ResourceWithStoreContainerLoaderAspect
Transações do Recurso	grenj.model.businesstransaction; grenj.model.businesstransaction.util	BusinessResourceTransaction, TransactionQuantificationStrategy, ResourceWithTransactionQuantificationStrategyContainerLoaderAspect, TransactionStatus, BusinessResourceTransactionWithResourceContainerLoaderAspect
Transação Única	grenj.model.businesstransaction.single	SingleResTransaction

Transação Instanciável	grenj.model.businesstransaction.instantiable	InstantiableResTransaction, InstantiableTransactionContainerLoaderAspect
Transação Mensurável	grenj.model.businesstransaction.measurable	MeasurableResTransaction, MeasurableTransactionContainerLoaderAspect,
Transação Mensurável ou em Lotes	grenj.model.businesstransaction.measurableorlotable	MeasurableOrLotableTransactionContainerLoaderAspect
Transação em Lotes	grenj.model.businesstransaction.lotable	LotableResTransaction, LotableTransactionContainerLoaderAspect
Transação com Destination Party	grenj.model.businesstransaction.withdestinationparty	DestinationParty
Transação com Source Party	grenj.model.businesstransaction.withsourceparty	SourceParty, BusinessResourceTransactionWithSourcePartyContainerLoaderAspect
Transação com Locação ou Reserva	grenj.model.businesstransaction.withrentalorreservation	BusinessResourceTransactionWithRentalOrReservationContainerLoaderAspect
Locar Recurso	grenj.model.businesstransaction.rentresource; grenj.model.businesstransaction.rentorpayresource.util	ResourceRental, ResourceRentalContainerLoaderAspect, AbstractCalculator
Locar Recurso Instanciável	grenj.model.businesstransaction.rentresource.instantiable	InstantableResourceWithRentalContainerLoaderAspect
Locar Recurso Mensurável	grenj.model.businesstransaction.rentresource.measurable	MeasurableResourceWithRentalContainerLoaderAspect
Locar Recurso Único	grenj.model.businesstransaction.rentresource.single	SingleResourceWithRentalContainerLoaderAspect
Locar Recurso Itemizado	grenj.model.businesstransaction.rentresource.itemized	ResourceRentalItemizedContainerLoaderAspect
Locar Recurso com Multa	grenj.model.businesstransaction.rentresource.withrate; grenj.model.businesstransaction.rentorpayresource.withrate.util	ResourceRentalWithFineRateContainerLoaderAspect, FineRate, NumberRangeCalculator
Locar Recurso com Reserva	grenj.model.businesstransaction.rentresource.withreservation	ResourceReservation, BusinessResourceTransactionWithReservationContainerLoaderAspect, RentalWithReservationContainerLoaderAspect
Locar Recurso com Reserva Instanciável	grenj.model.businesstransaction.rentresource.withreservation.instantiable	ReservationInstantiableAspect
Locar Recurso com Reserva em Lotes	grenj.model.businesstransaction.rentresource.withreservation.lotable	ReservationLotableAspect
Locar Recurso com Reserva Mensurável	grenj.model.businesstransaction.rentresource.withreservation.measurable	ReservationMeasurableAspect
Locar Recurso com Reserva Itemizada	grenj.model.businesstransaction.rentresource.withreservation.itemized	RentalAndReservationItemizedContainerLoaderAspect
Locar Recurso com Reserva associada com Source Party	grenj.model.businesstransaction.-rentresource.withreservation.withsourceparty	ResourceReservationWithSourcePartyContainerLoaderAspect
Locar Recurso com Reserva associada com Source Party Itemizado	grenj.model.businesstransaction.rentresource.withreservation.withsourceparty.itemized	ResourceReservationWithSourcePartyItemizedContainerLoaderAspect
Locar Recurso com Venda	grenj.model.businesstransaction.rentresource.withsale	RentResouceWithSaleContainerLoaderAspect

Comercializar Recurso	grenj.model.businesstransaction.resourcecommercialization	BasicNegotiation, ResourceTrade, TradeResourceContainerLoaderAspect
Comercializar Recurso: Compra	grenj.model.businesstransaction.resourcecommercialization.buy	BasicPurchase
Comercializar Recurso: Compra Mensurável	grenj.model.businesstransaction.resourcecommercialization.buy.measurable	BasicPurchaseMeasurableContainerLoaderAspect
Comercializar Recurso: Compra Itemizada	grenj.model.businesstransaction.resourcecommercialization.buy.itemized	BasicPurchaseItemizedContainerLoaderAspect
Comercializar Recurso: Compra com Entrega	grenj.model.businesstransaction.resourcecommercialization.buy.withdelivery	PurchaseDelivery
Comercializar Recurso: Compra com Entrega Mensurável	grenj.model.businesstransaction.resourcecommercialization.buy.withdelivery.measurable	PurchaseDeliveryMeasurableAspect
Comercializar Recurso: Compra com Entrega Itemizada	grenj.model.businesstransaction.resourcecommercialization.buy.withdelivery.itemized	PurchaseDeliveryItemizedContainerLoaderAspect
Comercializar Recurso Único	grenj.model.businesstransaction.resourcecommercialization.single	SingleResourceWithCommercializationContainerLoaderAspect
Comercializar Recurso Instanciável	grenj.model.businesstransaction.resourcecommercialization.instantiable	InstantiableCommercializationContainerLoaderAspect
Comercializar Recurso Mensurável	grenj.model.businesstransaction.resourcecommercialization.measurable	MeasurableCommercializationContainerLoaderAspect
Comercializar Recurso Mensurável Itemizado	grenj.model.businesstransaction.resourcecommercialization.measurable.itemized	ResourceTradeAndBasicNegotiationMeasurableItemizedContainerLoaderAspect
Comercializar Recurso com Entrega	grenj.model.businesstransaction.resourcecommercialization.withdelivery	BasicDelivery, ResourceTradeWithDeliveryContainerLoaderAspect
Comercializar Recurso com Entrega Mensurável	grenj.model.businesstransaction.resourcecommercialization.withdelivery.measurable	BasicDeliveryMeasurableAspect
Comercializar Recurso com Entrega em Lotes	grenj.model.businesstransaction.resourcecommercialization.withdelivery.lotable	BasicDeliveryLotableAspect
Comercializar Recurso com Entrega Instanciável	grenj.model.businesstransaction.resourcecommercialization.withdelivery.instantiable	BasicDeliveryInstantiableAspect
Comercializar Recurso com Entrega Itemizada	grenj.model.businesstransaction.resourcecommercialization.withdelivery.itemized	DeliveryItemizedContainerLoaderAspect
Comercializar Recurso: Venda	grenj.model.businesstransaction.resourcecommercialization.sale	BasicSale

Comercializar Recurso: Venda Mensurável	grenj.model.businesstransaction.resourcecommercialization.sale.measurable	BasicSaleMeasurableContainerLoaderAspect
Comercializar Recurso: Venda Mensurável Itemizada	grenj.model.businesstransaction.resourcecommercialization.sale.measurable.itemized	BasicSaleMeasurableItemizedContainerLoaderAspect
Comercializar Recurso: Venda com Entrega	grenj.model.businesstransaction.resourcecommercialization.sale.withdelivery	SaleDelivery
Comercializar Recurso: Venda com Entrega Mensurável	grenj.model.businesstransaction.resourcecommercialization.sale.withdelivery.measurable	SaleDeliveryMeasurableAspect
Comercializar Recurso: Venda com Entrega Mensurável Itemizada	grenj.model.businesstransaction.resourcecommercialization.sale.withdelivery.measurable.itemized	SaleDeliveryMeasurableItemizedContainerLoaderAspect
Cotar Recurso	grenj.model.businesstransaction.resourcecomercialization.withquotation	BusinessResourceQuotation, QuotationItem, QuotationItemWithResourceContainerLoaderAspect, ResourceTradeWithQuotationContainerLoaderAspect
Cotar Recurso com Executor	grenj.model.businesstransaction.resourcecomercialization.withquotation.withexecutor	BusinessResourceQuotationWithExecutorContainerLoaderAspect
Cotar Recurso com Source Party	grenj.model.businesstransaction.resourcecomercialization.withquotation.withsourceparty	BusinessResourceQuotationWithSourcePartyContainerLoaderAspect
Manter Recurso	grenj.model.businesstransaction.maintainresource	BasicMaintenance, ResourceMaintenance
Manter Recurso Itemizado	grenj.model.businesstransaction.maintainresource.itemized	MaintainanceItemizedContainerLoaderAspect
Cotar a Manutenção do Recurso	grenj.model.businesstransaction.maintainresource.withquotation	MaintenanceQuotation, MaintenanceWithQuotationContainerLoaderAspect
Cotar a Manutenção do Recurso Itemizada	grenj.model.businesstransaction.maintainresource.withquotation.itemized	MaintenanceWithQuotationItemizedContainerLoaderAspect
Identificar Tarefas de Manutenção	grenj.model.businesstransaction.maintainresource.withtasks	MaintenanceTask, MaintenanceTaskWithBasicMaintenanceContainerLoaderAspect, MaintenanceWithTasksContainerLoaderAspect
Identificar Tarefas de Manutenção Itemizado	grenj.model.businesstransaction.maintainresource.withtasks.itemized	MaintenanceWithTaskItemizedContainerLoaderAspect
Identificar Tarefas de Manutenção com Executor da Transação	grenj.model.businesstransaction.maintainresource.withtasks.withexecutor	MaintenanceTaskWithExecutorContainerLoaderAspect
Identificar Peças de Manutenção	grenj.model.businesstransaction.maintainresource.withparts	MaintenancePart, Part, MaintenancePartWithBusinessResourceTransactionContainerLoaderAspect, MaintenanceWithPartsContainerLoaderAspect
Identificar Peças de Manutenção Itemizado	grenj.model.businesstransaction.maintainresource.withparts.itemized	MaintenanceWithPartsItemizedContainerLoaderAspect
Identificar o Executor da Transação	grenj.model.businesstransaction.transactiondetails.identifytransactionexecutor	TransactionExecutor, BusinessResourceTransactionWithExecutorContainerLoaderAspect
Itemizar Transação do Recurso	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction	TransactionItem, ItemQuantificationStrategy, BusinessResourceTransactionWithItemContainerLoaderAspect, TransactionItemWithResourceContai-

