UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

“Construção da Camada de Interface Gráfica e de um Wizard para o Framework GRENJ”

Matheus Carvalho Viana

São Carlos/SP
Maio/2009

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

V614cc

Viana, Matheus Carvalho.
Construção da camada de interface gráfica e de um wizard para o framework GRENJ / Matheus Carvalho Viana. -- São Carlos : UFSCar, 2009.
117 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Análise e projeto de sistemas. 2. Framework (Programa de computador). 3. Interface gráfica de computador. 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

"Construção da Camada de Interface Gráfica e de um Wizard para o Framework GRENJ"

MATHEUS CARVALHO VIANA

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 Ap. **Dellosso** Penteado
(Orientadora - DCNFSCar)

Prof. Dr. Antonio Francisco do Prado
(DCNFSCar)

Profa. **Dra.** Selma Shin Shimizu **Melnikoff**
(POLINSP)

São Carlos
Maio/2009

Este trabalho é dedicado aos meus pais e aos meus irmãos.

Agradecimentos

Primeiramente, agradeço a Deus pela oportunidade de fazer um curso de mestrado e por muitos outros acontecimentos em minha vida.

À minha mãe, ao meu pai, ao meu irmão e à minha irmã por sempre acreditarem no meu potencial e por oferecerem o seu apoio quando precisei.

À Professora Dra. Rosângela por ter me escolhido como seu orientando. Obrigado pela amizade, pela confiança e por me ajudar a evoluir como aprendiz e pesquisador.

À Professora Dra. Rosana Braga pelas explicações e pelas sugestões que contribuíram para a realização do trabalho.

A todos os meus amigos e companheiros de mestrado por sofrerem e se divertirem junto comigo. Espero ter sido um bom amigo tanto quanto vocês foram para mim. Sou muito feliz por ter estado aqui na mesma época que vocês.

Aos meus amigos do Kung Fu Taisan. Além de ganhar novos amigos, pude relaxar a minha mente e aprender que uma pessoa é capaz de estender os seus limites com concentração e dedicação.

Ao CNPq pelo apoio financeiro ao meu trabalho.

Resumo

As empresas utilizam sistemas de informação para executarem seus processos de maneira eficiente. O mercado exige que esses sistemas sejam desenvolvidos em curto prazo, com baixo custo e que possuam alta qualidade e flexibilidade. Reúso é um conceito da Engenharia de Software que objetiva atender a essa demanda com a reutilização de artefatos, previamente construídos e testados, para o desenvolvimento de novos sistemas. Sistemas de informação podem ser desenvolvidos com o apoio de técnicas baseadas em reúso, como frameworks orientados a objetos e geradores de aplicações. Este projeto de pesquisa apresenta o processo de construção da camada de interface gráfica e de um wizard gerador de aplicações para o framework GRENJ (Gestão de Recursos de Negócios com implementação em Java). Nesse processo, o framework Guiwe (do inglês, ***G****raphical* ***u****ser* ***i****nterface for* ***we****b*) foi criado para apoiar a construção e a instanciação da camada de interface do GRENJ que, além da linguagem Java, utiliza tecnologias voltadas para a web. Por sua vez, o wizard do framework GRENJ foi desenvolvido com o apoio de um gerador de aplicações configurável e com a construção de gabaritos e da ferramenta GRENJ-Gens. Essa ferramenta permite a instanciação de sistemas no framework GRENJ com a utilização de diferentes geradores de aplicações que tenham por base os gabaritos desse framework. Tanto a camada de interface gráfica quanto a do wizard do framework GRENJ foram construídos de forma iterativa e intercalados entre si com apoio nos padrões da GRN (Gestão de Recursos de Negócios), que é a linguagem de padrões da qual foi desenvolvido o framework GRENJ. Além disso, a prática de desenvolvimento guiado por testes também foi utilizada. As vantagens do uso do GRENJ com o wizard e a interface gráfica foram verificadas com a realização de alguns estudos de caso.

Abstract

Enterprises need information systems for executing their processes on an efficient way. Market demands that these systems are developed at short term, at low cost and they need to get high quality and flexibility. Reuse is a concept of Software Engineering that aims to take this demand into consideration with the reutilization of artefacts, previously built and tested, for developing new systems. Information systems can be developed with reuse-based techniques, such as object-oriented frameworks and application generators. This project presents the process of construction of the graphical user interface layer and an application generator wizard of GRENJ (Business Resource Management implemented in Java, in Portuguese, *Gestão de REcursos de Negócios com implementação em Java*) framework. In this process, Guiwe (**G**raphical **u**ser **i**nterface for **we**b) framework was created to support the building and the instantiation of the graphical user interface layer of GRENJ, which makes use of Java language plus web technologies. The wizard of GRENJ framework was developed with the use of a configurable application generator and the creation of templates and the GRENJ-Gens tool. This tool provides the instantiation of systems on GRENJ framework with the support of different application generators that can utilize the templates of that framework. The graphical user interface and the wizard layers were built on an iterative and intercalated way following the sequence of the patterns of GRN (Business Resource Management, in Portuguese, *Gestão de Recursos de Negócios*), witch is the pattern language that originated GRENJ framework. Beyond this, the practice of test-driven development was also applied. The advantages of using GRENJ framework with its graphical user interface and its wizard layers were verified with the realization of some case studies.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AJAX: *Asynchronous Javascript And XML*. XML e JavaScript Assíncrono.

CSS: *Cascading Style Sheets*. Folhas de estilo em cascata.

DOM: *Document Object Model*. Modelo de objetos do documento.

DSOO: Desenvolvimento de Software Orientado a Objetos.

EAF: *Enterprise Application Frameworks*. Frameworks para aplicações empresariais.

GA: Gerador de Aplicações.

GAC: Gerador de Aplicações Configurável.

GRN: linguagem de padrões Gestão de Recursos de Negócios.

GREN: framework Gestão de Recursos de Negócios.

GRENJ: framework Gestão de Recursos de Negócios com implementação em Java.

Guiwe: framework *Graphical user interface for web*.

HTML: *HyperText Markup Language*. Linguagem de marcação de hipertexto.

IDE: *Integrated Development Environment*. Ambiente de desenvolvimento integrado.

JDK: *Java Development Kit*. Pacote de desenvolvimento da linguagem Java.

JSP: *Java Server Pages*. Versão da linguagem HTML que contém instruções Java.

LMA: Linguagem de Modelagem da Aplicação.

LP: Linguagem de Padrões.

LPS: Linha de Produtos de Software.

MIF: *Middleware Integration Frameworks*. Frameworks intermediários de integração.

MVC: padrão arquitetural *Model-View-Controller*.

SIF: *System Infrastructure Frameworks*. Frameworks de sistemas para infra-estrutura.

TDD: *Test-Driven Development*. Desenvolvimento guiado por testes.

UML: *Unified Modeling Language* – Linguagem de modelagem unificada.

XHTML: *eXtended HyperText Markup Language*. Linguagem estendida de marcação de hipertexto.

XML: *eXtensible Markup Language*. Linguagem de marcação extensível.

XP: método de desenvolvimento ágil *eXtreme Programming*.

XSL: *eXtensible Stylesheet Language*. Linguagem de estilo extensível.

XSL-FO: *eXtensible Stylesheets Language Formating Objects*. Linguagem de estilo extensível para formatação de objetos.

XSLT: *eXtensible Stylesheets Language Transformations*. Linguagem de estilo extensível para transaformação.

W3C: *World Wide Web Consortium*. Órgão para padronização de tecnologias web.

1 Introdução

1.1. Considerações Iniciais

O intuito da Engenharia de Software é desenvolver software de boa qualidade no menor tempo possível (Pressman, 2006). Uma das formas mais comuns de se atingir esse objetivo é com a prática do reúso. Nesse sentido, as linguagens de programação evoluíram para adotar conceitos relacionados com o reúso cada vez mais eficazes como, por exemplo, modularização, classes, objetos, composição, herança e polimorfismo. Um dos resultados da evolução das linguagens foi o desenvolvimento de software orientado a objetos. Contudo, esses conceitos fornecem somente reúso em nível de código. Com isso, técnicas foram criadas para aumentar a eficiência no desenvolvimento de um sistema de informação sem que haja perda em sua qualidade, tais como padrões de software, linguagens de padrões, frameworks orientados a objetos e geradores de aplicações.

Padrões de software, ou, simplesmente, padrões, fornecem soluções de sucesso para determinado problema em um contexto (Gamma *et al.*, 1995). Essa definição indica que padrões proporcionam o reúso de experiência no desenvolvimento de software (Fowler, 1997), pois descrevem como o desenvolvedor deve proceder em determinadas situações.

Em um nível mais alto em relação aos padrões, as linguagens de padrões são aplicadas no desenvolvimento de software completo pertencente a um domínio específico, enquanto que um simples padrão resolve apenas um problema isolado dentro de um dado software (Coplien, 1996). Os padrões constituintes de uma linguagem de padrões são organizados de maneira que seus elementos trabalhem em uma seqüência ordenada de passos para resolução de problemas referentes ao desenvolvimento de software (Schmidt *et al.*, 1996).

Frameworks são soluções reutilizáveis, de todo ou parte do sistema, em que entidades abstratas são customizadas com o propósito de definir o comportamento específico (Johnson, 1997a). Além de reúso de código, frameworks fornecem também

reúso de projeto, propiciando maior eficiência no desenvolvimento de um sistema de informação.

Outra maneira de desenvolver software mais rapidamente é por meio de geradores de aplicações, que são capazes de traduzir de maneira automatizada requisitos em uma aplicação (Wu *et al.*, 2003). Um gerador de aplicações também pode ser utilizado no desenvolvimento de sistemas pertencentes a um domínio especifico (Thibault e Consel, 1997) (Braga e Masiero, 2002).

Por reduzir o esforço de desenvolvimento de um sistema de informação, a utilização de frameworks e de geradores de aplicações tem crescido gradualmente (Johnson, 1997a) (Wu *et al.*, 2003). Esse é o contexto na qual se aplica este trabalho, cuja proposta é construir uma camada de interface gráfica para o framework GRENJ (Gestão de REcursos de Negócios com implementação em Java) e um wizard que faz uso de geradores de aplicações para instanciar sistemas baseados nesse framework.

1.2. Motivação do Trabalho

O framework GRENJ pode apoiar o desenvolvimento de sistemas implementados em linguagem Java pertencentes ao domínio de gestão de recursos de negócios (Durelli, 2008). Sua construção é o resultado de um processo de reengenharia iterativo das camadas de persistência e de negócios do framework GREN (Gestão de REcursos de Negócios), que é implementado em linguagem Smalltalk.

Apesar de prover reúso de código e projeto e, consequentemente, reduzir o esforço de desenvolvimento de um sistema, a utilização do framework GRENJ apresenta dificuldades, como:

- Sua utilização exige que o desenvolvedor tenha conhecimento de suas classes, de seus relacionamentos e dos métodos que devem ser sobrescritos. Em outras palavras, o desenvolvedor precisa conhecer os pontos variáveis (*hot spots*) do framework para instanciá-lo.
- O desenvolvedor precisa construir a camada de interface gráfica do sistema sem o apoio do framework GRENJ, assim como a interligação dessa camada com as de persistência e de negócios. Em muitos sistemas, a construção da camada de interface gráfica representa a maior parte do esforço gasto em seu desenvolvimento, principalmente, no caso de sistemas voltados para a web.

A existência de uma camada de interface gráfica para ao framework GRENJ reduz o esforço necessário no desenvolvimento de sistemas instanciados a partir desse framework, bem como os erros que podem ocorrer na construção dessa camada e na sua interligação com as demais. Além disso, o uso de tecnologias voltadas para a web na implementação da camada de interface gráfica do framework GRENJ possibilita a utilização distribuída dos sistemas instanciados a partir desse framework.

Por fim, a existência de um *wizard* que utilize geradores de aplicações para a instanciação automatizada de sistemas baseados no framework GRENJ dispensa a necessidade de conhecimento da arquitetura desse framework e, até mesmo, da linguagem Java e de outras tecnologias utilizadas em sua implementação. Dessa forma, o desenvolvedor informa para o wizard as características específicas do sistema que deseja instanciar para, então, gerá-lo de forma automatizada. Além da facilidade de uso, obtém-se eficiência na construção de um sistema de informação e na sua manutenção, pois, quando ocorre uma mudança nos requisitos, pode-se novamente usar esse gerador para produzir uma outra versão do sistema que atenda a nova realidade (Braga, 2002).

1.3. Objetivo do Trabalho

Os principais objetivos deste trabalho são: a construção da camada de interface gráfica do framework GRENJ e a criação de um wizard que faz uso de geradores de aplicações para instanciar sistemas baseados nesse framework. Com isso, espera-se obter os seguintes benefícios:

1. Aumento do percentual de reúso proporcionado pelo framework, que passa a abranger todas as camadas de um sistema de software.
2. Aumento significativo da eficiência no desenvolvimento de sistemas de informação instanciados a partir do framework GRENJ, pois o desenvolvedor passa a contar com seu apoio na construção da camada de interface gráfica e de sua interligação com as demais.
3. Padronização das interfaces gráficas com o usuário dos sistemas instanciados a partir do framework GRENJ, garantindo-lhes maior usabilidade.
4. Melhoria da qualidade dos sistemas instanciados a partir do framework GRENJ, pois a camada de interface gráfica com o usuário será composta de classes previamente testadas, assim como as demais.

5. Os sistemas instanciados a partir do framework GRENJ poderão ser acessados remotamente por meio de uma rede de computadores.
6. Maior facilidade na instanciação de sistemas a partir do framework GRENJ por meio do seu wizard.
7. Desenvolvimento automatizado de sistemas de informação no domínio de gestão de recurso de negócios a partir de informações passadas pelo desenvolvedor ao wizard do framework GRENJ e do apoio da linguagem de padrões GRN.
8. Redução do tempo de desenvolvimento de um sistema de informação com a utilização do wizard do framework GRENJ.
9. Redução do custo de desenvolvimento de sistemas de informação pertencentes ao domínio de gestão de recursos de negócios.

1.4. Trabalhos Relacionados

Não foram encontrados outros trabalhos que tratam da construção da camada de interface gráfica e de um wizard para um framework existente. Normalmente, wizards geradores de aplicações são construídos para a geração de artefatos de sistemas não instanciados de frameworks como, por exemplo, o wizard GAwCRe (Pazin, 2004). O wizard proposto neste trabalho tem como objetivo gerar sistemas instanciados do framework GRENJ.

A construção da camada de interface gráfica do framework GRENJ deu origem a um novo framework, denominado Guiwe (do inglês, ***G****raphic* ***u****ser* ***i****nterface for* ***we****b*), que pode ser utilizado para a construção de uma camada de interface gráfica web para sistemas de informação e para frameworks de domínio específico. Em um outro trabalho, Chaves *et al.* (2008) apresentam um estudo experimental sobre o framework Fast Interface Build (FIB), que foi desenvolvido com o intuito de apoiar a construção da camada de interface gráfica de sistemas implementados em linguagem Java. A interface criada por esse framework fornece meios para efetuação de operações de persistência de dados, assim como o framework Guiwe. Entretanto, o framework FIB é baseado na biblioteca Swing da linguagem Java (Sun Microsystems, 2009a) e, portanto, é capaz de criar interfaces somente para sistemas de informação *stand-alone*. O framework Guiwe faz uso de tecnologias web e pode ser utilizado tanto em sistemas de informação quanto em outros frameworks.

1.5. Organização da Dissertação

Além deste Capítulo introdutório, nos demais são apresentados os conceitos aplicados, atividades realizadas, os produtos desenvolvidos e os resultados obtidos neste trabalho.

No **Capítulo 2** são apresentadas abordagens de desenvolvimento de software com o uso de tecnologias, métodos e práticas, como linguagens orientadas a objetos, tecnologias voltadas para a web, linhas de produto de software, padrões, linguagens de padrões, frameworks, geradores de aplicações e desenvolvimento guiado por testes.

No **Capítulo 3** são apresentados a linguagem de padrões GRN, o framework GRENJ e o gerador de aplicações configurável Captor, que servem de base para o desenvolvimento deste trabalho.

No **Capítulo 4** é tratada a construção da camada de interface gráfica do framework GRENJ. Essa atividade deu origem a outro framework, denominado Guiwe, que também é apresentado nesse Capítulo.

No **Capítulo 5** é abordada a construção de um wizard para o framework GRENJ e da ferramenta GRENJ-Gens, que foram desenvolvidos para tornar o processo de instanciação de sistemas baseados no framework GRENJ mais simples e eficiente.

No **Capítulo 6** são descritos estudos de caso que descrevem o desenvolvimento de sistemas baseados no framework GRENJ com o apoio do seu wizard. Nesse Capítulo, também são feitas comparações com o desenvolvimento de sistemas similares com o uso da linguagem de padrões GRN, o framework GRENJ e seu wizard.

No **Capítulo 7** são apresentadas as considerações finais a respeito deste trabalho, com enfoque em suas contribuições e restrições. Sugestões de trabalhos futuros também são apresentadas com o objetivo de incitar a evolução das ferramentas desenvolvidas.