		nerLoaderAspect
Itemizar a Transação do Recurso com Source Party	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction.withsourceparty	BusinessResourceTransactionItemizedWithSourcePartyContainerLoaderAspect
Itemizar a Transação do Recurso Instanciável	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction.instantiable	InstResTransItem, TransactionItemInstantiableContainerLoaderAspect
Itemizar a Transação do Recurso Instanciável com Source Party	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction.instantiable.withsourceparty	BusinessResourceTransactionItemizedInstantiableWithSourcePartyAspect
Itemizar a Transação do Recurso em Lotes	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction.lotable	LotResTransItem, TransactionItemLotableContainerLoaderAspect
Itemizar a Transação do Recurso em Lotes com Source Party	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction.lotable.withsourceparty	BusinessResourceTransactionItemizedLotableWithSourcePartyAspect
Itemizar a Transação do Recurso Mensurável	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction.measurable	MeasResTransItem, TransactionItemMeasurableContainerLoaderAspect
Itemizar a Transação do Recurso Mensurável com Source Party	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction.measurable.withsourceparty	BusinessResourceTransactionItemizedMeasurableWithSourcePartyAspect
Itemizar a Transação do Recurso Único	grenj.model.businesstransaction.transactiondetails.itemizeresourcetransaction.single	SingleResTransItem
Pagar pela Transação do Recurso	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction; grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.util	AbstractCalculator, Payment, PaymentStrategy, BusinessResourceTransactionWithPayment, CreditCardValidator, PaymentStatus
Pagar pela Transação do Recurso: pagamento imediato	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.immediate	ImmediateReceiving
Pagar pela Transação do Recurso: pagamento imediato: dinheiro	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.immediate.withcash	Cash, PaymentWithCashContainerLoaderAspect
Pagar pela Transação do Recurso: pagamento imediato: transferência eletrônica	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.immediate.withelectronictrans	ElectronicTransfer
Pagar pela Transação do Recurso: pagamento imediato: ordem de pagamento	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.immediate.withmoneyorder	MoneyOrder
Pagar pela Transação do Recurso: pagamento posterior	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.later	LaterReceiving

Pagar pela Transação do Recurso: pagamento posterior: pagamento na entrega	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.later.withcashondelivery	CashOnDelivery
Pagar pela Transação do Recurso: pagamento posterior: cheque	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.later.withcheck	Check, PaymentWithCheckContainerLoaderAspect
Pagar pela Transação do Recurso: pagamento posterior: cartão de crédito	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.later.withcreditcard	CreditCard, PaymentWithCreditCardContainerLoaderAspect
Pagar pela Transação do Recurso: pagamento posterior: fatura	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.later.withinvoice	Invoice, PaymentWithInvoiceContainerLoaderAspect
Pagar pela Transação do Recurso: pagamento com cartão de crédito ou transferência eletronica	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.later.withcardorelectronic	PaymentWithCardOrElectronicContainerLoaderAspect
Pagar pela Transação do Recurso: pagamento com cheque, cartão de crédito, transferência eletrônica ou com ordem de pagamento	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.later.withcheckorcardorelectronicororder	PaymentWithCardCheckElectronicOrderContainerLoaderAspect
Pagar pela Transação do Recurso: com juros	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.withinterest	ExactNumberCalculator, InterestedRate, PaymentWithInterestContainerLoaderAspect
Pagar pela Transação do Recurso: com multa	grenj.model.businesstransaction.transactiondetails.paybyresourcetransaction.withrate	PaymentWithRateContainerLoaderAspect

Apêndice C

QUESTIONÁRIO DE PERFIL

Nome:_____

Data: _ _/ _ _/ _ _ _ _

Curso: ( ) Ciência da Computação

( ) Bacharelado em Informática

( ) Processamento de Dados

( ) Outros: _____

Conhece linguagens orientadas a objetos?_____ Quais? _____

_____

Conhecimento dos conceitos de Orientação a Objetos: Alto ( ) Médio ( ) Baixo ( )

Conhecimento dos conceitos de Orientação a Aspectos: Alto ( ) Médio ( ) Baixo ( )

Nível de conhecimento em relação à linguagem Java:

( ) Básico – aproximadamente 1 ano.

( ) Intermediário – mais de 2 anos.

( ) Avançado – 5 anos ou mais.

Nível de conhecimento relação à linguagem AspectJ:

( ) Básico – menos de 1 ano.

( ) Intermediário – mais de 1 ano.

( ) Avançado – 2 anos ou mais.

Em qual das categorias abaixo você se encaixa como desenvolvedor?

( ) sistemas triviais desenvolvidos durante o curso na universidade.

( ) sistemas de médio porte; com aproximadamente 7-20 mil linhas de código.

( ) sistemas de grande porte; com aproximadamente 20-50 mil linhas de código.

Conhece padrões de projeto?

( ) Não.

( ) Superficialmente.

( ) Conheço vários padrões de projeto.

( ) Conheço vários padrões em vários níveis de abstração.

- Você costuma utilizar padrões de projeto nos seus projetos acadêmicos ou pessoais?
  ( ) Não ( ) Usualmente ( ) Sempre que possível

Classifique seu conhecimento em relação aos conceitos: frameworks de software orientados a objetos, "princípio de Hollywood", *hot spots, frozen spots, hook methods* e *template methods*:

( ) Nenhum.

( ) Pouco.

( ) Intermediário.

( ) Avançado.

Conhece a técnica (prática) denominada refatoração?

( ) Não ( ) Sim

Emprega essa técnica em projetos acadêmicos ou pessoais?

( ) Não ( ) Usualmente ( ) Sempre

Realiza somente as refatorações automatizadas pelo ambiente de desenvolvimento integrado (IDE) ou também realiza refatorações mais complexas e não automatizadas?

( ) Não costumo refatorar o código.

( ) Aplico somente as refatorações automatizadas pelo IDE.

( ) Aplico tanto as refatorações automatizadas quanto as não automatizadas.

( ) Procuro sempre refatorar utilizando refatorações automatizadas e não automatizadas e, além disso, quando possível tento refatorar a fim de introduzir padrões de projeto.

Apêndice D

QUESTIONÁRIOS

FASE 1

Nome:_____

Data: / /

Perguntas para o Engenheiro do *Framework*:

1 – Cite quatro unidades (classes/interfaces/aspectos) que constituem o núcleo do *framework* GRENJ: **(Max 3 min)**

Hora Início (hh:mm:ss):

Hora Fim (hh:mm:ss):

R:

2 – Quais unidades do GRENJ estão relacionadas às variantes “Recurso com Tipo Simples” e “Recurso com Tipo Aninhado” do padrão Identificar Recurso? **(Max 2 min)**

Hora Início (hh:mm:ss):

Hora Fim (hh:mm:ss):

R:

Perguntas para o Engenheiro da Aplicação:

1 – Quando se utiliza a variante Recurso Mensurável (*Measurable Resource*) do padrão Quantificar Recurso, quais métodos abstratos da classe *Resource* necessitam de ser concretizados? **(Max 4 min)**

Hora Início (hh:mm:ss):

Hora Fim (hh:mm:ss):

R:

2 – Dado o requisito: Um sistema de aluguel de roupas necessita de armazenar informações relacionadas às roupas disponíveis para aluguel e dos aluguéis efetuados. Cada tipo de roupa (terno, vestido, calça) possui várias peças. Por exemplo, há vários ternos disponíveis para aluguel. Quais métodos abstratos da classe *Resource* do *framework* necessitam de implementações concretas para construir a aplicação que satisfaça os requisitos do enunciado. (**MAX 4**)

Hora Início (hh:mm:ss):

Hora Fim (hh:mm:ss):

R:

FASE 2

Nome:_____

Data: / /

Perguntas para o Engenheiro do *Framework*:

1 – Quais as unidades (classe/interface/aspecto) do GRENJ que respectivamente representam as variantes Recurso Único, Recurso Mensurável e Recurso Instanciável do padrão Quantificar Recurso? **(Max 4 min)**

Hora Início (hh:mm:ss):

Hora Fim (hh:mm:ss):

R:

2 – Quais as unidades (classe/aspecto/interface) no GRENJ que contêm código relacionado ao padrão Reservar Recurso? **(Max 5)**

Hora Início (hh:mm:ss):

Hora Fim (hh:mm:ss):

R:

Perguntas para o Engenheiro da Aplicação:

1 – Cite dois métodos da classe *TransactionItem* do GRENJ que necessitam de ser implementados por classes que a estendem, ou seja, são considerados obrigatórios? **(Max 3 min)**
Hora Início (hh:mm:ss):
Hora Fim (hh:mm:ss):
R:

2 – Dado o requisito: Um sistema que necessita de armazenar informações relacionadas ao estoque de matérias primas utilizadas em uma farmácia de manipulação para a elaboração de medicamentos. Cada matéria prima está associada a uma unidade de medida (exemplo: grama, mililitro). Quais classes do *framework* GRENJ necessitam de ser estendidas para construir a aplicação que satisfaça os requisitos do enunciado? **(Max 4 min)**
Hora Início (hh:mm:ss):
Hora Fim (hh:mm:ss):
R:

Apêndice E

FORMULÁRIO DE CONSENTIMENTO E TABELAS PARA REGISTRO DE TEMPO

Modelo de Formulário de Consentimento

Título do projeto e Propósito

Experimento Comparativo entre as Versões Orientadas a Objetos e Orientadas a Aspectos do *Framework* GRENJ

O objetivo do experimento é comparar o tempo gasto para instanciar aplicações e realizar manutenções com as versões OO e OA do framework GRENJ, que é um *framework* de aplicação baseado na linguagem de padrões GRN. A partir da comparação procura-se verificar as vantagens e desvantagens do uso da versão OA do GRENJ.

Declaração de idade

Eu declaro que sou maior de 18 anos de idade e que desejo participar do experimento conduzido por André Luiz de Oliveira, estudante de mestrado do PPGCC-UFSCar, no contexto da disciplina de Tópicos em Engenharia de Software.

Procedimentos

O experimento envolve quatro etapas. Na primeira etapa será conduzido um treinamento que aborda a linguagem de padrões GRN e o uso das versões OO e OA do *framework* GRENJ. Na segunda etapa serão distribuídos dois questionários aos alunos com perguntas relacionadas à arquitetura de ambos, com finalidade de verificar o grau facilidade para respondê-la analisando cada uma das versões do GRENJ. Na terceira etapa serão

efetuadas duas manutenções, uma na versão OO e outra na versão do GRENJ. Na quarta etapa será feita a instanciação de duas aplicações, uma com a versão OO e outra com a versão OA do *framework*.

Confidência

Todas as informações coletadas no experimento são confidências, e meu nome não será identificado em tempo algum.

Benefícios, liberdade para retirar-se.

Estou consciente que não terei remuneração pela participação no experimento, mas que o estudo ajudará a aprender mais sobre o uso de *frameworks* de aplicação OO e OA. Eu entendo que sou livre para perguntar ou para retirar minha participação em qualquer tempo sem penalidade, e que eu terei acesso aos resultados principais do experimento.

Responsáveis

André L. de Oliveira	Prof. Dr. Valter V. de Camargo	Prof. Dra Rosângela A. D. Penteado
Univ. Federal de São Carlos	Univ. Federal de São Carlos	Univ. Federal de São Carlos
São Carlos-SP	São Carlos-SP	São Carlos-SP

_____

Nome do participante

_____

Assinatura do participante Data

Exemplos de Tabelas para a Marcação do Tempo de Duração de Cada Manutenção Realizada no Experimento

Fase 1 - Tabela 1 – Tempo para realizar a manutenção de adicionar a estratégia de quantificação Instantiable Resource na versão GRENJ-OA.

Hora de Início (hh:mm:ss)	Hora de Conclusão (hh:mm:ss)
Tempo Total:	

Fase 2 - Tabela 2 – Tempo para realizar a manutenção de adicionar o padrão Itemizar a Transação do Recurso na versão GRENJ-OO.

Hora de Início (hh:mm:ss)	Hora de Conclusão (hh:mm:ss)
Tempo Total:	

Exemplos de Tabelas para a Marcação do Tempo de Duração de Cada Instanciação Realizada no Experimento

Fase 1 – Tabela 1 - Tempo para configurar o *framework* para instanciar a aplicação BibliotecaOA na versão GRENJ-OA.

Tempo Total:	

Fase 1 – Tabela 2 - Tempo para instanciar a aplicação BibliotecaOA na versão GRENJ-OA.

Hora de Início (hh:mm:ss)	Hora de Conclusão (hh:mm:ss)	Caso de teste executou com sucesso? (s/n)
Tempo Total:		

Fase 2 – Tabela 3 - Tempo para configurar o *framework* para instanciar a aplicação HotelOO na versão GRENJ-OO.

Tempo Total:	

Fase 2 - Tabela 4 – Tempo para instanciar a aplicação HotelOO na versão GRENJ-OO.

Hora de Início (hh:mm:ss)	Hora de Conclusão (hh:mm:ss)	Caso de teste executou com sucesso? (s/n)
Tempo Total:		

Apêndice F

MODELOS DE CLASSES, MANUAIS DE INSTANCIAÇÃO E ROTEIRO DE MANUTENÇÃO

Nas Figuras 1 e 2 são apresentados os modelos de classes das aplicações de Biblioteca e Hotel utilizados no experimento descrito no Capítulo 7.

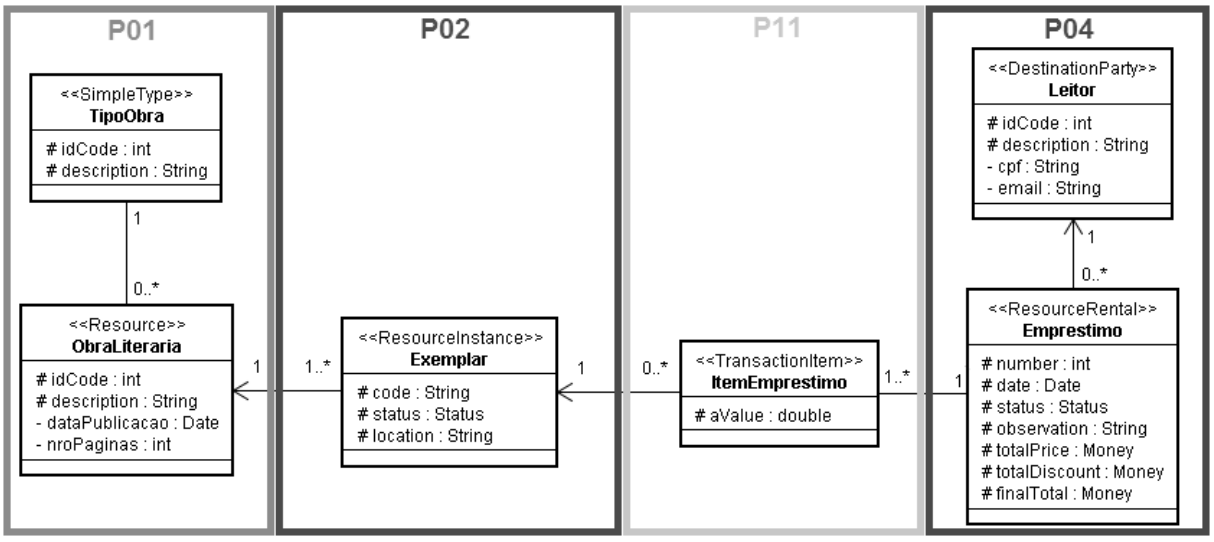

Figura 1. Modelo de classes do sistema para uma Biblioteca.

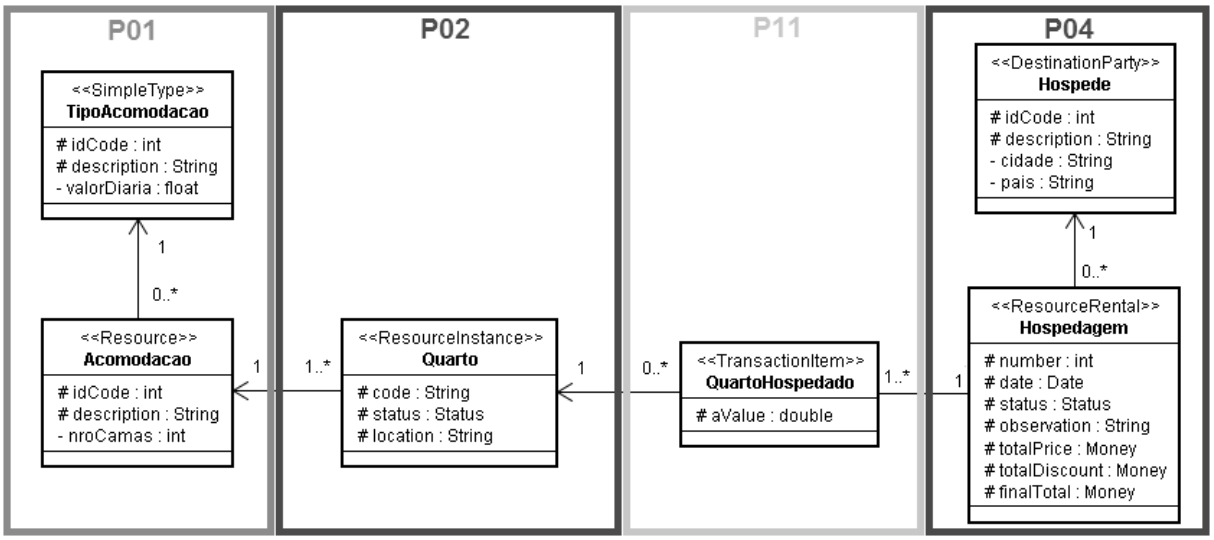

Figura 2. Modelo de classes do sistema para um Hotel.