2 Desenvolvimento de Software com Reúso

2.1. Considerações Iniciais

As empresas dependem de sistemas de informação para executarem seus processos de maneira eficiente. O dinamismo do mercado exige que os sistemas de informação sejam desenvolvidos em curto prazo, com baixo custo e possuam alta qualidade e flexibilidade. Contudo, no modelo tradicional de desenvolvimento de software essas características tendem a ser conflitantes. A qualidade é sacrificada e o custo se eleva quando há pressa na finalização do projeto, assim como a construção de um sistema flexível pode exigir um projeto com prazos e custos maiores.

Um dos meios criados pela Engenharia de Software para atender à demanda imposta pelo mercado se baseia no conceito de reúso, que fornece redução no esforço de desenvolvimento e melhoria na qualidade do software construído (Shiva e Shala, 2007). Além disso, o reúso pode ocorrer em diferentes níveis de abstração, etapas do desenvolvimento de software e artefatos.

O Desenvolvimento de Software Orientado a Objetos surgiu como um paradigma em que os sistemas são construídos com base em entidades inter-relacionadas, conhecidas como classes (Pressman, 2006). Uma classe pode ser reutilizada para originar outras classes, o que proporciona o reúso de código e, consequentemente, garante vantagens ao desenvolvimento orientado a objetos.

Com o avanço das redes de computadores, as empresas perceberam que poderiam acelerar seus processos ao tornar suas informações acessíveis em localidades remotas: filiais, empresas parceiras, terminais para clientes, Internet, entre outros. Por conta dessa demanda, surgiram as tecnologias de desenvolvimento de software voltado para a web. Essas tecnologias permitem o reúso de código, de forma semelhante ao que ocorre no desenvolvimento orientado a objetos.

A evolução do desenvolvimento de software resultou no surgimento de conceitos e técnicas que proporcionam reúso em outros níveis de abstração, além do de código. Padrões de projeto (Gamma *et al.*, 1995), frameworks orientados a objetos

(Johnson, 1997a), geradores de aplicações (Thibault e Consel, 1997) e linhas de produtos de software (Weiss e Lai, 1999) são exemplos desses conceitos e técnicas.

Práticas também podem ser adotadas durante o desenvolvimento de software com o intuito de aumentar a flexibilidade e a qualidade dos sistemas de informação produzidos. Iteratividade, *test first* e refatoração são exemplos dessas práticas e são utilizadas na estratégia de desenvolvimento guiada por testes (Janzen e Saiedian, 2008).

Nesse contexto, este Capítulo aborda os temas relacionados ao desenvolvimento de software com reúso, que servem de embasamento para este trabalho. Na Seção 2.2 são tratadas as características básicas de desenvolvimento orientado a objetos e, na Seção 2.3, de desenvolvimento voltado para a web. Na Seção 2.4 linhas de produto de software são abordadas. Padrões de software e linguagens de padrões são apresentados na Seção 2.5 e na Seção 2.6 é abordado o desenvolvimento de software com o uso de frameworks. Na Seção 2.7, desenvolvimento de software com o apoio de frameworks baseados em linguagens de padrões é comentado. Uso de geradores de aplicação é tratado na Seção 2.8; na Seção 2.9, é apresentada a estratégia de desenvolvimento guiado por testes e, finalmente, na Seção 2.10 são apresentadas as considerações finais.

2.2. Desenvolvimento de Software Orientado a Objetos

O Desenvolvimento de Software Orientado a Objetos (DSOO) tem como base a idéia de que as entidades do mundo real, físicas ou abstratas, podem ser vistas como objetos que possuem atributos e operações e que podem interagir uns com os outros. Por exemplo, uma pessoa tem atributos como nome, endereço, peso, altura, número do documento e operações como andar, enxergar, escrever e conversar com outras pessoas. O paradigma orientado a objeto surgiu na década de 1960, mas somente trinta anos depois que se tornou o paradigma de programação preferido da maioria dos desenvolvedores (Pressman, 2006).

No paradigma orientado a objetos, os atributos e as operações de objetos semelhantes são definidos em uma classe. Considerando-se a reutilização de software, a principal vantagem em utilizar programação orientada a objetos é o reúso obtido tanto por meio de herança quanto por composição (Pressman, 2006). Além de evitar a reescrita de código, a herança permite agrupar objetos de classes que herdam direta ou indiretamente a mesma classe. Finalmente, o polimorfismo fornece a capacidade de uma operação possuir comportamentos diferentes em situações distintas.

O paradigma orientado a objetos representa mais do que o uso de uma linguagem orientada a objetos, como Java (Sun Microsystems, 2009a) ou Smalltalk (Smalltalk Dot Org, 2009). As etapas do processo de desenvolvimento de software, como análise, projeto, implementação, testes e manutenção, são influenciadas pelas características desse paradigma (Pressman, 2006). De maneira semelhante, ferramentas, métodos, técnicas e práticas tiveram de ser adaptados ou criados para que o processo de desenvolvimento aproveitasse os benefícios da orientação a objetos.

A etapa de análise do DSOO consiste da identificação das entidades existentes no ambiente do software (Pressman, 2006). Primeiramente, um modelo é criado para ilustrar as suas funções básicas. Esse modelo representa, de forma simplificada, cenários de utilização do software (Larman, 2004). Algumas das entidades podem ser externas e apenas interagem com o software, enquanto que outras compõem partes do sistema, representando elementos ou eventos. As entidades relevantes ao software se tornam as classes da lógica do sistema e são agrupadas em um modelo em nível de análise.

Com as classes básicas identificadas, inicia-se a etapa de projeto. Nessa etapa, detalhes específicos das tecnologias utilizadas são acrescentados ao modelo de classes. Também são definidos no modelo de classes e em outros modelos a arquitetura do software, interfaces gráficas, pacotes, detalhes do funcionamento das operações, entre outras coisas. A etapa de implementação consiste da construção do que foi definido na fase de projeto. Algumas classes serão construídas a partir do zero, outras farão uso de composição e/ou herança e outras poderão ser reutilizadas da biblioteca da linguagem de programação escolhida (Larman, 2004).

Na etapa de testes ocorre a verificação se as funcionalidades foram implementadas corretamente (Myers *et al.*, 2004). Teste é um elemento crítico da garantia da qualidade de software e representa a revisão final da especificação, projeto e construção de código (Pressman, 2006). Normalmente, testes são aplicados em unidades independentes do código de um software como, por exemplo, em uma operação. Em seguida, testes de integração são executados para verificar a coerência da funcionalidade do software.

A manutenção representa a etapa em que o software é modificado com o intuito de corrigir defeitos (corretiva), realizar adaptações (adaptativa), aprimorar o projeto do sistema (preventiva) e introduzir novas funções (perfectiva). O paradigma orientado a objetos facilita o trabalho de manutenção em relação ao paradigma imperativo,

principalmente, por separar as entidades que compõem o software e agrupar as operações de acordo com as responsabilidades de cada classe (Pressman, 2006).

A depender do método utilizado para execução do processo de desenvolvimento, essas etapas podem ocorrer sequencialmente ou iterativamente. Existem técnicas e práticas que podem alterar a forma e a seqüência das etapas do desenvolvimento de software, em busca da redução do esforço e do aumento da qualidade. Algumas dessas técnicas e praticas serão apresentadas nas Seções 2.4 a 2.9.

2.3. Desenvolvimento de Software Voltado para a Web

Inicialmente, as empresas utilizavam a web apenas como um ambiente de marketing e informativo para clientes e parceiros. Contudo, percebeu-se que a web também poderia servir como um meio pelo qual as empresas pudessem prestar seus serviços remotamente e disponibilizar suas informações de forma automática e instantânea. Para tornar essas operações possíveis, a tecnologia empregada na construção de sites evoluiu de um panorama estático, em que havia apenas textos, imagens e vídeos, para um dinâmico, que permite, entre outras coisas, a interação, modificação e personalização por parte de quem o acessa. Essa mudança de panorama permitiu que os sistemas de informação fossem inseridos no ambiente da web.

O desenvolvimento de software voltado para a web possui as mesmas etapas do desenvolvimento de software orientado a objetos. A diferença está nas tecnologias empregadas. Sistemas de software web mais complexos mantêm as camadas de persistência e de negócio (modelo) implementadas em tecnologia não web. Somente as camadas de interface gráfica com o usuário (visão) e de controle, que realiza a intermediação entre o modelo e a visão, utilizam as tecnologias web. Esta Seção trata das tecnologias de desenvolvimento para a web utilizadas neste trabalho: *Hyper Text Markup Language* (HTML), *Java Server Pages* (JSP), *Servlets*, *JavaScript*, *Asynchronous Javascript And XML* (AJAX), *Cascading Style Sheets* (CSS), *eXtensible Markup Language* (XML) e *eXtensible Stylesheets Language* (XSL).

2.3.1. Hyper Text Markup Language

A principal linguagem para a construção de sites é *Hyper Text Markup Language* (HTML) (W3C, 2009). Uma página HTML é um arquivo de texto que contém um conjunto de elementos que representam as estruturas que devem ser apresentadas ao

usuário pelo navegador. Por exemplo, o navegador tem como entender que deve apresentar para o usuário o conteúdo de um elemento *table* no interior uma tabela na página que está sendo visualizada pelo usuário. Os elementos HTML são pré-definidos e o desenvolvedor não pode criar os seus próprios elementos.

Usado isoladamente, HTML permite somente a criação de páginas web estáticas com conteúdo multimídia. Mas, por motivo de compatibilidade, os navegadores ainda fazem uso de HTML. A maioria das tecnologias de criação de sites dinâmicos faz uso de recursos existentes em linguagens de programação para gerar código HTML com base em informações passadas pelo usuário durante a navegação.

Uma nova versão da HTML, chamada *eXtended Hyper Text Markup Language* (XHTML), foi criada para que seus arquivos passem a respeitar a sintaxe XML. Dessa forma, o código das páginas web se torna mais simples para que possam ser acessados a partir de navegadores de dispositivos móveis, que possuem recursos computacionais limitados (W3C, 2009).

2.3.2. Java Server Pages e Servlets

Além de ser uma linguagem de programação de uso livre e portável em múltiplas plataformas, Java (Sun Microsystems, 2009a) fornece inúmeros meios para ser utilizada na construção de sistemas para a web. *Java Server Pages* (JSP) (Sun Microsystems, 2009b) é um código HTML que permite a inclusão de código Java, enquanto servlet (Sun Microsystems, 2009c) é exatamente o oposto. Ambos permitem a construção de páginas cujo conteúdo é montado e disponibilizado dinamicamente de acordo com informações passadas pelo usuário. Utilizando JSP ou servlet é possível realizar operações existentes em sistemas *stand-alane* como, por exemplo, leitura e escrita em uma base de dados.

Um servidor, por exemplo, o Apache Tomcat (The Apache Software Foundation, 2009a), é responsável por receber as requisições provenientes do usuário (via AJAX, por exemplo), executá-las e retornar uma resposta. Essa resposta pode estar em diferentes formatos, porém, os mais comuns são HTML, XML e texto. JSP e servlet são tecnologias equivalentes, ou seja, que é possível realizar com uma, também pode ser feito utilizando a outra. Contudo, JSP é mais recomendado quando a finalidade principal da página em construção é a apresentação de conteúdo, enquanto servlet é mais recomendado quando a finalidade principal é processar dados. Quando é carregado

na memória principal do computador, o servidor converte todas as páginas JSP em servlets, de modo que as requisições são executadas sempre utilizando servlets.

2.3.3. JavaScript

JavaScript (W3C, 2009) não deve ser confundida com a linguagem de programação Java (Sun Microsystems, 2009a). Apesar do nome e de semelhanças sintáticas, essas duas linguagens têm propósitos diferentes.

JavaScript é a mais conhecida linguagem de script, utilizada para adicionar interatividade às páginas web, por exemplo, a verificação e envio dos dados de um formulário a um servidor, a criação de efeitos visuais, o uso de componentes gráficos com mais recursos, entre outras coisas (W3C, 2009). Uma linguagem de script é utilizada para estender a funcionalidade de um programa e/ou controlá-lo (Asleson e Schutta, 2006). No caso do JavaScript, o programa que tem sua funcionalidade estendida é o navegador. Como a maioria das linguagens de script, JavaScript é interpretada e atipada, o que a torna mais leve e flexível.

JavaScript permite manipular o *Document Object Model* (DOM), que representa a estrutura (elementos HTML) de uma página web na forma de um conjunto de objetos (Crane *et al.*, 2006). Com isso, é possível acrescentar, remover, acessar e modificar os valores dos elementos HTML, além de alterar o CSS que define seu leiaute.

2.3.4. Asynchronous Javascript And XML

Asynchronous Javascript And XML (AJAX) (Crane *et al.*, 2006) permite manipular o DOM para redesenhar partes de uma página web dinamicamente e aumentar sua interatividade com o usuário. Apesar do nome, envolve a combinação de outras tecnologias além de JavaScript e XML como, por exemplo, HTML, CSS, JSP e servlet. Com o uso de AJAX, o envio de uma requisição para um servidor ocorre de forma assíncrona, garantindo-lhe o tratamento de eventos que antes eram possíveis apenas em aplicações *stand-alone*. Por exemplo, considere um campo destinado ao preenchimento de nomes de clientes que está relacionado a um evento de autocompletar. Esse evento permite que, a cada letra que o usuário digita, uma busca na base de dados seja realizada para sugerir o nome completo do cliente desejado. Sem AJAX, o usuário seria impedido de digitar a próxima letra até que a busca na base de dados fosse completada. Isso certamente tornaria o evento de autocompletar lento, irritante e inútil.

Com o uso de AJAX são obtidas as seguintes vantagens (Asleson e Schutta, 2006): redução do tamanho das requisições, pois é possível solicitar apenas parte de uma página web; requisições mais rápidas por possuírem tamanho reduzido; aumento da interatividade com o usuário com a eliminação das interrupções; maior segurança, pois as validações podem ser realizadas no servidor; interfaces mais sofisticadas com a possibilidade de múltiplas ações assíncronas.

2.3.4. Cascading Style Sheets

HTML foi originalmente criada para definir o conteúdo e a apresentação das páginas web. Com a evolução de suas versões, além de definir a estrutura da apresentação, seus elementos passaram a incluir muitos atributos relacionados com o leiaute dessa apresentação. Para tornar a linguagem HTML melhor organizada, a W3C passou a recomendar que seus elementos definissem apenas estrutura da apresentação. Assim, foi criado um novo tipo de documento, chamado *Cascading Style Sheets* (CSS) (W3C, 2009), que é responsável pelo leiaute da apresentação. Exemplificando, o documento HTML indica que determinado texto ser apresentado dentro de uma tabela, enquanto o CSS indica detalhes sobre a cor e o estilo da borda dessa tabela e a fonte do texto.

A palavra *cascading* (em português, cascata) refere-se ao fato de que uma definição em uma estrutura pode ser herdada por todas as estruturas filhas. Por exemplo, a definição de um tipo de fonte para uma tabela também é válida para as suas linhas, suas colunas e outros elementos nela inseridos. Isso simplifica o documento CSS e reduz o esforço necessário para criá-lo.

Uma das vantagens do uso do CSS é a possibilidade de alterar completamente a aparência de todas as páginas de um site web por meio de um único documento CSS, ao invés de inúmeros documentos HTML.

2.3.5. EXtensible Markup Language

Assim como HTML, *eXtensible Markup Language* (XML) (W3C, 2009) é uma linguagem de marcação. Porém, XML foi projetada para estruturar, armazenar e transportar informação. Os elementos XML não são pré-definidos, portanto, o desenvolvedor deve definir seus próprios elementos. Essa característica garante aos documentos XML uma extensa variedade de maneiras em que podem ser utilizados. O

que resulta no surgimento de tipos de documentos XML com finalidades específicas como, por exemplo, XHTML e XSL.

Por ser um arquivo de texto, XML possui as seguintes vantagens (W3C, 2009): é independente de plataforma; pode ser facilmente manipulado por qualquer linguagem de programação ou de script; serve como um meio de compartilhamento entre software/hardware distintos; reduz a complexidade do transporte de informações em uma rede; simplifica a atualização de software/hardware mantendo as informações compatíveis.

2.3.6. EXtensible Stylesheets Language

HTML possui elementos pré-definidos e com significado conhecido, de maneira que o navegador sabe como apresentá-los e como formatar o leiaute desses elementos por meio de CSS. Mas, como os elementos XML não são pré-definidos, seus significados não são conhecidos pelo navegador que, assim, não sabe como apresentá-los. A finalidade da *eXtensible Stylesheets Language* (XSL) (W3C, 2009) é descrever como os elementos de um documento XML devem ser apresentados.