Manual de Instanciação da versão GRENJ-OO

Relação entre a GRN e as classes da camada de negócios do *framework* GRENJ- OO

Padrão GRN	Variante	Classe no Padrão	Classe no GRENJ	Tabela
1 – Identificar o Recurso	-	Recurso	Resource	Sim
	Tipos Simples	Tipo de Recurso	SimpleType	Sim
	Tipos Aninhados	Tipo de Recurso	NestedType	Sim
2 – Quantificar o Recurso	Recurso Simples	-	-	Não
	Recurso Instanciável	Instância do Recurso	ResourceInstance	Sim
	Recurso Mensurável ou Recurso em Lotes	Unidade de Medida	MeasureUnity	Sim
	Recurso em Lotes	Lote do Recurso	MeasureUnity	Sim
4 a 10 – Transações	-	Origem	SourceParty	Sim
		Destino	DestinationParty	Sim
4 – Locar o Recurso	-	Locação do recurso	ResourceRental	Sim
		Taxa de Multa	FineRate	Sim
5 – Reservar o Recurso	-	Reserva do Recurso	ResourceReservation	Sim
6 – Comercializar o Recurso	Compra	Comercio do Recurso	BasicPurchase	Sim
	Venda		BasicSale	Sim
7 – Cotar o Recurso	-	Cotação do Recurso	BusinessResourceQuotation	Sim
		Item da Cotação	QuotationItem	Sim
8 – Conferir a Entrega do Recurso	Entrega de Compra	Entrega do Recurso	PurchaseDelivery	Sim
	Entrega de Venda		SaleDelivery	Sim
9 – Manter o Recurso	-	Manutenção do Recurso	ResourceMaintenance	Sim
10 – Cotar a Manutenção	-	Cotação da Manutenção	MaintenanceQuotation	Sim
11 – Itemizar a Transação do Recurso	-	Item da Transação	TransactionItem	Sim
12 – Pagar pela Transação do Recurso	-	Pagamento	Payment	Sim
		Taxa de Multa	FineRate	Sim
		Taxa de Juros	InsterestRate	Sim
		Dinheiro	Cash	Sim
		Cartão de Crédito	CreditCard	Sim
		Cheque	Check	Sim
		Fatura	Invoice	Sim
		Ordem de Pagamento	MoneyOrder	Sim
		Pagamento na Entrega	CashOnDelivery	Sim
		Transferência Eletrônica	ElectronicTransfer	Sim
13 – Identificar o Executor da Transação	-	Executor da Transação	TransactionExecutor	Sim
14 – Identificar as Tarefas da Manutenção	-	Tarefa da Manutenção	MaintenanceTask	Sim
		Executor da Manutenção	TransactionExecutor	Sim
15 – Identificar as Peças da Manutenção	-	Peça	Part	Sim
		Peça Usada na Manutenção	MaintenancePart	Sim

Métodos Comuns a todas as classes

Nome	Entrada	Saída	Descrição

Construtor sem argumentos (use o próprio nome da classe)	-	-	Na aplicação, o código do construtor sem argumentos deve chamar o construtor de sua super classe:
			`super();`
			Além disso, se a classe possuir atributos próprios é necessário inicializá-los com valores default. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria:
			`ano = 0;`

Construtor com argumentos (use o próprio nome da classe)	ResultSet result, Index anIndex	-	Na aplicação, o código do construtor com argumentos deve chamar o construtor de sua super classe:
			`super(result, anIndex);`
			Além disso, se a classe possuir atributos próprios é necessário inicializá-los com os valores provenientes do banco de dados dentro de um bloco de tratamento de exceção (*try catch*). Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria:
			`try {`
			`  ano = result.getInt(anIndex.get`
			`  Index() );`
			`  anIndex.incrementIndexByOne();`
			`}`
			`catch ( SQLException e ) {`
			`  e.printStackTrace();`
			`}`

getters	-	O tipo do atributo obtido	Retorna o valor do atributo relacionado ao método. Deve haver um método desse tipo para cada atributo próprio da classe. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria:
			`public int getAno() {`
			`  return ano;`
			`}`

setters	O tipo do atributo alterado	void	Altera o valor do atributo relacionado ao método. Deve haver um método desse tipo para cada atributo próprio da classe. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria:
			`public void setAno( int newAno ) {`
			`  ano = newAno;`
			`}`

insertionFieldClause	-	String	Define os campos que são utilizados nas operações de inserção no banco de dados. Por exemplo, para um atributo do tipo inteiro chamado *ano*, o código seria: `return super.insertionFieldClause() + ", ano";`

insertionValueClause	-	String	Informa valores a serem gravados nas operações de inserção no banco de dados. Por exemplo, para um atributo do tipo inteiro chamado *ano*, o código seria:
			`return super.insertionValueClause()`
			`  + ", " + getAno();`

updateSetClause	-	String	Informa os campos e valores a serem alterados nas operações de atualização do banco de dados. Por exemplo, para um atributo do tipo inteiro chamado *ano*, o código seria:
			`return super.updateClause()`
			`  + ", ano = " + this.getAno();`

Classe Resource

Atributos

Nome	Descrição
idCode	Código identificador (chave).
description	Descrição ou nome do recurso.

Métodos

Nome	Entrada	Saída	Descrição
typeClasses	-	Class[]	Indica as classes de tipo de recurso. Por exemplo, considere que o recurso está associado a duas subclasses de *SimpleType*: *TipoDeProduto* e *Fabricante*. Então o código seria: `return new Class[] { TipoDeProduto.class, Fabricante.class };`
typeFieldsInitialize	-	String[]	Indica os nomes dos campos dos tipos de recurso na tabela do recurso do banco de dados. Por exemplo, considere que o recurso está associado a duas subclasses de *SimpleType*: *TipoDeProduto* e *Fabricante*. Então o código seria: `return new String[] { "tipoDeProduto", "fabricante" };`
getQuantificationStrategyInstance	-	QuantificationStrategy	Indica a estratégia de quantificação do recurso (Padrão 2). Seu valor pode ser uma instância de uma subclasse de QuantificationStrategy: • *SingleResource* – para um recurso único; • *InstantiableResource* – para um recurso que possui cópias; • *MeasurableResource* – para um recurso mensurável ou em lotes; Por exemplo, considere que o recurso é instanciável, então o código seria: `return new InstantiableResource();`
getMeasureUnityClass	-	Class< ? extends MeasureUnity>	Indica uma classe que representa uma unidade de medida. Se o recurso não é mensurável, então esse método retorna nulo. Exemplo: `return null;`
getResourceInstanceClass	-	Class< ? extends ResourceInstance>	Indica uma classe que representa as instâncias do recurso. Por exemplo, considere a classe DVD que representa as instâncias do recurso. Então o código seria: `return DVD.class;`
getResourceRentalClass	-	Class< ? extends ResourceRental>	Indica uma classe que representa uma transação de aluguel. Por exemplo, considere que a classe *Locacao* representa a transação de aluguel do recurso. Então o código seria: `return Locacao.class;`

Classe SimpleType

Atributos

Nome	Tipo	Descrição
idCode	int	Código identificador (chave).
description	String	Descrição ou nome do recurso.

Métodos

Nome	Entrada	Saída	Descrição
typeClasses	-	Class[]	Indica as classes de tipo de recurso. Em *SimpleType* esse método sempre retorna um array vazio, como indica o código: `return new Class[] { };`
typeFieldsInitialize	-	String[]	Indica os nomes dos campos dos tipos de recurso na tabela do recurso do banco de dados. Em *SimpleType* esse método sempre retorna um array vazio, como indica o código: `return new String[] { };`

Classe ResourceInstance

Atributos

Nome	Tipo	Descrição
resourceIdCode	int	Armazena o número de identificação do recurso.
code	String	Armazena o código de identificação da instância.
location	String	Indica o local onde a instância pode ser encontrada.
status	Status	Indica o status da instância: disponível ou não disponível.