XSL se divide em três linguagens (W3C, 2009): 1) *eXtensible Stylesheets Language Transformations* (XSLT), que é utilizada para transformar documentos XML em outros tipos de arquivos de texto; *XPath*, que possibilita a navegação em elementos a atributos de documentos XML; e *eXtensible Stylesheets Language Formating Objects* (XSL-FO), que permite a formatação de documentos XML para dispositivos de saída (monitor, impressora, etc). Neste trabalho, somente XSLT e XPath foram utilizados.

XSLT é considerada a parte mais importante de XSL (W3C, 2009). A transformação de um documento XML é feita por meio de um *parser* que extrai suas informações por meio de instruções XPath. Essas informações são mescladas com o conteúdo de um documento XSL, que funciona como um gabarito, para dar origem a um novo arquivo. A maioria dos navegadores possui esse tipo de *parser*, assim como muitas linguagens de programação fornecem bibliotecas para a manipulação de XSL.

Um documento XML pode conter uma instrução que o associe a um documento XSL e, desse modo, ser apresentado por um navegador como se fosse um documento HTML, por exemplo. Um gerador de aplicações também pode utilizar documentos XSL como gabaritos para geração dos artefatos de uma aplicação.

2.4. Desenvolvimento de Software em Linhas de Produtos

O desenvolvimento de software tradicional consiste da construção de um único produto. Entretanto, as empresas desenvolvedoras de software tendem a produzir softwares que possuem mais semelhanças do que diferenças. Com base na idéia de reúso e no princípio das linhas de montagem da indústria, formou-se o conceito de Linhas de Produto de Software (LPS) (Jensen, 2009) (Clements e Northrop, 2001). Em uma linha de produtos, todos os softwares desenvolvidos pertencem ao mesmo domínio. Um domínio, ou família de softwares, representa um conjunto de softwares que possuem características comuns e outras variáveis (Weiss e Lai, 1999). Os produtos de uma LPS não são desenvolvidos a partir do zero. Cada novo software é resultado da combinação de artefatos previamente desenvolvidos e testados que tornam seu desenvolvimento mais eficiente e sua qualidade melhorada (Sugumaran *et al.*, 2006).

O desenvolvimento de software em uma linha de produtos é dividido em dois processos (Weiss e Lai, 1999) (Gomaa, 2005): engenharia do domínio e engenharia da aplicação. A engenharia do domínio refere-se à construção do núcleo de artefatos que representam as características de uma família de softwares. Seu modelo mais comum compreende quatro etapas: análise, projeto, implementação e testes. Na etapa de análise são identificadas as características dos membros do domínio escolhido. Essas características podem ser documentadas usando uma Linguagem de Modelagem da Aplicação (LMA), que representa as abstrações das aplicações e também é uma forma de documentação das especificações do domínio (Weiss e Lai, 1999). A etapa de projeto define quais tecnologias serão empregadas na construção dos artefatos e a forma pela qual eles poderão ser reutilizados para o desenvolvimento das aplicações. Na implementação são construídos os artefatos do domínio que, em seguida, passam pela etapa de testes. Durante o processo de engenharia de domínio também podem ser desenvolvidas ferramentas específicas que apóiam os processos de engenharia da aplicação (Weiss e Lai, 1999).

A LMA criada na etapa de análise da engenharia do domínio serve como um modelo de definição do domínio. Pode ser feita na forma de um formulário, de um diagrama ou de um documento XML (Shimabukuro Junior, 2006). Durante a engenharia da aplicação, uma instância da LMA do domínio é criada para armazenar os dados específicos da aplicação que está sendo gerada.

O processo de engenharia da aplicação se assemelha com o processo de desenvolvimento de software tradicional, porém, com o uso dos artefatos construídos na engenharia do domínio. Inicia com uma etapa de análise dos requisitos e identificação dos artefatos que representam as características variáveis necessárias ao novo software. É possível que a aplicação necessite de um artefato que não foi previamente construído. Na etapa de projeto, cria-se o modelo que interliga os artefatos selecionados na etapa de análise, inclusive os artefatos novos, se houverem. A etapa de implementação consiste na montagem do software com a interligação da arquitetura básica com os artefatos das características variáveis. Os artefatos que não foram previamente construídos precisam ser desenvolvidos e interligados aos demais. Esses artefatos podem ser adicionados ao núcleo de artefatos do domínio para serem utilizados em processos de engenharia da aplicação que ocorrerem posteriormente. Na etapa testes, apenas se houverem artefatos novos é que são realizados testes de unidade. Testes de integração precisam ser executados para verificar se os artefatos foram interligados corretamente e validar a funcionalidade do produto final. A Figura 2.1 apresenta o fluxo de execução dos processos de engenharia do domínio e da aplicação de uma LPS.

Figura 2.1. Fluxo dos processos de engenharia do domínio e da aplicação de uma LPS.

Manutenções no produto de uma LPS são realizadas com o acréscimo ou remoção de artefatos do núcleo do domínio. Assim, como ocorre no desenvolvimento da aplicação, novas características podem ser identificadas e transformadas em novos artefatos para o núcleo do domínio.

O desenvolvimento de uma linha de produtos de software exige um esforço inicial elevado para a realização da etapa de engenharia de domínio. Esse esforço só é compensado após o desenvolvimento de alguns softwares. A Figura 2.2 ilustra a comparação entre os esforços de um processo de desenvolvimento de software tradicional e de uma LPS.

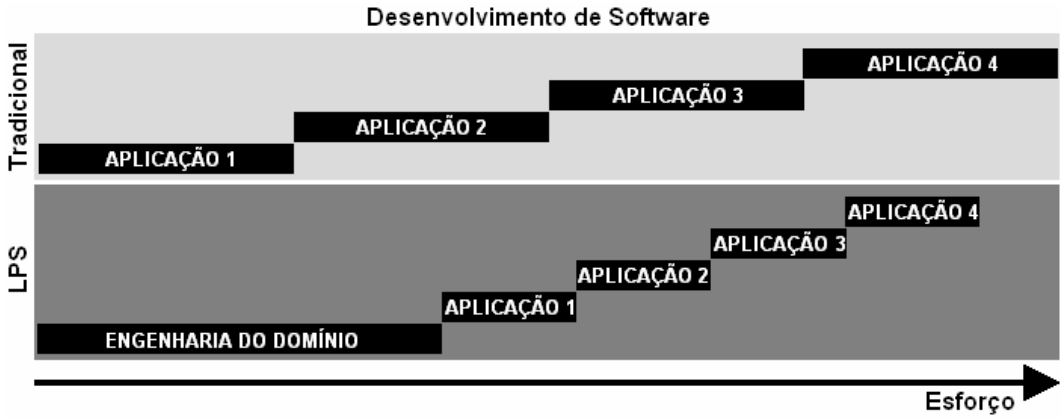

Figura 2.2. Gráfico de esforço do desenvolvimento de software tradicional e de uma LPS.

2.5. Desenvolvimento de Software com o Apoio de Padrões e de Linguagens de Padrões

Muitos dos problemas enfrentados por um desenvolvedor de software são conhecidos e já foram solucionados por outros desenvolvedores. Além disso, esses problemas tendem a ocorrer diversas vezes, de maneira que a mesma solução genérica possa ser reaplicada. Para evitar o desperdício de tempo e garantir a aplicação da solução mais adequada, os problemas e suas soluções começaram a ser catalogadas e passaram a ser conhecidos como padrões de software, com base na idéia definida por Christopher Alexander (arquitetura civil) de reutilizar soluções para problemas semelhantes (Alexander, 1997, *apud* Gamma *et al.*, 1995).

Um padrão de software constitui uma solução de sucesso para determinado problema em um contexto específico, descreve os esforços despendidos e os benefícios obtidos ao aplicá-lo (Gamma *et al.*, 1995). Técnicas da programação orientada a objetos, como herança e composição, fornecem apenas reúso de código. Padrões de software estão em um nível maior de abstração, pois fornecem reúso de experiência (Fowler, 1997).

Os padrões de software se tornaram um vocabulário comum entre os desenvolvedores (Manolescu *et al.*, 2007). É possível compreender o projeto de um software pela menção de quais padrões foram aplicados em sua construção (Johnson, 1997a). O que possibilita o entendimento e a manutenção de software mais eficiente.

Apesar dos padrões de projeto serem os mais conhecidos, existem padrões que atendem a outros propósitos, tais como: de análise, de arquitetura, de implementação (idiomas), de reengenharia, de processo, entre outros (Shalloway e Trott, 2001) (Kircher

e Völter, 2007). Os padrões de análise, de projeto, de arquitetura[1] e de implementação estão intimamente relacionados com as etapas de desenvolvimento de software e servem de apoio à realização de suas tarefas. Por exemplo, um padrão de análise pode indicar quais classes devem ser criadas na construção de uma lógica de negócio, enquanto que um padrão de projeto pode indicar a melhor forma de tratar a persistência das informações contidas nas classes.

Alguns padrões podem ser complementares, de modo que os desenvolvedores os utilizam em conjunto para resolver um único problema. Há casos que se tornaram tão comuns que subconjuntos de padrões passaram a ser conhecidos como um único padrão composto (Bushmann *et al.*, 2007). Por exemplo, o padrão composto *Batch Iterator* reúne dois padrões complementares, *Batch Method* e *Iterator*, para permitir o acesso remoto a elementos agregados, como listas e vetores. Outro exemplo muito comum de padrão composto é o Model-View-Controller (Freeman *et al.*, 2004), que trata da separação entre a lógica do negócio e a interface com o usuário de um software e é formado pelos padrões de projeto *Observer*, *Composite* e *Strategy*.

Quando um conjunto de padrões mantém uma relação de dependência, de modo que passam atender a um propósito maior, obtém-se uma linguagem de padrões (LP) (Coplien, 1996). Há duas diferenças básicas entre uma linguagem de padrões e padrões compostos: uma linguagem de padrões pertence a um domínio específico de software e seus padrões são fortemente relacionados a ponto de não existirem de maneira independente (Bushmann *et al.*, 2007). Os padrões constituintes de uma linguagem de padrões devem cobrir todas as peculiaridades de um dado domínio (Santos, 2004).

Enquanto um padrão resolve apenas um problema isolado em um software, as linguagens de padrões são aplicadas ao desenvolvimento de um software pertencente a um domínio específico (Coplien, 1996). Os padrões de uma linguagem de padrões compõem um grafo que indica a seqüência na qual devem ser aplicados (Brugali e Sycara, 2000). Cada padrão cria o contexto para o próximo padrão a ser utilizado e, dessa forma, uma linguagem de padrões representa a seqüência temporal de decisões que guiam o processo de desenvolvimento de software (Braga, 2002). Podem existir

[1] A arquitetura é definida na etapa de projeto, por isso alguns autores também consideram os padrões de arquitetura como padrões de projeto (Coplien, 1996).

padrões opcionais e, com isso, mais de uma seqüência de aplicação dos padrões. Também é possível que um ou mais padrões sejam aplicados mais de uma vez.

O desenvolvimento de software com o apoio de uma linguagem de padrões de análise facilita as tarefas dessa etapa com a aplicação dos padrões que correspondem aos requisitos do software. Como resultado, obtém-se o modelo de classes completo, exceto se houver requisitos que não estão relacionados com nenhum padrão da linguagem.

2.6. Desenvolvimento de Software com o Apoio de Frameworks

Um framework orientado a objetos é um projeto reutilizável de todo ou parte de um sistema representado por um conjunto de classes abstratas e pelo modo como suas instâncias se relacionam. Outra definição afirma que um framework é um esqueleto de uma aplicação que pode ser customizado por um desenvolvedor. Essas definições não são conflitantes: a primeira descreve a estrutura de um framework e a segunda enfoca o seu propósito (Johnson, 1997a).

Frameworks possuem a vantagem de serem poderosos e flexíveis e a desvantagem de serem dependentes da linguagem de programação em que foram implementados e difíceis de usar. Porém, à medida que se ganha experiência com o uso de um framework, essa dificuldade é amenizada.

Em uma comparação entre conceitos e técnicas, frameworks são constituídos de um conjunto de classes, assim como uma biblioteca de classes, porém as classes de um framework são dependentes entre si e não podem ser utilizadas separadamente como as classes de uma biblioteca (Johnson, 1997b) (Liem e Nugroho, 2008). Isso se deve, pois, além de reúso de código, frameworks também fornecem reúso de projeto. Uma outra diferença existente entre biblioteca de classes e frameworks é quanto à ordem de controle do software. O código específico da aplicação é que determina quando e como as classes de uma biblioteca são utilizadas. Contudo, quando um framework é utilizado, ocorre uma inversão de controle, pois é o framework que invoca o código específico da aplicação. Essa inversão de controle é conhecida como Princípio de Hollywood (Larman, 2004).

Todo framework é composto de duas partes básicas (Brugali e Sycara, 2000): os pontos fixos (*frozen spots*), que constituem a parte imutável, independentemente da

aplicação que esteja sendo desenvolvida, e os pontos variáveis (*hot spots*), que representam a parte utilizada pelo desenvolvedor para instanciá-lo de acordo com as especificações da aplicação desejada.

Um framework no qual o acesso aos pontos variáveis ocorre por meio da herança de suas classes é classificado como caixa branca. Quando esse acesso se dá por meio de composição, é classificado como caixa preta. Um framework que pode ser instanciado pelas duas maneiras é classificado como caixa cinza. Os mais fáceis de construir e mais flexíveis são os caixa branca, mas também são mais difíceis de usar, pois exigem maior conhecimento do desenvolvedor sobre suas classes. Já com os frameworks caixa preta ocorre exatamente o inverso (Johnson, 1997b).

Fayad e Schmidt (1997) afirmam que frameworks também são classificados conforme seu escopo em três tipos: 1) *System Infrastructure Frameworks* (SIF), 2) *Middleware Integration Frameworks* (MIF) e 3) *Enterprise Application Frameworks* (EAF). SIF simplificam o desenvolvimento de software de sistemas como, por exemplo, sistemas operacionais, interfaces com o usuário e compiladores. MIF aumentam a capacidade de modularização, reúso, extensão e integração de aplicações distribuídas e componentes. *Struts* (The Apache Software Foundation, 2009b) e *Spring* (Spring Source, 2009) são exemplos de MIF. EAF abordam sistemas de domínio específico voltado a atividades de indústria ou de prestação de serviços como, por exemplo, sistema para aeroportos, manufatura, gestão de negócios, finanças, entre outros (Abi-Antoun, 2007). O framework GRENJ, apresentado na Seção 3.3, é um exemplo de EAF.

Por pertencer a um domínio específico, um EAF exige trabalho de engenharia de domínio para identificar as características comuns e variantes dos membros (softwares) pertencentes ao domínio em questão. Isso implica em um esforço inicial antes de qualquer software vir a ser desenvolvido (Weiss e Lai, 1999) (Pacios, 2006). Abi-Antoun (2007) relata que o desenvolvimento de um framework é mais complexo do que o desenvolvimento de uma aplicação porque é difícil identificar corretamente suas abstrações. Por abstrações compreendem-se modelos do domínio, casos de uso, arquitetura, interfaces, entre outras coisas.

No desenvolvimento de software com o apoio de um framework, o reúso não se resume ao código das classes. O projeto e a implementação da arquitetura, da persistência das informações, da interface gráfica e de sua ligação com as classes da camada de negócio, entre outras coisas, podem ser fornecidos pelo framework. Portanto,

nessas etapas, o desenvolvedor necessita somente adaptar os requisitos específicos do software que está sendo construído. Os detalhes dessa adaptação dependem da forma de acesso aos pontos variáveis do framework: caixa branca, preta ou cinza. As etapas de testes e manutenção também são afetadas. As classes do framework já foram previamente testadas, o que reduz a existência de defeitos e o número de testes necessários. A manutenção do software é facilitada de forma semelhante à etapa de implementação. A adaptação de novos requisitos representa a utilização de outros pontos variáveis, caso esses requisitos sejam previstos pelo framework.

2.7. Desenvolvimento de Software com o Apoio de Frameworks Construídos com Base em Linguagens de Padrões de Análise

O uso de frameworks como apoio ao processo de desenvolvimento de software oferece vantagens devido ao reúso de projeto e de código. Contudo, problemas podem ocorrer e, consequentemente, essas vantagens não são obtidas e o produto final é invalidado. Um desses problemas é a identificação incorreta das classes que compõem o software, uma vez que o framework não oferece apoio à etapa de análise. Além disso, quanto maior é o número de variabilidades existentes no framework, maior é o risco de se cometer erros durante sua instanciação. Kirk *et al.* (2002) realizaram uma pesquisa que apontou as quatro maiores dificuldades encontradas quando se instancia um framework:

1. **Mapeamento** – expressar corretamente o problema utilizando os elementos do framework.
2. **Compreensão da funcionalidade** – decifrar o comportamento dos elementos do framework.
3. **Compreensão das interações** – entender como ocorre a comunicação entre os elementos do framework.
4. **Compreensão da arquitetura** – como os elementos podem ser utilizados e/ou modificados de forma que a integridade e as características não-funcionais do framework sejam preservadas.

Kirk *et al.* (2002) afirmam que esses problemas podem ser resolvidos por meio de uma linguagem de padrões que serve de documentação para um framework. Por outro lado, Brugali e Sycara (2000) afirmam que um framework é uma implementação de uma linguagem de padrões. As afirmações desses autores revelam que essas duas

técnicas se complementam e o ideal é que sejam utilizadas em conjunto, conforme ilustra a Figura 2.3.

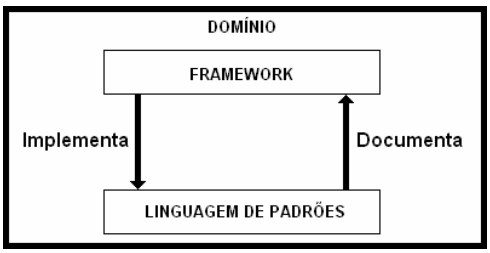

Figura 2.3. Relações existentes entre um framework e uma linguagem de padrões.

A linguagem de padrões GRN e o framework GRENJ (Durelli *et al.*, 2008), apresentados nas Seções 3.2 e 3.3, respectivamente, são um exemplo da ligação existente entre essas duas técnicas. A Tabela 2.1 contém outros exemplos de frameworks e linguagens de padrões que se complementam.

Tabela 2.1. Exemplos de frameworks originados a partir de linguagens de padrões.

Framework	**Linguagem de Padrão**	**Domínio**
G++ Application Framework	G++ Pattern Language	Manufatura integrada ao computador
JHotDraw	JHotDraw Pattern Language	Editores de imagem
Qd+	LV	Leilões virtuais

O *G++ Application Framework* e a linguagem de padrões *G++* atuam no domínio de sistemas distribuídos de manufatura integrada ao computador (Aarsten *et al.*, 1997). O framework *JHotDraw* é o resultado de um processo de reengenharia realizado por Kirk *et al.* (2002) sobre framework *HotDraw*, que mudou a linguagem de programação de Smalltalk para Java e também realizou adaptações na linguagem de padrões do framework original. O framework *Qd+* gera aplicações web e foi baseado na linguagem de padrões de Leilões Virtuais (LV) (Ré *et al.*, 2004), que possui dez padrões em nível de análise e atua no domínio de vendas efetuadas por leilões virtuais.

Um EAF construído com base em uma linguagem de padrões de análise pertence ao mesmo domínio dessa e fornece reúso na etapa de análise, além das de projeto e de implementação. A identificação das classes é apoiada pela linguagem de padrões e as variabilidades do framework correspondem às opções fornecidas na linguagem. Outra vantagem é que a identificação das classes segue o fluxo de padrões da linguagem, de modo que o problema é dividido em partes menores e torna-se mais fácil de ser resolvido. Como, normalmente, a linguagem de padrões oferece uma descrição textual e

exemplos de instanciação de cada padrão, torna-se mais fácil compreender a arquitetura do framework e como os seus elementos estão conectados. Assim, a ocorrência de erros durante a instanciação do framework é reduzida.

2.8. Desenvolvimento de Software com o Apoio de Geradores de Aplicações

Frameworks fornecem diversas vantagens no desenvolvimento de sistemas, mas ainda exigem do desenvolvedor conhecimento em programação, em particular sobre os conceitos e técnicas de uma linguagem orientada a objetos (Braga, 2002). Geradores de Aplicações (GA) traduzem especificações em software pronto para ser utilizado. As especificações são coletadas por meio de um diálogo interativo no qual o usuário insere as informações em uma série de menus em uma interface gráfica (Wu *et al.*, 2003). De posse dessas especificações, um gerador de aplicações cria uma ou mais aplicações de maneira automatizada sem a necessidade de uma ferramenta extra, além das já utilizadas por um software desenvolvido do modo convencional (Johnson, 1997a). Também pode ser considerado como um compilador para uma linguagem de um domínio específico (Smaragdaski e Batory, 1998, *apud* Pazin, 2004). Portanto, assim como uma linguagem de padrões e um EAF, um gerador de aplicações exige um trabalho de engenharia de domínio para ser construído.

Além da eficiência no desenvolvimento e a facilidade de uso, geradores de aplicações possuem outras vantagens, tais como redução de erros, manutenção facilitada das aplicações geradas, entrega rápida de protótipos ao usuário e garantia na padronização das interfaces das aplicações geradas. Mas também possuem desvantagens: impõem restrições às aplicações geradas, têm seu uso efetivo aplicável apenas a um número reduzido de casos e são difíceis de construir (Thibault e Consel, 1997) (Liem e Nugroho, 2008).

O uso de um gerador de aplicações altera a forma como são realizadas as etapas do desenvolvimento de software. O desenvolvedor se preocupa apenas com os requisitos. Não é necessário conhecimento sobre as classes que serão criadas, nem com nenhum outro detalhe interno do software gerado. Em outras palavras, o desenvolvedor define “o que” deve ser gerado, enquanto que o gerador define “como” ocorre essa geração. Também é esperado que as classes geradas estejam corretas e não necessitem de testes, apesar de ser possível realizá-los.

Normalmente, um gerador fornece flexibilidade somente dos requisitos funcionais. Detalhes sobre persistência, segurança, arquitetura tendem a ser fixos para todas as aplicações criadas a partir do mesmo gerador. Além do código, alguns geradores são capazes de gerar a documentação do software, o manual do usuário modelos, imagens e casos de testes (Cleaveland, 2001).

Podem existir requisitos que não sejam previstos pelo gerador utilizado e o desenvolvedor necessita acrescentá-los ao software sem o apoio do gerador de aplicações. Portanto, devem existir mecanismos que permitam ao gerador distinguir o código gerado do acrescentado ou alterado e, desse modo, um software pode passar por um processo de re-geração sem que as partes modificadas pelo desenvolvedor sejam perdidas (Shimabukuro Junior, 2006). Uma das formas de se realizar isso é por meio de zonas de segurança (Herrington, 2004). A delimitação dessas zonas é feita por comentários especiais para que o compilador da linguagem programação utilizada ignore tais marcações. Nesse caso, é necessário que o gerador leia o código gerado anteriormente, reconheça e mantenha o código inserido na zona de segurança.

Alguns geradores de aplicações fazem uso de um EAF internamente e, portanto, pertencem ao mesmo domínio. As especificações são utilizadas para realizar as customizações sobre os pontos variáveis do framework e instanciar uma aplicação. A diferença é que, com o gerador de aplicações, o usuário não precisa conhecer o framework, mas somente o domínio da aplicação (Braga e Masiero, 2004a). Se o EAF tiver sido construído com base em uma linguagem de padrões, os formulários do gerador de aplicações refletem as informações necessárias para a aplicação de seus padrões constituintes.

Um Gerador de Aplicações Configurável (GAC) é um tipo de gerador de aplicações que não possui um domínio específico (Shimabukuro Junior, 2006). Diversos domínios podem ser definidos por meio de um processo de engenharia do domínio. O modelo genérico desse processo possui as seguintes etapas:

1. Análise do domínio: as características comuns e as variáveis são identificadas e definidas em uma a LMA.
2. Projeto do domínio: são definidos os artefatos de software reutilizáveis que representam a parte invariável do domínio.

3. Implementação do domínio: os artefatos definidos na etapa de projeto são desenvolvidos e testados.
4. Configuração do gerador: são criados os gabaritos e os arquivos de configuração. As informações armazenadas em uma instância da LMA são mapeadas nos gabaritos para originarem aos artefatos específicos da aplicação que está sendo gerada. Os arquivos de configuração definem ações realizadas antes e depois da geração dos artefatos de uma aplicação e identificam quais gabaritos serão utilizados.

Com um domínio definido, o gerador de aplicações configurável passa a funcionar da mesma forma que um gerador de aplicações comum. O engenheiro de aplicações deve iniciar um novo projeto e selecionar o domínio da aplicação que deseja gerar. As informações preenchidas nos formulários do gerador são armazenadas em uma instância da LMA do domínio. Essa instância da LMA é utilizada pelos gabaritos para dar origem aos artefatos específicos da aplicação. Os arquivos de configuração interligam os artefatos gerados com os artefatos reutilizáveis e realizam as ações necessárias para completar a geração da aplicação.

2.9. Desenvolvimento Guiado por Testes

Durante o desenvolvimento ou a manutenção de um software, seu código pode ser implementado de maneira não apropriada, ocasionando resultados inesperados em sua execução. Para que o desenvolvedor possa detectar erros antes do software ser utilizado pelo usuário, testes devem ser realizados sobre seu código.

Uma forma de realizar testes pode ser por meio da execução do software e da verificação de seu funcionamento. Entretanto, essa abordagem não é aconselhável, pois pode ser tendenciosa e não cobrir todas as possíveis entradas, além de consumir maior tempo, porque o software precisa ser executado uma vez para cada entrada a ser testada. Com o uso de uma abordagem de testes automatizados, pode-se amenizar essas desvantagens. Testes automatizados podem ser escritos com o apoio de uma ferramenta de testes como, por exemplo, a família xUnit – que é composta de versões implementadas em diversas linguagens de programação (Janzen e Saiedian, 2005). Essa ferramenta permite ao desenvolvedor executar os testes no mesmo ambiente de desenvolvimento do sistema após a implementação de uma unidade e fornece uma comparação entre o resultado esperado e o obtido.

Em um modelo de desenvolvimento tradicional, os testes automatizados são criados após a construção das unidades de código (*test last*). Contudo, a criação de testes antes da implementação da unidade (*test first*) torna explícita a responsabilidade do código e simplifica o projeto do software (Beck, 2001). Essas vantagens podem ser obtidas com o Desenvolvimento Guiado por Testes (do inglês, *Test-Driven Development* - TDD) (Janzen e Saiedian, 2008).

TDD é uma prática baseada na criação de testes automatizados para a construção de pequenas unidades de um software, desenvolvidas em ciclos iterativos e evolucionários (Janzen e Saiedian, 2005). É uma das práticas mais importantes entre as preconizadas pelo XP (*eXtreme Programing*), mas também pode ser aplicado a outros modelos (Beck, 2001).

A Figura 2.4(a) ilustra um fluxo de desenvolvimento evolucionário tradicional com testes automatizados sendo criados após a codificação de cada unidade. A Figura 2.4(b) apresenta um fluxo com TDD em que os testes automatizados são criados antes da codificação de cada unidade.

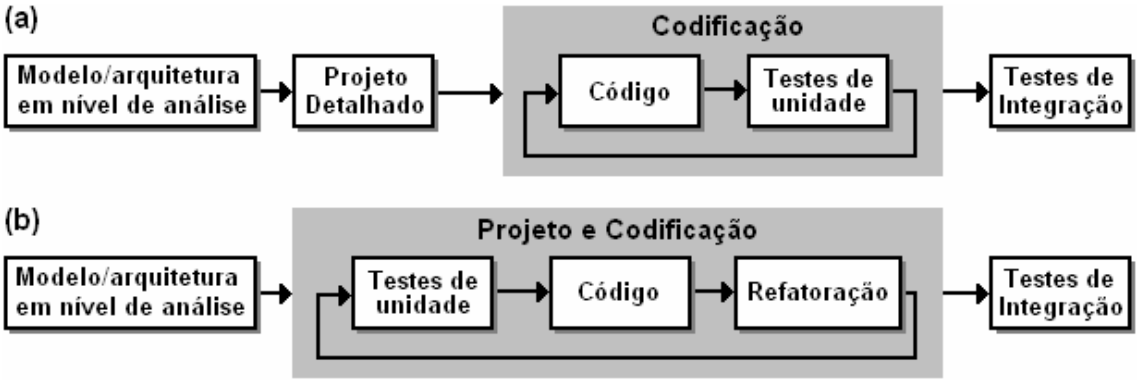

Figura 2.4. Fluxos de desenvolvimento (a) evolucionário tradicional e (b) TDD.

A aplicação de testes automatizados garante maior eficiência ao processo de desenvolvimento de software e maior qualidade ao produto final. Com a prática de TDD, a criação dos testes define detalhes sobre a implementação das unidades como escopo, interface, resultados esperados e tratamento de exceções (Beck, 2001). Portanto, TDD constitui uma prática para definição do projeto de software e não uma técnica para criação de testes automatizados antes da codificação (Janzen e Saiedian, 2008).

Como pode ser visto na Figura 2.4(a), o fluxo de desenvolvimento evolucionário tradicional inicia com a construção de um modelo em alto-nível que define a funcionalidade e a arquitetura do software. A etapa seguinte consiste da evolução desse

modelo para outro mais detalhado e dependente de tecnologia. Somente após a conclusão desse modelo detalhado é que as unidades de código começam a ser implementadas. Os testes são criados e executados depois da implementação da unidade correspondente. Por fim, as unidades são integradas e o software é testado como um todo (Janzen e Saiedian, 2008).

O fluxo do TDD, Figura 2.4(b), também inicia com a construção de um modelo em alto-nível, porém, nenhum modelo detalhado é construído. Ao invés disso, são definidos ciclos iterativos e evolucionários (Janzen e Saiedian, 2008). A Figura 2.5 mostra que cada ciclo TDD é composto dos seguintes passos (Williams *et al.*, 2003):

1. Crie uma lista de casos de teste para definir as situações (casos) que devem ser tratadas pela unidade.
2. Implemente os casos de teste da lista na mesma linguagem de programação do software e com o uso de uma ferramenta de testes como, por exemplo, o xUnit.
3. Implemente o código da unidade sem se preocupar com eficiência ou com legibilidade.
4. Execute os casos de teste para verificar o código implementado.
5. Em caso de falha, corrija o código e teste novamente até que seja aprovado.
6. Refatore o código da unidade para melhorar a qualidade do código da unidade, principalmente, em relação à legibilidade.
7. Re-execute os casos de teste para verificar que falhas não foram introduzidas depois do código ter sido refatorado.

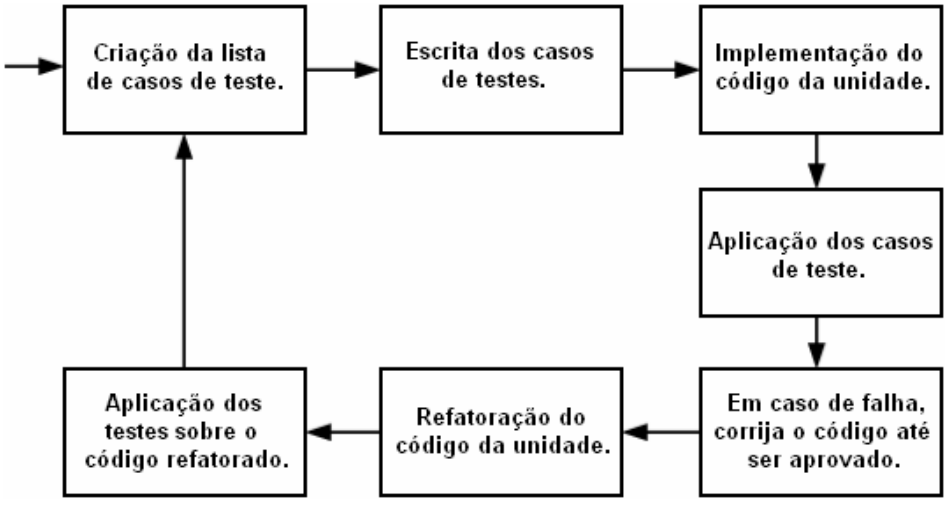

Figura 2.5. Fluxo do Ciclo TDD (Williams *et al.*, 2003).

Uma unidade de código é aprovada em um teste quando apresenta os resultados esperados. Porém, ainda pode ser inflexível e complexa. A refatoração é uma prática que muda a estrutura interna de um software sem alterar seu comportamento observável (externo), tornando-o mais fácil de ser entendido e modificado (Fowler, 1999). Além disso, a refatoração evita a deterioração do código de um sistema que passou por inúmeras alterações, facilita a detecção de erros, melhora a qualidade e aumenta a eficiência de desenvolvimento (Kerievsky, 2004).

A execução dos ciclos do TDD garante ao processo de desenvolvimento de software as seguintes vantagens: feedback rápido e contínuo ao desenvolvedor por meio de ciclos de baixa granularidade; detecção e remoção de erros tão logo o código é implementado; e redução do número de erros inseridos no código devido à divisão da implementação em pequenas partes testáveis (Williams *et al.*, 2003). Essas vantagens também são desejáveis quando se trata de manutenção de software.

2.10. Considerações Finais

Uma das vantagens principais que o desenvolvimento de software orientado a objetos apresentou com relação ao desenvolvimento estruturado foi facilitar a reutilização de entidades de código de forma mais eficiente, uma vez que encapsulam tanto dados quanto operações. Além de representarem mais claramente os objetos do mundo real, essas entidades tornaram os softwares mais flexíveis e melhor organizados. De forma semelhante, o desenvolvimento de software voltado para a web possui as mesmas vantagens. Pois, apesar das diferenças nos ambientes em que são aplicadas essas duas formas de desenvolvimento, suas tecnologias apresentam, em essência, os mesmos princípios.