Métodos

Nome	Entrada	Saída	Descrição

getResource	-	Classe que representa o recurso	Retorna o recurso que é representado pela instância. Por exemplo, considere uma classe *DVD* que representa as instâncias do recurso *Filme*, então o código seria:
			`try {`
			`  return(Filme)`
			`  PersistentObject.load(`
			`  getResourceIdCode(), Filme.class`
			`  );`
			`}`
			`catch (Exception e) {`
			`  e.printStackTrace();`
			`  return null;`
			`}`

Classe ResourceRental

Atributos

Nome	Tipo	Descrição
number	int	Indica um número de identificação para cada objeto.
date	java.sql.Date	Armazena a data de efetuação da transação.
observation	String	Armazena informações relacionadas com a transação.
status	TransactionStatus	Indica o status da transação: aberta ou fechada.
totalPrice	Money	Armazena o valor inicial da transação.
totalDiscount	Money	Armazena um valor de desconto.
finalTotal	Money	Armazena um valor calculado com base no preço inicial da transação, no desconto, no valor de seus itens e outros valores que podem incidir sobre o valor final.
destinationParty	DestinationParty	Indica o destino do(s) recurso(s) da transação.
items	List<TransactionItem>	Indica os itens da transação, se ela for itemizada.

Métodos

Nome	Entrada	Saída	Descrição
getTransactionQuantificationStrategyInstance	-	TransactionQuantificationStrategy	Indica para a transação qual estratégia de quantificação foi aplicada sobre o recurso. A estratégia utilizada é indicada por uma das classes abaixo: • *SingleResTransaction*, se o recurso for único; • *InstantiableResTransaction*, se o recurso for instanciável; • *MeasurableResTransaction*, se o recurso for mensurável; • *LotableResTransaction*, se o recurso for tratado como lotes. Por exemplo, considere que o recurso é instanciável, então o código seria: `return new InstantiableResTransaction();`
getDestinationPartyClass	-	Class<? extends DestinationParty >	Indica a classe que representa o destino da transação. Por exemplo, se o destino for representado pela classe *Cliente*, então o código seria: `return Cliente.class;`
getExecutorClass	-	Class<? extends TransactionExecutor >	Indica a classe que representa o executor da transação. Por exemplo, se o executor da transação for representado pela classe *Funcionario*, então o código seria: `return Funcionario.class;`
getPaymentClass	-	Class<? extends Payment >	Indica a classe que representa o pagamento da transação. Por exemplo, se a aplicação não define uma classe para o pagamento, então o código seria: `return null;`
getResourceClass	-	Class<? extends Resource >	Indica a classe que representa o recurso. Por exemplo, se o recurso for representado pela classe *Filme*, então o código seria: `return Filme.class;`
getSourcePartyClass	-	Class<? extends SourceParty >	Indica a classe que representa a origem da transação. Por exemplo, se a aplicação não define uma classe para a origem, então o código seria: `return null;`
getFineRateClass	-	Class<? extends FineRate >	Indica a classe que representa a taxa de multa da transação de aluguel. Por exemplo, se a taxa de multa for representada pela classe *Multa*, então o código seria: `return Multa.class;`
getTransactionItemClass	-	Class<? extends TransactionItem>	Indica a classe que representa o item da transação. Por exemplo, se o item for representado pela classe *ItemLocacao*, então o código seria: `return ItemLocacao.class;`
getReservationClass	-	Class<? extends ResourceReservation>	Indica a classe que representa uma transação de reserva. Por exemplo, se o sistema não contemplar a reserva do recurso, então o código seria: `return null;`
isItemized	-	boolean	Indica se a transação do recurso é ou não itemizada. Por exemplo, se a transação for itemizada, então o código seria: `return true;`
hasExecutor	-	boolean	Indica se existe uma classe que representa o executor da transação. Por exemplo, se não existir uma classe que é o executor da transação, então o código seria: `return false;`
hasPayment	-	boolean	Indica se existe uma classe que representa a forma de pagamento da transação. Por exemplo, se não existir uma classe que é o pagamento da transação, então o código seria: `return false;`

hasSourceParty	-	boolean	Indica se existe uma classe que representa a origem da transação. Por exemplo, se não existir uma classe que é a origem da transação, então o código seria: `return false;`
hasAssociatedSale	-	boolean	Indica se na aplicação é possível associar uma venda à transação de aluguel. Por exemplo, se a aplicação não associa vendas com alugueis, então o código seria: `return false;`
hasFineRate	-	boolean	Indica se existe uma classe que representa a taxa de multa. Por exemplo, se não existir uma classe que é a taxa de multa, então o código seria: `return false;`
hasReservation	-	boolean	Indica se existe uma classe que representa transações de reserva. Por exemplo, se não existir uma classe que define as reservas, então o código seria: `return false;`
generateInstanceCode	-	boolean	Indica se a transação de aluguel deve gerar um código para a instância do recurso. Esse método retorna *true* somente se o recurso for instanciável e a transação não for itemizada, caso contrário, retorna *false*.
transactionPluralName	-	String	Indica o formado em plural do nome da transação. Por exemplo: `return "Locações";`

Classe DestinationParty

Atributos

Nome	Tipo	Descrição
idCode	int	Código identificador (chave).
description	String	Descrição ou nome do recurso.

Classe TransactionItem

Atributos

Nome	Tipo	Descrição
transaction	BusinessResourceTransaction	Armazena a transação a qual o item está associado.
resource	Resource	Armazena o recurso relacionado com o item da transação.
aValue	double	Armazena o valor do recurso no item da transação.

Métodos

Nome	Entrada	Saída	Descrição

Construtor com argumentos (use o próprio nome da classe)	ResultSet result, Index anIndex, BusinessResourceTransaction transaction	-	Na aplicação, o código do construtor com argumentos deve chamar o construtor de sua super classe:
			`super(result,anIndex, transaction);`
			Além disso, se a classe possuir atributos próprios, é necessário inicializá-los com os valores provenientes do banco de dados dentro de um bloco de tratamento de exceção (*try catch*). Por exemplo, para um atributo do tipo inteiro chamado *ano*, o código seria:
			`try { ano = result.getInt( anIndex. getIndex() ); anIndex.incrementIndexByOne(); } catch ( SQLException e ) { e.printStackTrace(); }`

getResourceClass	-	Class<? extends Resource >	Indica a classe que representa o recurso. Por exemplo, se o recurso for representado pela classe *Filme*, então o código seria: `return Filme.class;`
getTransactionClass	-	Class<? extends BusinessResourceTransaction >	Indica a classe que representa a transação. Por exemplo, se a transação for representada pela classe *Locacao*, então o código seria: `return Locacao.class;`
getItemQuantificationStrategyInstance	-	ItemQuantificationStrategy	Indica para o item da transação qual estratégia de quantificação foi aplicada sobre o recurso. A estratégia utilizada é indicada por uma das classes abaixo: • *SingleResTransItem*, se o recurso for único; • *InstResTransItem*, se o recurso for instanciável; • *MeasResTransItem*, se o recurso for mensurável; • *LotResTransItem*, se o recurso for em lotes. Por exemplo, considere que o recurso é instanciável, então o código seria: `return new InstResTransItem();`

Manual de Instanciação da versão GRENJ-OA

Relação entre a GRN e as classes da camada de negócios do framework GRENJ- OA

Padrão GRN	Variante	Classe no Padrão	Classe no GRENJ	Tabela
1 – Identificar o Recurso	-	Recurso	Resource	Sim
	Tipos Simples	Tipo de Recurso	SimpleType	Sim
	Tipos Aninhados	Tipo de Recurso	NestedType	Sim
2 – Quantificar o Recurso	Recurso Simples	-	-	Não
	Recurso Instanciável	Instância do Recurso	ResourceInstance	Sim
	Recurso Mensurável	Unidade de Medida	MeasureUnity	Sim
	Recurso em Lotes	Lote do Recurso	MeasureUnity	Sim
4 a 10 – Transações	-	Origem	SourceParty	Sim
		Destino	DestinationParty	Sim
4 – Locar o Recurso	-	Locação do recurso	ResourceRental	Sim
		Taxa de Multa	FineRate	Sim
5 – Reservar o Recurso	-	Reserva do Recurso	ResourceReservation	Sim
6 – Comercializar o Recurso	Compra	Comercio do Recurso	BasicPurchase	Sim
	Venda		BasicSale	Sim
7 – Cotar o Recurso	-	Cotação do Recurso	BusinessResourceQuotation	Sim
		Item da Cotação	QuotationItem	Sim
8 – Conferir a Entrega do Recurso	Entrega de Compra	Entrega do Recurso	PurchaseDelivery	Sim
	Entrega de Venda		SaleDelivery	Sim
9 – Manter o Recurso	-	Manutenção do Recurso	ResourceMaintenance	Sim
10 – Cotar a Manutenção	-	Cotação da Manutenção	MaintenanceQuotation	Sim
11 – Itemizar a Transação do	-	Item da Transação	TransactionItem	Sim

Recurso				
12 – Pagar pela Transação do Recurso	-	Pagamento	Payment	Sim
		Taxa de Multa	FineRate	Sim
		Taxa de Juros	InsterestRate	Sim
		Dinheiro	Cash	Sim
		Cartão de Crédito	CreditCard	Sim
		Cheque	Check	Sim
		Fatura	Invoice	Sim
		Ordem de Pagamento	MoneyOrder	Sim
		Pagamento na Entrega	CashOnDelivery	Sim
		Transferência Eletrônica	ElectronicTransfer	Sim
13 – Identificar o Executor da Transação	-	Executor da Transação	TransactionExecutor	Sim
14 – Identificar Tarefas da Manutenção	-	Tarefa da Manutenção	MaintenanceTask	Sim
		Executor da Manutenção	TransactionExecutor	Sim
15 – Identificar Peças da Manutenção	-	Peça	Part	Sim
		Peça Usada na Manutenção	MaintenancePart	Sim

Métodos Comuns a todas as classes

Nome	**Entrada**	**Saída**	**Descrição**

Construtor sem argumentos (use o próprio nome da classe)	-	-	Na aplicação, o código do construtor sem argumentos deve chamar o construtor de sua super classe:
			`super();`
			Além disso, se a classe possuir atributos próprios é necessário inicializá-los com valores default. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria:
			`ano = 0;`

Construtor COM argumentos (use o próprio nome da classe)	ResultSet result, Index anIndex	-	Na aplicação, o código do construtor com argumentos deve chamar o construtor de sua super classe:
			`super(result, anIndex);`
			Além disso, se a classe possuir atributos próprios é necessário inicializá-los com os valores provenientes do banco de dados dentro de um bloco de tratamento de exceção (*try catch*). Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria:
			`try {`
			`  ano = result.getInt(anIndex.`
			`  getIndex() );`
			`anIndex.incrementIndexByOne();`
			`}`
			`catch ( SQLException e ) {`
			`  e.printStackTrace();}`

getters	-	O tipo do atributo obtido	Retorna o valor do atributo relacionado ao método. Deve haver um método desse tipo para cada atributo próprio da classe. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria:
			`public int getAno() {`
			`  return ano; }`

setters	O tipo do atributo alterado	void	Altera o valor do atributo relacionado ao método. Deve haver um método desse tipo para cada atributo próprio da classe. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria: `public void setAno(int newAno) { ano = newAno; }`
insertionFieldClause	-	String	Define os campos que são utilizados nas operações de inserção no banco de dados. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria: `return super. insertionFieldClause()+",ano";`
insertionValueClause	-	String	Informa valores a serem gravados nas operações de inserção no banco de dados. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria: `return super.insertionValueClause() + ", " + getAno();`
updateSetClause	-	String	Informa os campos e valores a serem alterados nas operações de atualização do banco de dados. Por exemplo, para um atributo do tipo inteiro chamado `ano`, o código seria: `return super. updateClause() +", ano = "+ this.getAno();`

Classe Resource

Atributos

Nome	**Descrição**
idCode	Código identificador (chave).
Description	Descrição ou nome do recurso.

Métodos

Nome	**Entrada**	**Saída**	**Descrição**
typeClasses	-	Class[]	Indica as classes de tipo de recurso. Por exemplo, considere que o recurso está associado a duas subclasses de *SimpleType*: *TipoDeProduto* e *Fabricante*. Então o código seria: `return new Class[] { TipoDeProduto.class, Fabricante.class };`
typeFieldsInitialize	-	String[]	Indica os nomes dos campos dos tipos de recurso na tabela do recurso do banco de dados. Por exemplo, considere que o recurso está associado a duas subclasses de *SimpleType*: *TipoDeProduto* e *Fabricante*. Então o código seria: `return new String[] { "tipoDeProduto", "fabricante"};`
getQuantificationStrategyInstance	-	QuantificationStrategy	Indica a estratégia de quantificação do recurso (Padrão 2). Seu valor pode ser uma instância de uma subclasse de QuantificationStrategy: • *SingleResource* – para um recurso único; • *InstantiableResource* – para um

			recurso que possui cópias; • *MeasurableResource* – para um recurso mensurável ou para um recurso medido em lotes. Por exemplo, considere que o recurso é instanciável, então o código seria: `return new InstantiableResource();`
getMeasureUnityClass	-	Class< ? extends MeasureUnity>	Esse método deve ser implementado em classes que estendem *Resource* somente se a estratégia de quantificação representada pela classe *MeasurableResource* for utilizada. Caso contrário não é necessário fornecer uma implementação para este método na versão OA do GRENJ. Esse método indica uma classe que representa uma unidade de medida. Então o código seria: `return UnidadeMedida.class;`
getResourceInstanceClass	-	Class< ? extends ResourceInstance>	Esse método deve ser implementado em classes que estendem *Resource* somente se a estratégia de quantificação representada pela classe *InstantiableResource* for utilizada. Caso contrário não é necessário fornecer uma implementação para este método na versão OA do GRENJ. Esse método indica uma classe que representa as instâncias do recurso. Por exemplo, considere a classe *DVD* que representa as instâncias do recurso. Então o código seria: `return DVD.class;`
getResourceRentalClass	-	Class< ? extends ResourceRental>	Esse método deve ser implementado em classes que estendem *Resource* somente se o padrão Locar Recurso for utilizado. Caso contrário não é necessário fornecer uma implementação para este método na versão OA do GRENJ. Esse método indica uma classe que representa uma transação de aluguel. Por exemplo, considere que a classe *Locacao* representa a transação de aluguel do recurso. Então o código seria: `return Locacao.class;`

Classe SimpleType

Atributos

Nome	Tipo	Descrição
idCode	Int	Código identificador (chave).
description	String	Descrição ou nome do recurso.

Métodos

Nome	Entrada	Saída	Descrição
typeClasses	-	Class[]	Indica as classes de tipo de recurso. Em *SimpleType* esse método sempre retorna um array vazio, como indica o código: `return new Class[] { };`
typeFieldsInitialize	-	String[]	Indica os nomes dos campos dos tipos de recurso na tabela do recurso do banco de dados. Em *SimpleType* esse método sempre retorna um array vazio, como indica o código: `return new String[] { };`

Classe ResourceInstance

Atributos

Nome	Tipo	Descrição
resourceIdCode	Int	Armazena o número de identificação do recurso.
code	String	Armazena o código de identificação da instância.
location	String	Indica o local onde a instância pode ser encontrada.
status	Status	Indica o status da instância: disponível ou não disponível.

Métodos

Nome	Entrada	Saída	Descrição

getResource	-	Classe que representa o recurso	Retorna o recurso que é representado pela instância. Por exemplo, considere uma classe *DVD* que representa as instâncias do recurso *Filme*, então o código seria:
			`try {`
			`  return (Filme)`
			`  PersistentObject.load(`
			`  getResourceIdCode(),`
			`  Filme.class );`
			`} catch (Exception e) {`
			`  e.printStackTrace();`
			`  return null;}`

Classe ResourceRental

Atributos

Nome	Tipo	Descrição
number	Int	Indica um número de identificação para cada objeto.
date	java.sql.Date	Armazena a data de efetuação da transação.
observation	String	Armazena informações relacionadas com a transação.
status	TransactionStatus	Indica o status da transação: aberta ou fechada.
totalPrice	Money	Armazena o valor inicial da transação.
totalDiscount	Money	Armazena um valor de desconto.
finalTotal	Money	Armazena um valor calculado com base no preço inicial da transação, no desconto, no valor de seus itens e outros valores que podem incidir sobre o valor final.
destinationParty	DestinationParty	Indica o destino do(s) recurso(s) da transação.
items	List<TransactionItem>	Indica os itens da transação, se ela for itemizada.

Métodos

Nome	Entrada	Saída	Descrição
getTransactionQuantificationStrategyInstance	-	TransactionQuantificationStrategy	Indica para a transação qual estratégia de quantificação foi aplicada sobre o recurso. A estratégia utilizada é indicada por uma das classes abaixo: • *SingleResTransaction*, se o recurso for único; • *InstantiableResTransaction*, se o recurso for instanciável; • *MeasurableResTransaction*, se o recurso for mensurável; • *LotableResTransaction*, se o recurso for em lotes. Por exemplo, considere que o recurso é instanciável, então o código seria: `return new InstantiableResTransaction();`

getDestinationPartyClass	-	Class<? extends DestinationParty >	Indica a classe que representa o destino da transação. Por exemplo, se o destino for representado pela classe *Cliente*, então o código seria: `return Cliente.class;`
getExecutorClass	-	Class<? extends TransactionExecutor >	Esse método deve ser implementado em classes que estendem *ResourceRental* somente se o padrão Identificar o Executor da Transação for utilizado. Caso contrário não é necessário fornecer uma implementação para este método na versão OA do GRENJ. Esse método indica a classe que representa o executor da transação. Por exemplo, se o executor da transação for representado pela classe *Funcionario*, então o código seria: `return Funcionario.class;`
getPaymentClass	-	Class<? extends Payment >	Esse método deve ser implementado em classes que estendem *ResourceRental* somente se o padrão Identificar o Executor da Transação for utilizado. Caso contrário não é necessário fornecer uma implementação para este método na versão OA do GRENJ. Esse método indica a classe que representa o pagamento da transação. Por exemplo: `return Cash.class;`
getResourceClass	-	Class<? extends Resource >	Indica a classe que representa o recurso. Por exemplo, se o recurso for representado pela classe *Filme*, então o código seria: `return Filme.class;`
getSourcePartyClass	-	Class<? extends SourceParty >	Esse método deve ser implementado em classes que estendem *ResourceRental* somente se a variante Transação com Source Party for utilizada. Caso contrário não é necessário fornecer uma implementação para esse método na versão GRENJ-OA. Esse método indica a classe que representa a origem da transação. Por exemplo, suponha que a origem em sistema de controle de biblioteca seja a bibliotecária: `return Bibliotecaria.class;`
getFineRateClass	-	Class<? extends FineRate >	Esse método deve ser implementado em classes que estendem *ResourceRental* somente se a variante Transação com Multa for utilizada. Caso contrário não é necessário fornecer uma implementação para este método na versão GRENJ-OA. Esse método indica a classe que representa a taxa de multa da transação de aluguel. Por exemplo, se a taxa de multa for representada pela classe *Multa*, então o código seria: `return Multa.class;`
getTransactionItemClass	-	Class<? extends TransactionItem>	Esse método deve ser implementado em classes que estendem *ResourceRental* somente se o padrão Itemizar a Transação do Recurso for utilizado. Caso contrário não é necessário fornecer uma implementação para este método na versão