Padrões, linguagens de padrões, frameworks e geradores de aplicações são técnicas que aperfeiçoam o desenvolvimento de software, seja orientado a objetos ou web. O ponto chave dessas técnicas é proporcionar outras formas de reúso, além do de código.

A Tabela 2.2 apresenta uma comparação quanto à forma de reúso, dependência de uma linguagem de programação, facilidade de uso, aplicabilidade e customização do software entre as técnicas apresentadas neste Capítulo. Com isso, obtém-se um resumo sobre as características que mostram suas principais semelhanças e diferenças.

Tabela 2.2. Comparação entre as principais técnicas que enfatizam o de reúso.

Técnica	Forma de Reúso	Dependente de Linguagem	Facilidade de uso	Aplicabilidade	Customização do software
Padrões	Experiência	Não	Média	Extensa	Extensa
Frameworks	Código e Projeto	Sim	Difícil	Extensa	Média
EAF + LP	Experiência, Código e Projeto	Sim	Difícil	Domínio	Média
GA	Código e Projeto	Sim	Fácil	Domínio	Pequena
GAC	Código e Projeto	Não	Média	Extensa	Pequena

Os padrões e frameworks documentados na década de 1990 estão relacionados com linguagens orientadas a objetos. Entretanto, essas técnicas já estão apoiando outras abordagens de desenvolvimento de software como, por exemplo, programação orientada a aspectos e desenvolvimento de software dirigido por modelos. A tendência é que novas tecnologias atinjam maturidade e aceitação maiores à medida que padrões, linguagens de padrões e frameworks baseados nelas sejam desenvolvidos.

Frameworks e geradores de aplicações são técnicas fortemente relacionadas com as linhas de produto de software. As características comuns dos softwares de uma mesma família podem ser implementadas como pontos fixos de um framework, enquanto que as variabilidades constituem os seus pontos variáveis. Por outro lado, um gerador de aplicações constitui uma linha de produtos de software automatizada. Seu desenvolvimento constitui um processo de engenharia de domínio. Quando utilizado para gerar uma aplicação, preenchimento de seus formulários representa a etapa de analise da aplicação, enquanto que a geração dos artefatos representa as etapas de projeto e de implementação da engenharia da aplicação.

Por conta da maior eficiência no desenvolvimento das aplicações, um gerador de aplicações também apóia o processo de prototipação. Pois novas aplicações podem ser geradas rapidamente apenas alterando as especificações fornecidas ao gerador, de modo que o usuário pode verificar qual produto lhe é mais adequado.

A prática de TDD propõe a divisão da implementação em ciclos evolucionários. Portanto, pode ser aplicada como modelo de processo no desenvolvimento de um EAF baseado em uma linguagem de padrões e de aplicações instanciadas desse framework, uma vez que os ciclos podem ser organizados de acordo com os padrões da linguagem

de padrões. Além disso, TDD proporciona outras vantagens como, por exemplo, aumento da qualidade do produto, simplificação do código implementado e rápido *feedback*.

O Capítulo 3 apresenta o framework GRENJ, que tem como base a linguagem de padrões Gestão de Recursos de Negócios. Também no Capítulo 3, é apresentado o gerador de aplicações configurável Captor, que foi utilizado para a construção de um wizard para o framework GRENJ.

3 Frameworks e Geradores de Aplicações

3.1. Considerações Iniciais

O desenvolvimento de software com o apoio de técnicas baseadas no conceito de reúso, como frameworks e geradores de aplicações, provêm economia de custos, aumento na produtividade e maior garantia da qualidade dos artefatos produzidos, conforme foi comentado no Capítulo 2. Devido a essas vantagens, o uso dessas técnicas tem se expandido para os mais variados domínios de aplicação.

Como existem muitas empresas no setor de prestação de serviços, diversos sistemas no domínio de gestão de recursos de negócios são repetitivamente desenvolvidos, o que ocasiona em intenso retrabalho. Na tentativa de amenizar esses problemas, foi criada a linguagem de padrões Gestão de Recursos de Negócios (GRN), que contempla aplicações nas quais seja necessário registrar transações de aluguel, comercialização e manutenção dos recursos de negócio (Braga *et al.*, 1999).

A GRN apóia o desenvolvimento de sistemas na etapa de análise, uma vez que auxilia o engenheiro da aplicação na identificação das classes que devem fazer parte do sistema em desenvolvimento. Assim, provê reúso da experiência acumulada sobre o domínio de gestão de recursos de negócios. Entretanto, a GRN não apóia as demais etapas do desenvolvimento de software. Por isso, foi construído o framework Gestão de REcursos de Negócios (GREN), implementado em Smalltalk (Smalltalk Dot Org, 2009) com base na GRN (Braga, 2002). Esse framework passou por um processo de reengenharia com uso de TDD que originou uma versão implementada em Java, denominada Gestão de REcursos de Negócios com implementação em Java (GRENJ) (Durelli, 2008). Um gerador de aplicações configurável pode ser utilizado com o GRENJ para facilitar a instanciação de sistemas de informação com base nesse framework e evitar o esforço necessário para o desenvolvimento de um novo gerador para o domínio da GRN.

A organização das demais Seções deste Capítulo é da seguinte forma: na Seção 3.2 é apresentada a linguagem de padrões GRN; na Seção 3.3 é apresentado o

framework GRENJ; na Seção 3.4 o gerador de aplicações configurável Captor é tratado; e na Seção 3.5 estão as considerações finais.

3.2. A Linguagem de Padrões GRN

A linguagem de padrões Gestão de Recursos de Negócios (GRN) foi construída para auxiliar o desenvolvimento de sistemas de informação no domínio de locação, comercialização e manutenção de recursos de negócios (Braga, 2002). Sua aplicação facilita a análise de um sistema desse domínio, principalmente para desenvolvedores menos experientes, sendo classificada como uma linguagem de padrões de análise.

A Figura 3.1 mostra o grafo do fluxo de aplicação dos padrões da GRN com a ordem em que eles podem ser aplicados. Alguns padrões podem ser seguidos por diferentes fluxos alternativos, de acordo com as requisições do sistema em desenvolvimento. O fluxo de aplicação também indica que alguns padrões são opcionais. Ao todo são quinze padrões de análise organizados em três grupos: Identificação do Recurso do Negócio, Transações de Negócio e Detalhes da Transação de Negócio (Braga *et al.*, 1999).

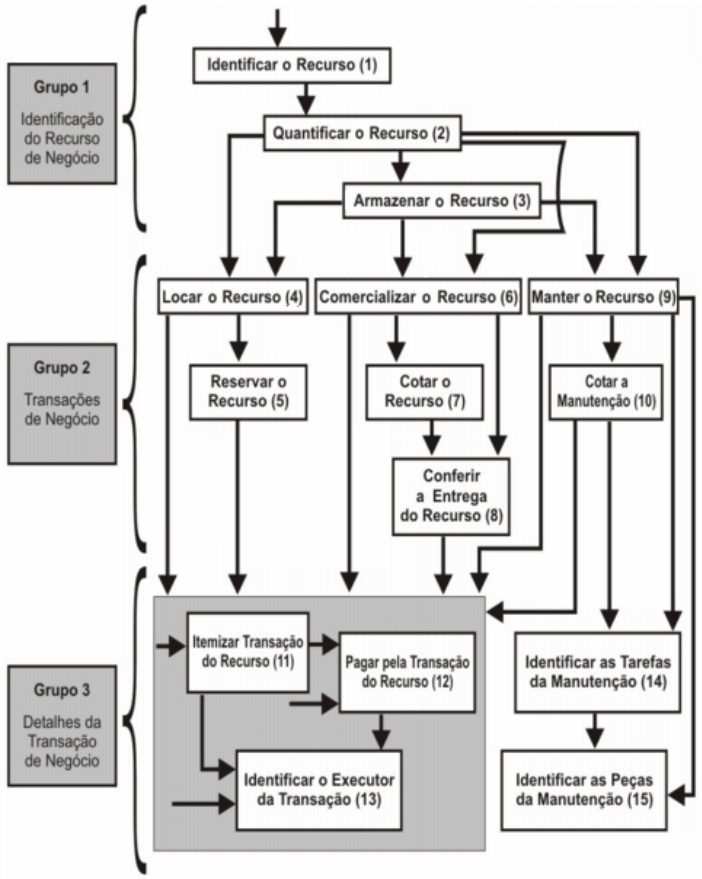

Figura 3.1. Grafo fluxo de aplicação dos padrões da GRN (Braga *et al.*, 1999).

O primeiro grupo de padrões da GRN tem como objetivo definir o recurso e a forma como ele deve ser tratado dentro do sistema. Três padrões compõem esse grupo na ordem que se segue:

1. **Identificar o Recurso**: fornece meios para definir o recurso e classificá-lo sobre diversas perspectivas.
2. **Quantificar o Recurso**: determina se o recurso é único (simples), instanciável, mensurável ou agrupado em lotes.
3. **Armazenar o Recurso**: padrão opcional que trata de informações relacionadas à forma e ao local de armazenamento do recurso.

No segundo grupo estão localizados os principais padrões da GRN, responsáveis pelas transações que serão realizadas pelo sistema, além de outros padrões que fornecem funções adicionais (Braga e Masiero, 2002). Segue-se a relação de padrões do segundo grupo da GRN enumerados de acordo com a sua ordem de aplicação e com os padrões principais destacados em itálico:

4. ***Locar o Recurso***: trata de transações relativas ao aluguel de recursos.
5. **Reservar o Recurso**: fornece a possibilidade de reserva de recursos para posterior locação.
6. ***Comercializar o Recurso***: aborda transações de venda de recursos.
7. **Cotar o Recurso**: determina o valor de um recurso para comercialização.
8. **Conferir a Entrega do Recurso**: trata de detalhes necessários para garantir o recebimento dos recursos de uma comercialização por parte do solicitante.
9. ***Manter o Recurso***: responsável pelas transações de manutenção de recursos.
10. **Cotar a Manutenção**: adiciona a função de determinação do valor de uma transação de manutenção com base nas informações da mesma.

O terceiro e último grupo é responsável por definir detalhes relevantes às transações de negócio. A Figura 3.1 mostra que os três primeiros padrões desse grupo são opcionais e aplicáveis a todos os tipos de transações abordados pela GRN, enquanto os dois últimos são aplicados somente para completar a transação de manutenção. Segue-se a descrição dos padrões do terceiro grupo:

11. **Itemizar Transação do Recurso**: define como tratar mais de um recurso por transação.
12. **Pagar pela Transação do Recurso**: permite gerenciar diferentes formas de pagamento para as transações.
13. **Identificar o Executor da Transação**: estabelece como manter informações sobre o usuário do sistema que executou uma transação.
14. **Identificar as Tarefas da Manutenção**: mantém informações sobre as tarefas necessárias para realizar a manutenção do recurso.
15. **Identificar as Peças da Manutenção**: mantém informações sobre as peças utilizadas em uma transação de manutenção de recursos.

Cada padrão da GRN possui uma descrição textual e é representado por um modelo de classes da UML, em nível de análise (Braga, 2002). Por meio deles o desenvolvedor faz reúso de experiência na elaboração do modelo de classes de uma aplicação específica.

O quarto padrão, *Locar o Recurso*, exemplifica os padrões da GRN por meio de dois modelos de classes: o modelo genérico exibe as classes do padrão e é mostrado na Figura 3.2 (a); e modelo da Figura 3.2 (b) exemplifica uma instância para um sistema de uma locadora de DVDs. Ressalta-se que a classe *Origem* é opcional e não está sendo utilizada neste exemplo.

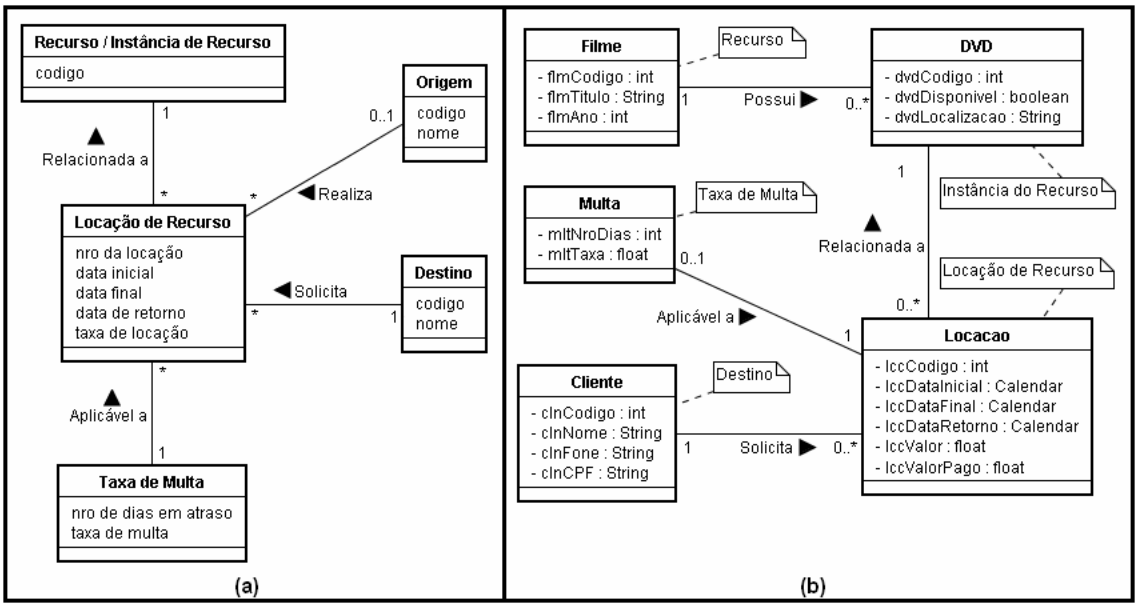

Figura 3.2. Diagramas de classe do quarto padrão da GRN, Locar o Recurso (Braga, 2002).

3.3. O Framework GRENJ

Como toda linguagem de padrões, a GRN fornece reúso de experiência por meio de seu modelo de classes em nível de análise. Para aumentar o nível de reúso obtido, o framework Gestão de REcursos de Negócios (GREN) foi construído com base nos padrões da GRN (Braga, 2002). Sua implementação foi realizada em linguagem Smalltalk (Smalltalk Dot Org, 2009) e com o uso do banco de dados MySQL (MySQL, 2009). Como a linguagem Smalltalk é mais amplamente utilizada no ambiente acadêmico, optou-se por realizar um processo de reengenharia para produzir uma nova versão desse framework, denominada Gestão de REcursos de Negócios com implementação em Java (GRENJ) (Durelli, 2008). O GRENJ é considerado um *Enterprise Application Framework* (EAF) por pertencer a um domínio específico voltado a atividade de prestação de serviços (Abi-Antoun, 2007).

O GRENJ é um framework caixa branca, ou seja, sua instanciação é feita por meio da extensão das classes (herança) que representam os seus pontos variáveis (*hot spots*) (Braga e Masiero, 2002). A identificação dos pontos variáveis é uma das maiores dificuldades quando se constrói um framework (Braga, 2002). No caso do GRENJ, essa identificação foi facilitada pelas descrições textuais e os diagramas de classe dos padrões existentes na GRN. As classes do GRENJ a serem estendidas correspondem às classes definidas pelos padrões da GRN (Braga e Masiero, 2004b).

O mapeamento das classes existentes nos modelos dos padrões da GRN para o framework GRENJ não é da ordem de um para um, ou seja, o número de classes existente no GRENJ é maior do que o apresentado nos diagramas de classe dos padrões da GRN (Braga e Masiero, 2002). Isso ocorre devido aos padrões da GRN serem em nível de análise, portanto uma série de hierarquias de classes foi originada na implementação dos pontos variáveis. O uso de padrões de projeto como, por exemplo, *Strategy*, *Factory Method* e *Template Method* (Gamma *et al.*, 1995) foi fundamental nesses casos, pois permitem a obtenção de um objeto da classe adequada para cada aplicação específica entre classes especificadas.

A arquitetura do GRENJ está dividida em duas camadas, como pode ser visto na Figura 3.3 (Durelli *et al.*, 2008). A Camada de Persistência é responsável pela leitura e escrita dos dados no banco de dados da aplicação. A Camada de Negócio cuida das

entidades relacionadas com a lógica das aplicações desenvolvidas e é constituída de uma série de hierarquias de classes baseadas nas definições dos padrões da GRN.

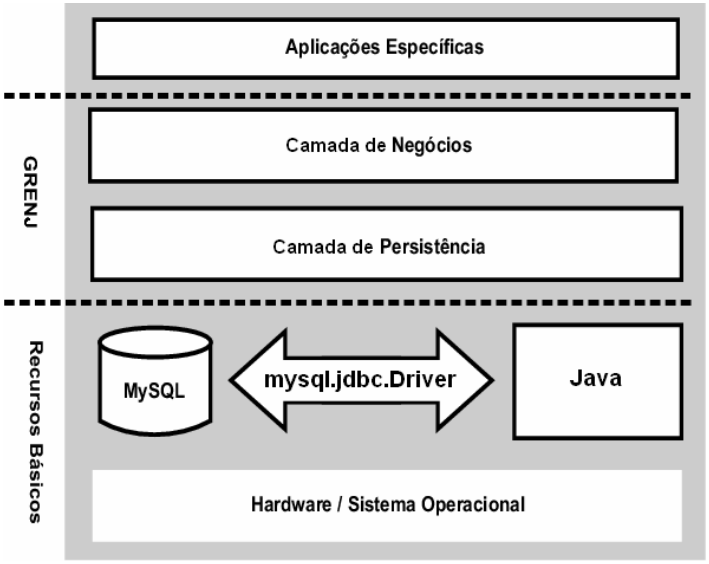

Figura 3.3. Arquitetura do framework GRENJ (Durelli *et al.*, 2008).