			GRENJ-OA. Esse método indica a classe que representa o item da transação. Por exemplo, se o item for representado pela classe *ItemLocacao*, então o código seria: `return ItemLocacao.class;`
getReservationClass	-	Class<? extends ResourceReservation>	Esse método deve ser implementado em classes que estendem *ResourceRental* somente se o padrão Reservar Recurso for utilizado. Caso contrário não é necessário fornecer uma implementação para este método na versão GRENJ-OA. Esse método indica a classe que representa uma transação de reserva. Por exemplo, em um sistema de locadora de DVDs é possível efetuar a reserva do filme antes da locação efetiva: `return Reserva.class;`
transactionPluralName	-	String	Indica o formado em plural do nome da transação. Por exemplo: `return "Locações";`

Classe DestinationParty

Atributos

Nome	Tipo	Descrição
idCode	Int	Código identificador (chave).
description	String	Descrição ou nome do recurso.

Classe TransactionItem

Atributos

Nome	Tipo	Descrição
transaction	BusinessResourceTransaction	Armazena a transação a qual o item está associado.
resource	Resource	Armazena o recurso relacionado com o item da transação.
aValue	Double	Armazena o valor do recurso no item da transação.

Métodos

Nome	Entradas	Saída	Descrição

Construtor com argumentos (use o próprio nome da classe)	ResultSet result, Index anIndex, BusinessResourceTransaction transaction	-	Na aplicação, o código do construtor com argumentos deve chamar o construtor de sua super classe:
			`super(result, anIndex, transaction);`
			Além disso, se a classe possuir atributos próprios é necessário inicializá-los com os valores provenientes do banco de dados dentro de um bloco de tratamento de exceção (*try catch*). Por exemplo, para um atributo do tipo inteiro chamado *ano*, o código seria:
			`try { ano = result.getInt( anIndex.getIndex() ); anIndex.incrementIndexByOne(); } catch ( SQLException e ) { e.printStackTrace();}`

getResourceClass	-	Class<? extends Resource >	Indica a classe que representa o recurso. Por exemplo, se o recurso for representado pela classe *Filme*, então o código seria:

			`return Filme.class;`
getTransactionClass	-	Class<? extends BusinessResourceTransaction >	Indica a classe que representa a transação. Por exemplo, se a transação for representada pela classe *Locacao*, então o código seria: `return Locacao.class;`
getItemQuantificationStrategyInstance	-	ItemQuantificationStrategy	Indica para o item da transação qual estratégia de quantificação foi aplicada sobre o recurso. A estratégia utilizada é indicada por uma das classes abaixo: • *SingleResTransItem*, se o recurso for único; • *InstResTransItem*, se o recurso for instanciável; • *MeasResTransItem*, se o recurso for mensurável; • *LotResTransItem*, se o recurso for em lotes. Por exemplo, considere que o recurso é instanciável, então o código seria: `return new InstResTransItem();`

Exemplo de Roteiro para Execução de Cenários de Manutenção do Experimento

Cenário de Manutenção: implementar a variante Recurso Instanciável no *framework* GRENJ-OA.

Para fazer isso vocês devem:

1 – Analisar a tabela (Figura 1), na qual são listadas as classes e os aspectos que implementam métodos para indicar ao *framework* o uso da variante Recurso Instanciável do padrão Quantificar o Recurso.

Aspecto	Pacote	Classe	Método	Entradas	Saída	Descrição
InstantiableQuantStrategyContainerLoaderAspect	grenj.model.quantifyresource.instantiableresource	Resource	getInstances	-	List<ResourceInstance>	O objetivo aqui é retornar uma lista de instâncias do recurso proveniente da estratégia de quantificação. Exemplo: `return quantification.getInstances();`
			setInstances	List<ResourceInstance> instances	void	O objetivo aqui é definir a lista de instâncias do recurso. Exemplo: `this.quantification.setInstances(instances);`
			getResourceInstanceClass	-	Class<? extends ResourceInstance>	Abstrato
		QuantificationStrategy	getInstances	-	List<ResourceInstance>	Abstrato
			setInstances	List<ResourceInstance> instances	void	Abstrato

Figura 1. Métodos relacionados com o uso da variante recurso instanciável.

Apêndice G

EXEMPLOS DE CASOS DE TESTE E SCRIPT DE BASE DE DADOS

Caso de Teste Manutenção 1: Adicionar estratégia de quantificação Recurso Instanciável

```
01 package grenj.model.instantiation.test;
02
03 import grenj.model.instantiation.DVD;
04 import grenj.model.instantiation.Filme;
05 import grenj.util.Status;
06
07 import java.util.Random;
08
09 public class Main {
10
11     public static void main(String[] args) {
12
13           Random rd = new Random();
14           rd.setSeed(System.currentTimeMillis());
15
16           //Cadastra um objeto do tipo Filme no Banco de dados
17           Filme film = new Filme();
18           film.setDescription("Demolidor");
19           film.setAno(2005);
20           film.setIdCode(rd.nextInt(1000)+1);
21
22           if(film.save() == 1){
23                 System.out.println("Success");
24           } else{
25                 System.out.println("Error");
26           }
27
28           //Cadastra um objeto do tipo DVD no banco de dados
29           DVD dvd = new DVD();
30           dvd.setResourceIdCode(film.getIdCode());
31           dvd.setLocation("a1");
32           dvd.setCode("1");
33           dvd.setStatus(Status.Available);
34
35           if (dvd.save() == 1) {
36                 System.out.println("Success");
37           } else {
38                 System.out.println("Error");
39           }
40     }
41 }
```

Caso de Teste Manutenção 2: Adicionar Itemização do Recurso

```
01 package grenj.model.instantiation.test;
02
03 import grenj.model.TransactionItem;
04 import grenj.model.instantiation.*;
05 import grenj.util.Status;
06 import grenj.util.TransactionStatus;
07 import java.util.ArrayList;
08 import java.util.List;
09 import java.util.Random;
10
11 public class Main {
12
13     public static void main(String[] args) {
14
15            Random rd = new Random();
16            rd.setSeed(System.currentTimeMillis());
17
18
19            //Cadastra um objeto do tipo Filme no Banco de dados
20            Filme film = new Filme();
21            film.setDescription("Demolidor");
22            film.setAno(2005);
23            film.setIdCode(rd.nextInt(1000)+1);
24
25            if(film.save() == 1){
26                   System.out.println("Success");
27            } else{
28                   System.out.println("Error");
29            }
30
31            //Cadastra um objeto do tipo DVD no banco de dados
32            DVD dvd = new DVD();
33            dvd.setResourceIdCode(film.getIdCode());
34            dvd.setLocation("a1");
35            dvd.setCode("1");
36            dvd.setStatus(Status.Available);
37
38            if (dvd.save() == 1) {
39                   System.out.println("Success");
40            } else {
41                   System.out.println("Error");
42            }
43
44            //Cadastra um objeto do tipo Cliente no banco de dados
45            Cliente cli = new Cliente();
46            cli.setIdCode(rd.nextInt(1000)+1);
47            cli.setDescription("Luiz Antônio da Silva");
48            cli.setCpf("02739873987");
49            cli.setTelefone("0xx1688223800");
50
51            if (cli.save() == 1) {
52                   System.out.println("Success");
53            } else {
54                   System.out.println("Error");
55            //Cadastra um objeto do tipo Locação no banco de dados
56            Locacao loc = new Locacao();
57
58            List<TransactionItem> items = new ArrayList<TransactionItem>();
59            ItemLocacao it1 = new ItemLocacao();
```

```
60            it1.setAValue(2);
61            it1.setTransaction(loc);
62            it1.setInstanceCode(dvd.getCode());
63            it1.setResource(film);
64            items.add(it1);
65
66            loc.setItems(items);
67
68            loc.setResource(film);
69            loc.setDestinationParty(cli);
70            loc.setNumber(rd.nextInt(1000)+1);
71            loc.setObservation("Locação feita com desconto de 0,50 centavos");
72
73            loc.setBeginDate(java.sql.Date.valueOf("2010-05-31"));
74            loc.setFinishingDate(java.sql.Date.valueOf("2010-06-03"));
75            loc.setStatus(TransactionStatus.option1);
76            loc.setTotalDiscount(0.50);
77
78            if (loc.save() == 1) {
79                  System.out.println("Success");
80            } else {
81                  System.out.println("Error");
82            }
83      }
84 }
```