3.3.1. Instanciação de um Sistema com o apoio do Framework GRENJ

Para a construção de um sistema completo utilizando o GRENJ, um desenvolvedor instancia as duas camadas inferiores e deve implementar uma camada de interface gráfica com o usuário sem o apoio do framework. Para exemplificar a instanciação de um sistema de informação com o GRENJ, foi considerado um estudo de caso sobre uma locadora de DVDs, cujos requisitos são os listados na Tabela 3.1. A instanciação desse sistema contempla somente as camadas existentes no framework GRENJ, ou seja, as camadas de persistência e de negócio.

Tabela 3.1. Relação dos requisitos do sistema para uma locadora de DVDs.

#	**Descrição**
1	A locadora realiza o aluguel de DVDs de filmes que podem ter uma ou mais cópias.
2	Cada filme possui um código, título e ano.
3	Cada DVD possui código que identifica sua posição na prateleira, informação que indica se está disponível ou alugado e o título do filme nele contido.
4	Os filmes são classificados por categoria que indica o valor diário da locação.
5	Os filmes também são classificados por gênero (comédia, terror, ação, etc.).
6	Os DVDs são alugados para os clientes cadastrados da locadora. As informações que o sistema deve manter sobre o cliente são: código, nome, telefone e CPF.
7	As informações de locação são: código, data de locação, data de devolução prevista, código do cliente, DVDs alugados, data de devolução efetiva e valor. Um cliente pode alugar mais de um DVD em uma mesma locação.
8	Se os DVDs não forem devolvidos na data de devolução prevista, o cliente deve pagar multa correspondente a um valor fixo multiplicado pelos dias de atraso na devolução.

Para a utilização do GRENJ, um desenvolvedor precisa determinar a ordem de aplicação dos padrões da GRN e conhecer a arquitetura do framework, principalmente seus pontos variáveis a serem customizados. O GRENJ proporciona reúso caixa-branca e, portanto, subclasses são criadas com base nas classes que representam os pontos variáveis do framework. As customizações necessárias, tais como adição de novos atributos e métodos para implementar a funcionalidade pretendida, são introduzidas nessas subclasses criadas.

O primeiro padrão da GRN aborda como identificar o recurso e suas classificações de acordo com os requisitos do sistema em desenvolvimento (Braga *et al.*, 1999). Analisando os requisitos da Tabela 3.1, um único recurso foi identificado e definido pela classe *Filme*, relacionada com as classes *Categoria* e *Genero*, que representam classificações do recurso. No GRENJ a classe que representa o recurso é *Resource*. Essa classe já possui os atributos referentes ao código de identificação e ao nome (descrição) do recurso, herdados indiretamente da classe *StaticObject*. As classes *Categoria* e *Genero* estendem *SympleType*, que representa uma classificação simples sobre o recurso. Todas as classes estendem, direta ou indiretamente, a classe *PersistentObject*, que é responsável pela persistência das informações no banco de dados. A Figura 3.4 mostra o modelo de classes do GRENJ em conjunto com as classes do sistema da locadora de DVD, destacadas pela cor cinza, para o primeiro padrão da GRN, *Identificar o Recurso*.

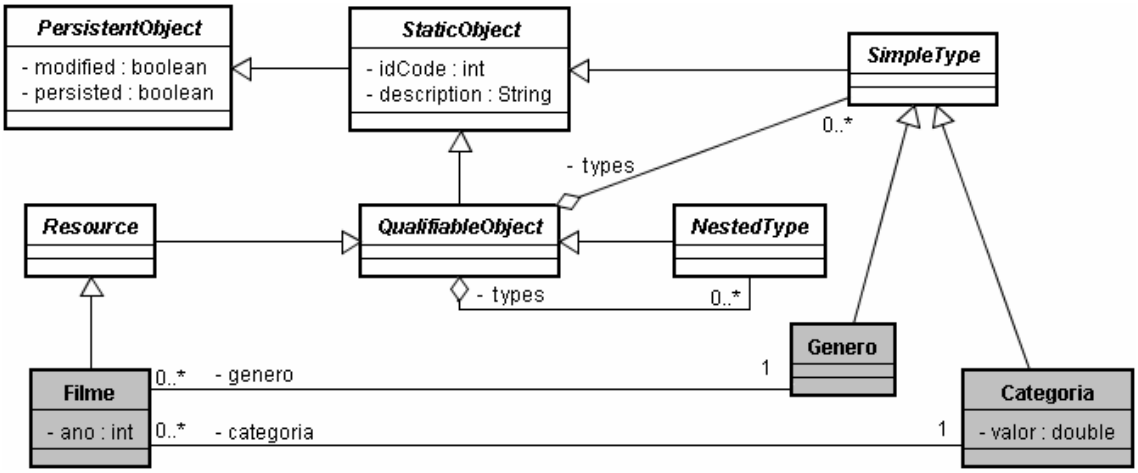

Figura 3.4. Classes do GRENJ para o primeiro padrão da GRN (Durelli *et al.*, 2008).

Na classe *Filme*, que estende a classe *Resource* do GRENJ, são acrescentados o atributo ano e os métodos correspondentes para acessá-lo. Os construtores e os métodos que configuram as operações de persistência no banco de dados precisam ser

sobrescritos para que tenham conhecimento do atributo adicionado, assim como os métodos que ligam o recurso às classes que o classificam, no caso *Categoria* e *Genero*.

A Figura 3.5 exibe parte do código da classe *Filme* com um dos métodos de configuração das instruções de persistência no banco de dados e os dois métodos que relacionam *Filme* com *Categoria* e *Genero*. No código correspondente às classes *Categoria* e *Genero*, são acrescentados métodos semelhantes aos da classe *Filme*, pois tanto *Resource* quanto *SimpleType* estendem a classe *QualifiableObject* do GRENJ.

```
public class Filme extends Resource {

   private int ano;

   public String updateSetClause() {
      StringBuilder updateClause = new StringBuilder( super.updateSetClause() );
      updateClause.append( ", ano = " + this.getAno() );
      return updateClause.toString();
   }

   public Class[] typeClasses() {
      return new Class[] { Categoria.class, Genero.class };
   }

   protected String[] typeFieldsInitialize() {
      return new String[] { "categoria", "genero" };
   }
}
```

Figura 3.5. Parte do código da classe *Filme* que estende a classe *Resource* do GRENJ.

A Figura 3.6 apresenta o modelo de classes do segundo padrão da GRN, *Quantificar o Recurso*, contendo as classes do framework GRENJ e a classe DVD, acrescentada na instanciação do sistema da locadora de DVDs. Tanto no GREN quanto no GRENJ o segundo padrão da GRN foi implementado com o uso do padrão de projeto *Strategy* (Gamma *et al.*, 1995), cujas classes envolvidas estão agrupadas em um retângulo na Figura 3.6. O padrão *Strategy* foi aplicado para que a classe *Resource* possa assumir o comportamento de um recurso simples, instanciável, mensurável ou em lote. Como para um *Filme* podem existir várias cópias, ele se caracteriza como um recurso instanciável. Portanto, a classe *DVD* foi criada como uma extensão de *ResourceInstance*, que possui os atributos e os métodos necessários para implementar a funcionalidade de recurso instanciável. O desenvolvedor só precisa acrescentar em *DVD* dois construtores e um método para recuperar um objeto da classe *Filme*. Também é preciso acrescentar na classe *Filme* um método para indicar que a classe DVD representa sua instância.

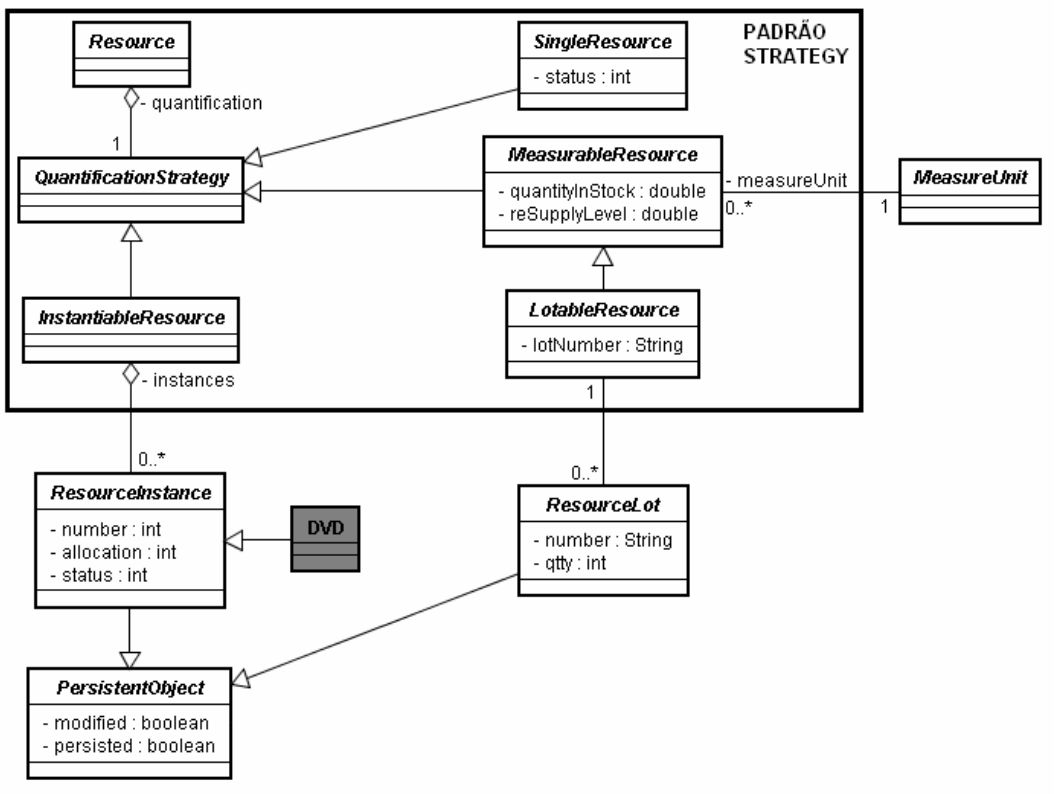

Figura 3.6. Classes do GRENJ para o segundo padrão da GRN (Durelli *et al.*, 2008).

A Figura 3.7 mostra o modelo de classes do framework GRENJ, referente ao segundo grupo de padrões da GRN, em conjunto com as classes implementadas para o sistema da locadora de DVDs, destacadas na cor cinza. O segundo grupo de padrões da GRN é responsável pelas transações do negócio (Braga *et al.*, 1999). Como os requisitos do sistema indicam uma transação de aluguel de DVDs, o padrão *Locar o Recurso* foi aplicado, acarretando na implementação das classes *Locacao, Cliente* e *TaxaDeMulta*. Neste estudo de caso, não é necessário implementar uma classe para a função de origem da transação, pois os recursos são provenientes da própria locadora. Assim não foi necessário criar uma extensão da classe *SourceParty*. *Locacao* estende a classe *ResourceRental* e representa a transação de mesmo nome realizada toda vez que clientes realizam a locação de um DVD. A classe *Cliente* é uma extensão de *DestinationParty* e representa os interessados nas transações. *TaxaDeMulta* estende *FineRate* e trata das tarifas cobradas quando ocorrem atrasos na devolução dos DVDs de uma locação. Por motivos de simplificação, somente as classes *ResourceReservation*, *BasicMaintenance* e *BasicNegotiation* dos demais padrões do segundo grupo da GRN aparecem na Figura 3.7. Essas classes não são utilizadas pelo sistema da locadora de DVDs.

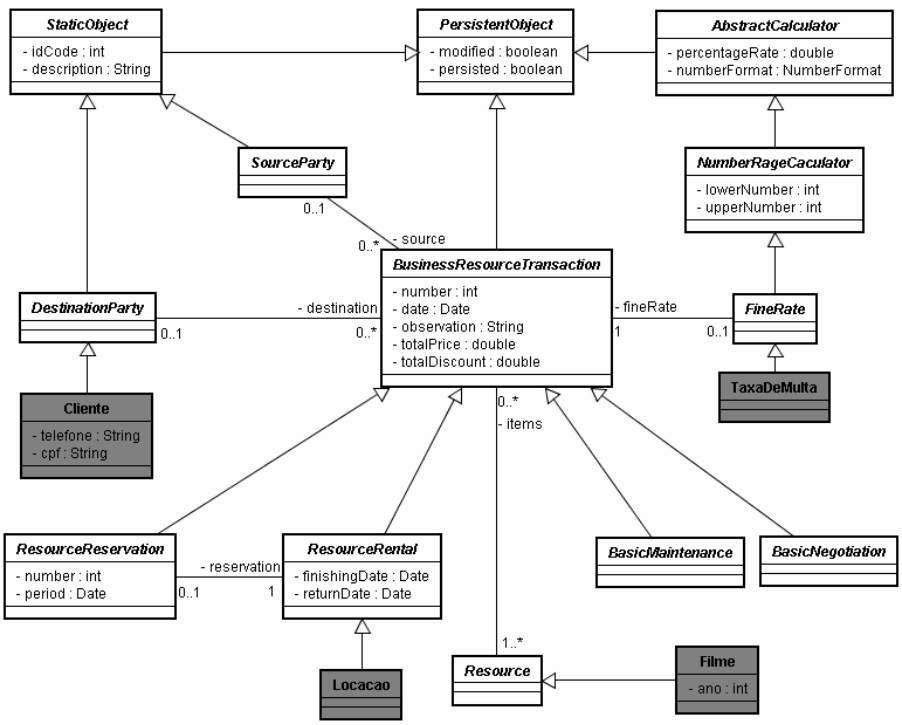

Figura 3.7. Classes do framework GRENJ e do sistema para uma locadora de DVDs relativas ao segundo grupo de padrões da GRN (Durelli *et al.*, 2008).

A Figura 3.8 mostra os métodos da classe *Locacao* que indicam que o recurso tratado pelo sistema é do tipo instanciável e quais classes do sistema representam o destino, o recurso e a taxa de multa da locação. É por meio desses métodos que são configuradas características específicas do sistema sobre uma locação. O GRENJ já fornece os recursos necessários para a funcionalidade de multa, por meio da classe *FineRate* e suas classes superiores, de forma que neste estudo de caso foi preciso apenas construir a classe *TaxaDeMulta* contendo dois construtores. Na classe *Cliente* somente os construtores e métodos de configuração das operações de persistência são necessários para que os atributos adicionados sejam contemplados pelo sistema.

```
public class Locacao extends ResourceRental {
   public TransactionQuantificationStrategy getTransactionQuantificationStrategyInstance() {
       return new InstantiableResTransaction();
   }
   public Class< ? extends DestinationParty> getDestinationPartyClass() {
       return Cliente.class;
   }
   public Class< ? extends Resource> getResourceClass() {
       return Filme.class;
   }
   public Class< ? extends FineRate> getFineRateClass() {
       return TaxaDeMulta.class;
   }
```

Figura 3.8. Código da classe *Locacao* que estende a classe *ResourceRental* do GRENJ.

Do terceiro grupo de padrões da GRN, apenas o padrão *Itemizar Transação do Recurso* foi aplicado para permitir que uma ou mais instâncias do recurso sejam vinculadas a uma transação. A classe *ItemLocacao* estende *TransactionItem* do GRENJ e é responsável por encapsular as informações referentes aos DVDs relacionados a uma locação. A classe *ItemQuantificationStrategy* implementa o padrão *Strategy* (Gamma *et al*, 1995) por meio de seus subtipos para permitir que um objeto de *ItemTransaction* tenha conhecimento em tempo de execução do tipo de quantificação do recurso a qual ele está associado. Pelo mesmo motivo, *TransactionQuantificationStrategy* também utiliza o padrão *Strategy* para indicar o tipo de quantificação do recurso associado com a classe *BusinessResourceTransaction* quando o padrão *Itemizar Transação do Recurso* não é utilizado. A Figura 3.9 mostra o modelo com a classe *ItemLocacao* e as classes do GRENJ que implementam o padrão Itemizar *Transação do Recurso*.