Caso de Teste Instanciação 1: Sistema de Biblioteca:

```
01 package test;
02
03 import grenj.model.QualifiableObject;
04 import grenj.model.businesstransaction.transactiondetails.itemizeresource
05 transaction.TransactionItem;
06 import grenj.model.businesstransaction.util.TransactionStatus;
07 import instantiation.*;
08 import grenj.util.Status;
09 import java.sql.Date;
10 import java.util.ArrayList;
11 import java.util.List;
12 import java.util.Random;
13
14 public class Main {
15
16    public static void main(String[] args) {
17
18           Random rd = new Random();
19           rd.setSeed(System.currentTimeMillis());
20
21           // Cadastra um objeto do tipo TipoObra no banco de dados
22           TipoObra tipoObra = new TipoObra();
23           tipoObra.setIdCode(rd.nextInt(1000) + 1);
24           tipoObra.setDescription("Livro");
25
26           if (tipoObra.save() == 1) {
27                 System.out.println("Success");
28           } else {
29                 System.out.println("Error");
30           }
31
32           // Cadastra um objeto do tipo ObraLiteraria no banco de dados
33           ObraLiteraria obra = new ObraLiteraria();
34           obra.setDescription("Memórias Póstumas de Braz Cubas");
35           obra.setIdCode(rd.nextInt(1000)+1);
36           obra.setDataPublicacao(new Date(System.currentTimeMillis()));
37           obra.setNroPaginas(400);
38
39           List<QualifiableObject> types = new ArrayList<QualifiableObject>();
40           types.add(tipoObra);
41           obra.setTypes(types);
42
43           if (obra.save() == 1) {
44                 System.out.println("Success");
45           } else {
46                 System.out.println("Error");
47           }
48
49           // Cadastra um objeto do tipo Exemplar no banco de dados
50           Exemplar exemplar = new Exemplar();
51           exemplar.setResourceIdCode(obra.getIdCode());
52           exemplar.setLocation("Prateleira a1");
53           exemplar.setCode(String.valueOf(rd.nextInt(1000)+1));
54           exemplar.setStatus(Status.Available);
55
56           if (exemplar.save() == 1) {
57                 System.out.println("Success");
58           } else {
59                 System.out.println("Error");
```

```
60          }
61           // Cadastra um objeto do tipo Emprestimo no banco de dados
62           Emprestimo emprestimo = new Emprestimo();
63
64           List<TransactionItem> items = new ArrayList<TransactionItem>();
65           ItemEmprestimo it1 = new ItemEmprestimo();
66           it1.setAValue(2);
67           it1.setTransaction(emprestimo);
68           it1.setInstanceCode(exemplar.getCode());
69           it1.setResource(obra);
70           items.add(it1);
71
72           emprestimo.setItems(items);
73
74           emprestimo.setDestinationParty(leitor);
75           emprestimo.setNumber(rd.nextInt(1000)+1);
76           emprestimo.setObservation("Emprestimo de 3 dias");
77
78           emprestimo.setBeginDate(java.sql.Date.valueOf("2010-05-31"));
79           emprestimo.setFinishingDate(java.sql.Date.valueOf("2010-06-03"));
80           emprestimo.setStatus(TransactionStatus.option1);
81           emprestimo.setTotalDiscount(0.00);
82           if (emprestimo.save() == 1) {
83                 System.out.println("Success");
84           } else {
85                 System.out.println("Error");
86           }
87     }
88 }
```

Caso de Teste Instanciação 2: Sistema de Hotel:

```
01 package test;
02
03 import grenj.model.QualifiableObject;
04 import grenj.model.TransactionItem;
05 import instantiation.*;
06 import grenj.util.Status;
07 import grenj.util.TransactionStatus;
08 import java.util.ArrayList;
09 import java.util.List;
10 import java.util.Random;
11
12 public class Main {
13
14     public static void main(String[] args) {
15
16           Random rd = new Random();
17           rd.setSeed(System.currentTimeMillis());
18
19           // Cadastra um objeto do tipo TipoAcomodacao no banco de dados
20           TipoAcomodacao tipo = new TipoAcomodacao();
21           tipo.setIdCode(rd.nextInt(1000) + 1);
22           tipo.setDescription("Suite");
23           if (tipo.save() == 1) {
24                 System.out.println("Success");
25           } else {
26                 System.out.println("Error");
27           }
```

```
28
29          // Cadastra um objeto do tipo Acomodacao no banco de dados
30          Acomodacao acomodacao = new Acomodacao();
31          acomodacao.setDescription("Dus camas e banheiro");
32          acomodacao.setIdCode(rd.nextInt(1000) + 1);
33          acomodacao.setNroCamas(2);
34
35          List<QualifiableObject> types = new ArrayList<QualifiableObject>();
36          types.add(tipo);
37          acomodacao.setTypes(types);
38
39          if (acomodacao.save() == 1) {
40                System.out.println("Success");
41          } else {
42                System.out.println("Error");
43          }
44
45          // Cadastra um objeto do tipo Quarto no banco de dados
46          Quarto quarto = new Quarto();
47          quarto.setResourceIdCode(acomodacao.getIdCode());
48          quarto.setLocation("Primeiro Andar");
49          quarto.setCode(String.valueOf(rd.nextInt(1000) + 1));
50          quarto.setStatus(Status.Available);
51
52          if (quarto.save() == 1) {
53                System.out.println("Success");
54          } else {
55                System.out.println("Error");
56          }
57
58          // Cadastra um objeto do tipo Hospede no banco de dados
59          Hospede hospede = new Hospede();
60          hospede.setIdCode(rd.nextInt(1000) + 1);
61          hospede.setDescription("Maria Luiza da Silva");
62          hospede.setCidade("São Carlos");
63          hospede.setPais("Brasil");
64
65          if (hospede.save() == 1) {
66                System.out.println("Success");
67          } else {
68                System.out.println("Error");
69          }
70
71          // Cadastra um objeto do tipo Hospedagem no banco de dados
72
73          Hospedagem hospedagem = new Hospedagem();
74          List<TransactionItem> items = new ArrayList<TransactionItem>();
75          QuartoHospedado it1 = new QuartoHospedado();
76          it1.setAValue(100);
77          it1.setTransaction(hospedagem);
78          it1.setInstanceCode(quarto.getCode());
79          it1.setResource(acomodacao);
80          items.add(it1);
81          hospedagem.setItems(items);
82          hospedagem.setDestinationParty(hospede);
83          hospedagem.setNumber(rd.nextInt(1000) + 1);
84          hospedagem.setObservation("Hospedagem com desconto");
85
86          hospedagem.setBeginDate(java.sql.Date.valueOf("2010-05-31"));
87          hospedagem.setFinishingDate(java.sql.Date.valueOf("2010-06-03"));
```

```
88            hospedagem.setStatus(TransactionStatus.option1);
89            hospedagem.setTotalDiscount(0.00);
90            if (hospedagem.save() == 1) {
91                  System.out.println("Success");
92            } else {
93                  System.out.println("Error");
94            }
95     }
96 }
```

Exemplo de Script de Base de Dados

```
DROP DATABASE IF EXISTS bibliotecaoa;
CREATE DATABASE bibliotecaoa;
USE bibliotecaoa;

DROP TABLE IF EXISTS TipoObra;
CREATE TABLE TipoObra (
       idCode INTEGER NOT NULL ,
    description VARCHAR(50)
       ,PRIMARY KEY (idCode));

DROP TABLE IF EXISTS Leitor;
CREATE TABLE Leitor (
       idCode INTEGER NOT NULL ,
    description VARCHAR(50),
       cpf VARCHAR(50),
       email VARCHAR(50)
       ,PRIMARY KEY (idCode));

DROP TABLE IF EXISTS ObraLiteraria;
CREATE TABLE ObraLiteraria (
       idCode INTEGER NOT NULL ,
    description VARCHAR(50),
    TipoObra INTEGER,
       dataPublicacao DATE,
       nroPaginas INTEGER
       ,PRIMARY KEY (idCode));

DROP TABLE IF EXISTS Exemplar;
CREATE TABLE Exemplar (
       resourceIdCode INTEGER NOT NULL,
    code CHAR(10) NOT NULL,
    location VARCHAR(20),
    status CHAR
       ,PRIMARY KEY (resourceIdCode,code));

DROP TABLE IF EXISTS Emprestimo;
CREATE TABLE Emprestimo (
       number INTEGER NOT NULL ,
    date DATE,
    observation VARCHAR(100),
    status CHAR,
    totalPrice FLOAT,
    totalDiscount FLOAT,
    destinationParty INTEGER,
    finishingDate DATE,
    returnDate DATE
       ,PRIMARY KEY (number));
DROP TABLE IF EXISTS ItemEmprestimo;
CREATE TABLE ItemEmprestimo (
       transaction INTEGER NOT NULL,
    value FLOAT,
    instanceCode INTEGER,
    resource INTEGER NOT NULL,
       ,PRIMARY KEY (transaction,resource));
```