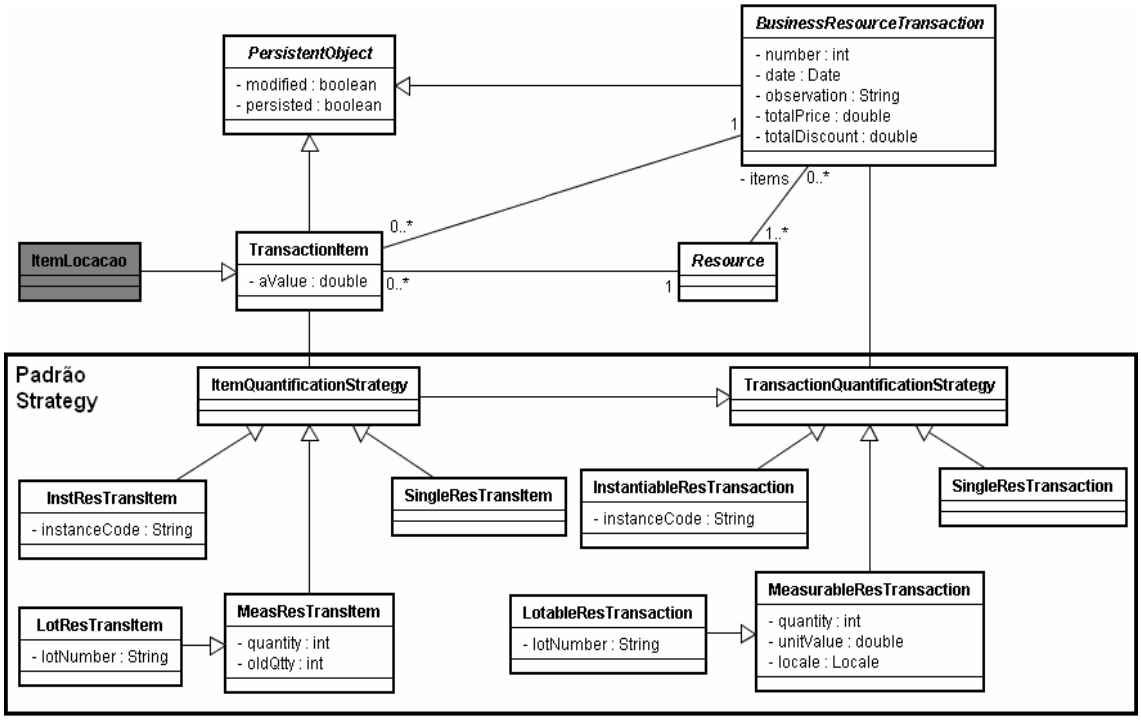

Figura 3.9. Classes do padrão Itemizar Transação do Recurso (Durelli, 2008).

O sistema da locadora de DVDs deste estudo de caso utilizou quatro padrões da GRN: *Identificar o Recurso*, *Quantificar o Recurso*, *Locar o Recurso* e *Itemizar Transação do Recurso*. A Figura 3.10 ilustra o grafo de fluxo da GRN com os padrões utilizados para o desenvolvimento do sistema da locadora de DVDs destacados na cor cinza.

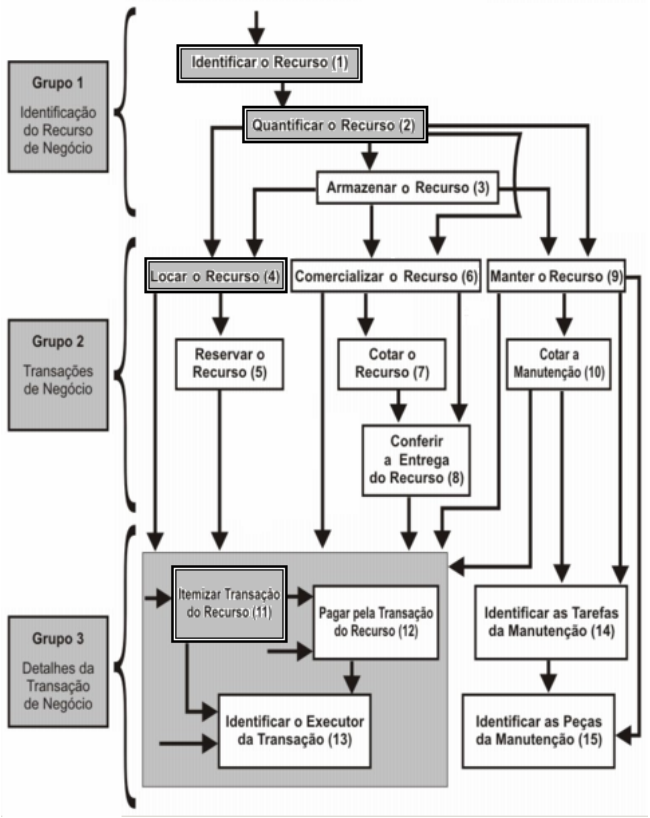

Figura 3.10. Fluxo de aplicação dos padrões da GRN utilizados no sistema da locadora de DVDs.

Oito classes foram criadas pelo desenvolvedor como extensões de classes do framework GRENJ para instanciar o sistema. A Tabela 3.2 apresenta as classes que foram criadas, especificamente, para o sistema para uma locadora de DVDs e suas respectivas superclasses provenientes do GRENJ. Como as classes definidas pela GRN estão implementadas no framework, a linguagem de padrões atua como um guia para a identificação das classes necessárias à camada de negócio do sistema. Uma das maiores vantagens do framewrok GRENJ em relação ao uso apenas da GRN é que, ao mesmo tempo em que as classes da camada de negócio são criadas, o desenvolvedor sobrescreve os métodos que configuram a camada de persistência.

Tabela 3.2. Relação das subclasses do estudo de caso e suas respectivas classes no GRENJ.

#	**Subclasse criada**	**Classe do GRENJ estendida**
1	*Filme*	*Resource*
2	*Categoria*	*SimpleType*
3	*Genero*	*SimpleType*
4	*DVD*	*ResourceInstance*
5	*Locacao*	*ResourceRental*
6	*Cliente*	*DestinationParty*
7	*Multa*	*FineRate*
8	*ItemLocacao*	*TransactionItem*

3.4. O Gerador de Aplicações Configurável Captor

O gerador de aplicações configurável Captor (do inglês, ***C****onfigurable* ***ap****plication generat****or***) foi desenvolvido para ser capaz de gerar aplicações pertencentes a domínios previamente configurados (Shimabukuro Junior, 2006). Identificadas as características de um domínio, sua configuração se inicia com a definição de um conjunto de formulários organizados hierarquicamente. Cada formulário contém campos (caixas de texto, caixas de seleção ou tabelas) para serem preenchidos com os dados específicos dos sistemas durante o processo de engenharia da aplicação. Os campos possuem propriedades que permitem identificá-los, customizá-los e atribuir-lhes regras de validação. Os formulários originam à LMA do domínio, que é persistida em um arquivo XML. A ordem hierárquica em que os formulários devem aparecer também deve ser definida nessa etapa. É possível que um formulário seja seguido por nenhum, um ou vários formulários ou que apareça em vários pontos da hierarquia. O Captor também permite que um formulário possua variantes. Quando um formulário é posicionado após outro na hierarquia, pode-se usar qualquer variante daquele formulário nessa posição.

Após a definição dos formulários, o engenheiro do domínio deve criar os gabaritos e os arquivos de configuração. As instruções XSL dos gabaritos devem indicar os campos dos formulários, cujos valores são extraídos para originarem o conteúdo dos artefatos resultantes. Um gabarito pode coletar informações de diversos formulários e as informações de um formulário podem ser utilizadas por diferentes gabaritos. Os arquivos de configuração fornecem mecanismos para a seleção dos gabaritos e a geração dos artefatos de uma aplicação (Shimabukuro Junior, 2006). A Figura 3.11 ilustra o processo de configuração de um domínio.

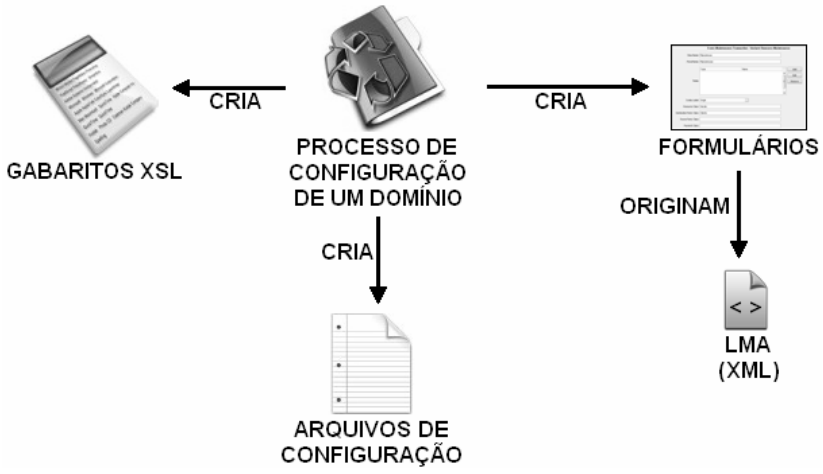

Figura 3.11. Processo de geração de uma aplicação por meio do Captor.

A geração de uma aplicação começa com a escolha de um domínio e a criação de um novo projeto de aplicação. Em seguida, os campos dos formulários devem ser preenchidos. Os formulários que forem preenchidos determinam quais características variáveis a aplicação que está sendo desenvolvida possuirá. Quando o projeto da aplicação é salvo, cria-se uma instância da LMA do domínio a qual pertence. Essa instância armazena dados sobre quais formulários foram utilizados, assim como os valores que foram preenchidos em seus campos. Em seguida, a aplicação pode ser gerada. O Captor mapeia os dados da instância da LMA nos gabaritos para dar origem aos artefatos de software (Shimabukuro Junior, 2006). A Figura 3.12 ilustra o processo de geração de uma aplicação por meio do Captor.

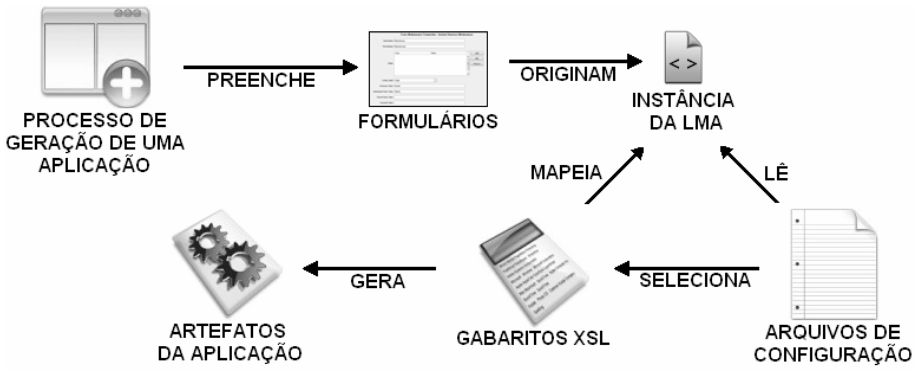

Figura 3.12. Processo de geração de uma aplicação por meio do Captor.

O Captor faz uso da ferramenta Ant (The Apache Software Foundation, 2009c) para a geração dos artefatos das aplicações. Essa ferramenta utiliza o arquivo de configuração *rules* para a seleção dos gabaritos de acordo com as informações contidas na sua instância da LMA. Instruções Ant podem ser escritas nos arquivos de configuração, *pre-build* e *pos-build*. No primeiro, são definidas as ações que devem ser executadas antes da geração dos artefatos de uma aplicação, enquanto que, no segundo, são definidas as ações que devem ser executadas após a geração da aplicação. Essas ações podem ser, por exemplo, copiar arquivos ou diretórios, compilar arquivos de código-fonte, executar scripts e compactar arquivos (Shimabukuro Junior, 2006).

3.4.1. Exemplo de utilização do Captor

Esta Seção apresenta um exemplo de configuração de um domínio e de geração de artefatos no Captor. O domínio a ser configurado é o de classes Java, por ser simples e permitir fácil entendimento do funcionamento desse gerador. Esse domínio é composto

de classes Java que possuem atributos, métodos *get* e *set*. A Tabela 3.3 contém a relação de características desse domínio.

Tabela 3.3. Relação de características do domínio de classes Java.

Característica		
Tipo	**Nome**	**Descrição**
Comum	Método construtor	Toda classe deve possuir um método construtor que provê valores iniciais aos atributos.
	Métodos *set*	Toda classe deve conter um método setNomeDoAtributo para cada atributo que possuir.
	Métodos *get*	Toda classe deve conter um método getNomeDoAtributo para cada atributo que possuir.
Variável	Classe	O nome da classe deve iniciar com uma letra maiúscula e deve conter apenas letras e números.
	Atributos	Os nomes dos atributos devem começar com uma letra minúscula e deve conter apenas letras e números..
	Tipo dos atributos	Pode ser um dos seguintes tipos: int, char, float, boolean ou String.

A configuração do domínio inicia com a definição dos formulários. Para o domínio de classes Java, um único formulário é suficiente para contemplar as características. Uma caixa de texto é adicionada ao formulário para o campo do nome da classe e uma tabela para os tipos e nomes dos atributos. A Figura 3.13(a) apresenta a tela do Captor para construção do formulário de uma classe Java. Customizações, como rótulo, identificação e regras de validação devem ser realizadas nas propriedades de cada campo do formulário. A Figura 3.13(b) apresenta as customizações realizadas na tabela para inserção de atributos na classe. Como o domínio permite gerar mais de uma classe, a Figura 3.13(c) ilustra o painel com a indicação de que várias instâncias do formulário criado podem ser inseridas.

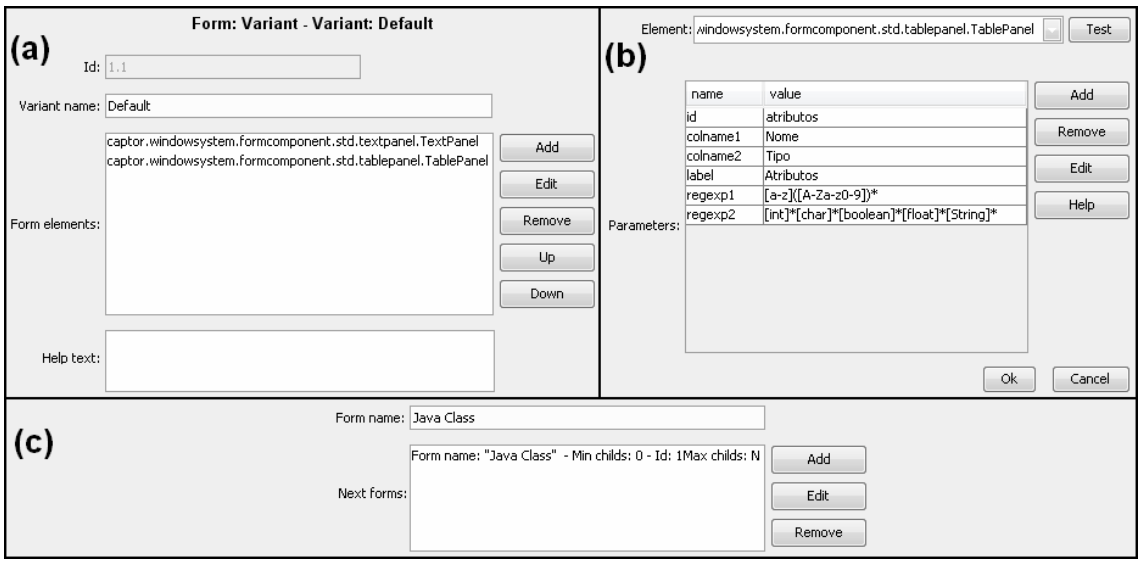

Figura 3.13. Construção do formulário para geração de uma classe Java.

O engenheiro do domínio deve salvar e ordenar a construção do projeto. Quando um projeto de domínio é construído, esse passa a constar na relação de domínios configurados e sua LMA, o gabarito *main* e os arquivos de configuração *rules*, *pre-build* e *pos-build* são gerados (Shimabukuro Junior, 2006). Porém, o conteúdo desses arquivos precisa ser alterado para que os artefatos desejados sejam gerados. Como no domínio de classes Java é gerado somente um tipo artefato, o gabarito *main* é suficiente. A Figura 3.14 ilustra o trecho de código do gabarito *main* que dá origem às instruções de definição do nome e dos atributos de uma classe Java.

```
public class <xsl:value-of select="/formsData/forms/form/data/textatt[@name='classe']"/>
{
<xsl:for-each select="/formsData/forms/form/data/table[@id='atributos']/row">
    private <xsl:value-of select="col[@number='1']/value"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="col[@number='0']/value"/>;
</xsl:for-each>
```

Figura 3.14. Trecho de código do gabarito *main*.

O arquivo de configurações *rules* deve indicar que, para cada instância do formulário de classe Java, o gabarito *main* deve ser utilizado para gerar um arquivo Java nomeado com o nome da classe definido na instância do formulário. A Figura 3.15 contém o trecho de código do arquivo *rules* que determina a execução dessa tarefa. Nenhuma ação é necessária antes da geração dos arquivos Java, portanto, o arquivo *pre-build* não precisa ser modificado. Contudo, uma instrução para compilação dos arquivos Java gerados é incluída no arquivo *pos-build*, Figura 3.16.

```
<main>
    <for-each select="//form[@name='Java Class']"> <callTask id="class"/> </for-each>
</main>

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

Figura 3.15. Trecho de código do arquivo *rules* que determina a criação dos arquivos Java.

```
<target name="compilacao">
    <javac srcdir="${project_output_path}/interaction_0/${project_name}"
        destdir="${project_output_path}/interaction_0/${project_name}" source="1.6" fork="yes"
        executable="${javahome}/bin/javac" classpathref="classpath"/>
</target>
```

Figura 3.16. Instrução para compilação dos arquivos Java incluída no arquivo *pos-build*.

O projeto de uma aplicação inicia com a seleção do domínio. O formulário para a primeira classe é mostrado automaticamente. O engenheiro da aplicação preenche os dados e pode escolher inserir outras instâncias do formulário clicando na árvore que mostra a hierarquia de formulários, posicionada no lado esquerdo da Figura 3.17. Ao final, o projeto deve ser salvo e gerado. A geração resulta em um arquivo Java e um arquivo compilado (*.class*) para cada instância do formulário de classe Java inserida.

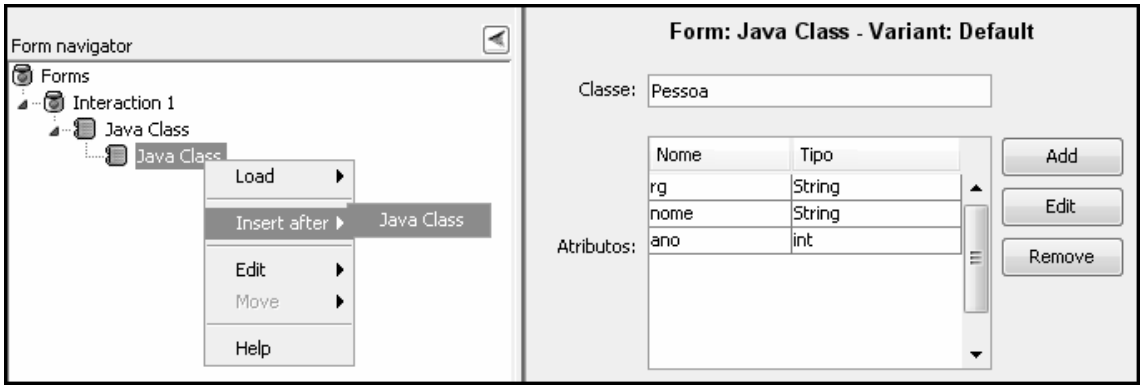

Figura 3.17. Preenchimento dos formulários de classe Java.

3.5. Considerações Finais

Com base no estudo de caso do sistema de informação de uma locadora de DVDs, apresentado na Seção 3.3.1, a implementação utilizando o framework GRENJ permitiu observar que houve diminuição de esforço de desenvolvimento necessário para um sistema do domínio de gestão de recursos de negócios. Com a utilização somente da GRN o desenvolvedor obtém reúso do modelo da camada de negócios (experiência proporcionada pelos padrões), mas fica sob sua responsabilidade a construção das outras camadas como, por exemplo, a de persistência e a de interface gráfica com o usuário, bem como a da arquitetura do sistema que define a comunicação entre todas as camadas.

Com o uso do GRENJ para o desenvolvimento de um sistema de informação, o desenvolvedor reutiliza, além do projeto, o código das camadas de persistência e de negócio, bem como da comunicação existente entre elas. O desenvolvedor precisa apenas estender algumas classes do framework e inserir métodos nessas classes com a finalidade de informar a essa camada a existência de novos atributos nas subclasses. No estudo de caso apresentado neste Capítulo, as camadas de persistência e de negócio do sistema da locadora de DVDs foram desenvolvidas com a construção de oito classes, com nove métodos por classe, em média, sendo que nenhum método possui mais de três

linhas de código. Isso representa um esforço muito inferior se comparado ao desenvolvimento do mesmo sistema utilizando apenas a GRN. Por exemplo, sem o uso do framework a camada de persistência exige a construção de classes para a conexão com o banco de dados e uma série de métodos para configurar as instruções de inclusão, atualização e exclusão dos dados do sistema. Com o framework GRENJ, o desenvolvedor só precisa acrescentar três métodos para modificar os comandos de script do banco de dados em cada classe da camada de negócio em que foram acrescentados atributos.

A construção da camada de interface gráfica e sua interligação com as demais camadas, normalmente, representam a maior parte do esforço gasto no desenvolvimento de um sistema instanciado do framework GRENJ, uma vez que sua implementação é realizada pelo desenvolvedor sem o apoio do framework. Além disso, existe maior possibilidade de inserção de defeitos no código do sistema, devido à não reutilização de classes previamente testadas, como ocorre nas demais camadas do framework GRENJ.

O gerador de aplicações configurável Captor facilita a obtenção de um gerador de aplicação para o domínio desejado, pois é mais simples configurar um domínio do que implementar um novo gerador. O engenheiro do domínio pode inserir defeitos na configuração do domínio, principalmente, durante a criação dos gabaritos XSL, em que os testes não são automatizados. Como conseqüência, o gerador irá gerar aplicações que possuem falhas em seu funcionamento. Entretanto, esses defeitos podem ser corrigidos à medida que novos sistemas são gerados e as falhas encontradas.

O Capítulo 4 descreve o desenvolvimento de uma camada de interface gráfica web para EAFs e a sua aplicação no framework GRENJ. Com isso, todas as camadas dos sistemas de informação pertencentes ao domínio de gestão de recursos de negócios poderão ser desenvolvidas com o apoio desse framework.

4 Um Framework para Construção da Camada de Interface Gráfica de Sistemas Web

4.1. Considerações Iniciais

Os benefícios obtidos com o uso de um EAF no desenvolvimento de um sistema de informação são proporcionais ao número de camadas construídas com o apoio desse framework. Todo EAF contempla, pelo menos, a camada de negócios, pois essa representa o domínio de aplicação a qual pertence o framework. Normalmente, a camada de persistência também é fornecida pelo EAF, mas pode ser implementada com o apoio de frameworks destinados a persistência de dados como, por exemplo, o Hibernate (Red Hat, 2009) e o JPA (Sun Mycrosystems, 2009d).

Alguns frameworks, como o *Struts* (The Apache Software Foundation, 2009b) o *Java Server Faces* (Sun Mycrosystems, 2009e), atuam como intermediadores entre a camada de negócio e a de interface gráfica com o usuário em aplicações voltadas para a web. Apesar de apoiarem com sucesso o desenvolvimento de sistemas com requisitos específicos, esses frameworks apresentam limitações quando utilizados no desenvolvimento de softwares genéricos como, por exemplo, outros frameworks. Além disso, esses frameworks apenas auxiliam na criação da estrutura (caixas de textos, menus, tabelas, etc.) da camada de interface gráfica, mas não definem a sua apresentação (cores, fontes, dimensões, etc.).

Este Capítulo descreve o desenvolvimento do framework Guiwe (do inglês, ***G****raphical* ***u****ser* ***i****nterface for* ***we****b*), que apóia a construção da camada de interface gráfica de um sistema de informação. Além disso, esse framework também pode ser instanciado para dar origem à camada de interface gráfica de um EAF, pois é capaz de se adaptar aos pontos variáveis em tempo de execução.

As demais seções deste Capítulo estão organizadas na seguinte ordem: na Seção 4.2 são descritas as características e o processo de desenvolvimento do framework Guiwe; na Seção 4.3 é abordado o processo de instanciação do framework Guiwe para a construção da camada de interface gráfica do framework GRENJ; e na Seção 4.4 são apresentadas as considerações finais.

4.2. Construção do Framework Guiwe

Inicialmente, uma das etapas deste projeto de mestrado consistia da construção da camada de interface gráfica do framework GRENJ. Entretanto, durante a realização dessa etapa, constatou-se que a implementação dessa camada poderia ser feita de maneira genérica. Desse modo, poderia ser aplicada não somente ao framework GRENJ, mas também a outros frameworks ou sistemas de informação implementados na linguagem Java. Essa mudança de enfoque deu origem ao framework Guiwe que, além de apoiar a construção de uma camada de interface gráfica web, fornece uma camada de controle que a interliga com a camada de negócios de um sistema e/ou framework.

O desenvolvimento de sistemas voltados para a web engloba várias tecnologias diferentes como, por exemplo, Java, servlets, JSP, HTML, CSS e JavaScript. Além de exigir que se tenha conhecimento sobre essas tecnologias, são necessários esforços extras para organizá-las e interligá-las. O que torna o processo de desenvolvimento complexo. Essas dificuldades se agravam quando se utiliza um framework, como o *Struts*, que depende de diferentes tecnologias web para ser instanciado. Desse modo, umas das características desejadas para o framework Guiwe é a redução do uso dessas tecnologias. A construção da camada de interface gráfica do framework GRENJ, que foi implementado em Java, também fez uso dessa linguagem. Foram implementados mecanismos para a geração do código HTML a partir de um código Java e servlets foram utilizados no lugar de JSP. O uso das linguagens CSS e de JavaScript não pôde ser substituído pela linguagem Java, devido a particularidades dessas tecnologias e, também, para evitar restrições à flexibilidade da interface gráfica. Porém, o desenvolvedor de um sistema instanciado a partir do framework Guiwe somente terá necessidade de utilizar essas tecnologias caso queira adicionar funções não previstas pelo framework ou alterar a apresentação da interface gráfica.

Esta Seção trata do processo de desenvolvimento do framework caixa-branca Guiwe, que foi construído com base no padrão arquitetural *Model-View-Controller* (MVC) (Gamma *et al.*, 1995). Esse processo foi realizado com a aplicação da prática de desenvolvimento guiado por testes (Janzen e Saiedian, 2008) com as seguintes etapas: definição das características do framework; projeto e codificação das unidades, com criação de testes, construção das classes, execução dos testes e refatoração; e a verificação do framework GRENJ por completo.

4.2.1. Características do Framework Guiwe

A interface gráfica provida pelo framework Guiwe pode ser organizada em uma tela com cabeçalho, menu principal, painel principal e rodapé. No cabeçalho, no rodapé e no painel principal podem ser inseridas páginas HTML com imagens e textos como, por exemplo, logotipo, nome e descrição do sistema. O menu principal deve conter as opções de páginas que podem ser carregadas no painel principal. A organização da interface gráfica criada pelo framework Guiwe é apresentada na Figura 4.1 (a).

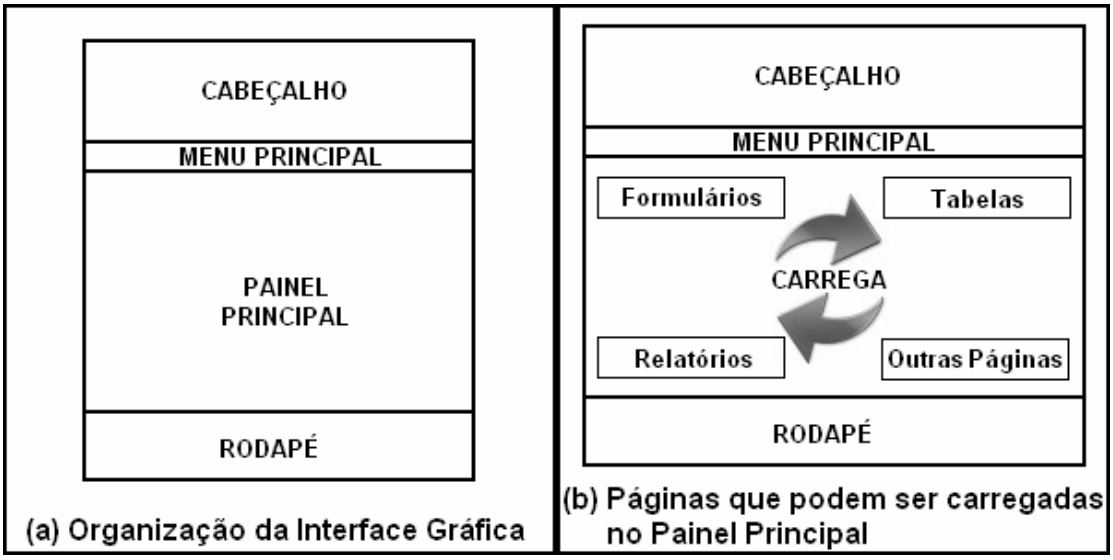

Figura 4.1. Organização da interface gráfica do framework Guiwe.

A Figura 4.1 (b) mostra que no painel principal da interface gráfica podem ser carregados formulários, tabelas, relatórios e outras páginas que compõem o sistema em desenvolvimento. Os formulários podem conter menus, tabelas, painéis e campos que são preenchidos pelo usuário. Cada formulário está relacionado com uma classe da camada de negócios do sistema em desenvolvimento e os campos desse formulário correspondem aos atributos dessa classe.

Os painéis representam classes que mantém um relacionamento de composição ou agregação com outras e, portanto, não possuem um formulário próprio. Um exemplo desse tipo de relacionamento existe entre uma classe que representa uma venda e outra que representa os itens de uma venda. Um painel contém um menu, uma tabela e campos e funciona de forma semelhante a um formulário. A Figura 4.2 apresenta um formulário com menu, tabela, campos e um painel que, também, possui um menu, uma tabela e campos de texto.

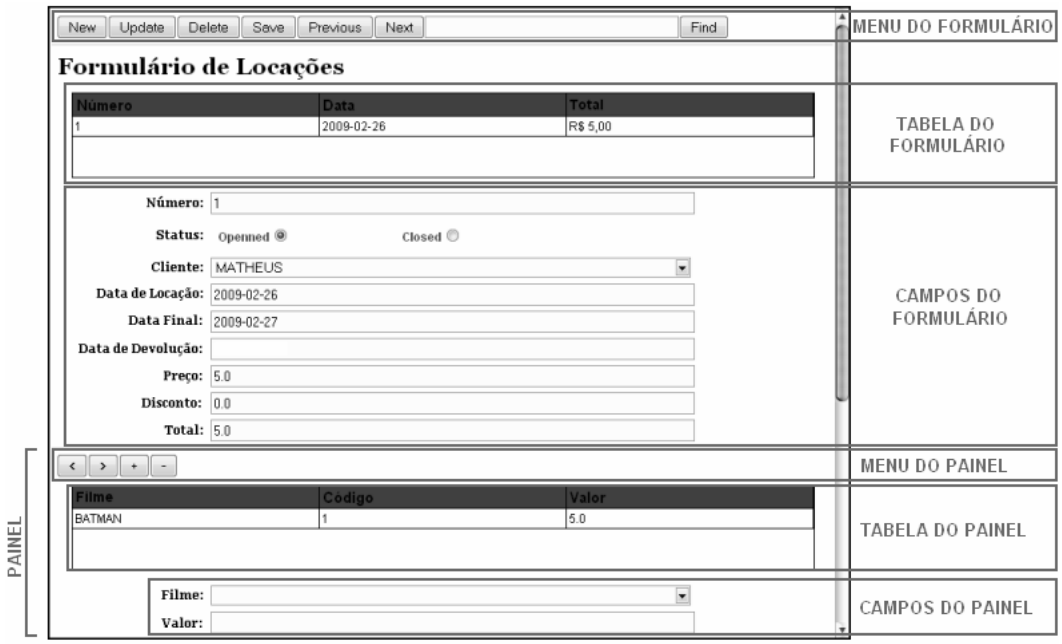

Figura 4.2. Componentes de um formulário do framework Guiwe.

Além de formulários, o framework deve prover meios para a construção de tabelas e relatórios. As tabelas podem carregar dados de todos os registros de uma tabela do banco de dados ou somente de alguns. Os relatórios devem permitir ao usuário filtrar os dados que devem ser apresentados. A camada de controle é responsável por carregar e enviar as informações que devem ser disponibilizadas nas tabelas e nos relatórios da camada de interface gráfica.

O framework Guiwe deve prover um mecanismo para a interligação de sua camada de interface gráfica com a camada de negócios do software para o qual está sendo instanciado. Esse mecanismo representa uma camada de controle que gerencia a execução da funcionalidade do software, criando objetos para persistir e recuperar informações da base de dados. Essas informações podem pertencer a atributos que foram declarados na própria classe ou provenientes de relacionamentos de associação ou herança. A camada de controle deve possuir pontos variáveis que permitem ao framework reconhecer e ter acesso aos valores dos atributos das classes da camada de negócios em qualquer um desses casos. Caso a camada de negócios pertença a um EAF, alguns atributos são declarados somente durante um processo de instanciação e só podem ser reconhecidos pela camada de controle em tempo de execução.

Os menus dos formulários, painéis e relatórios permitem ao usuário interagir com o sistema. A interface gráfica deve reconhecer os eventos do usuário e enviar as informações necessárias para que a camada de controle efetue a operação desejada. A

Figura 4.3 descreve, como exemplo, o que ocorre quando um usuário de um sistema implementado com o uso do framework Guiwe solicita a gravação dos dados preenchidos em um formulário.

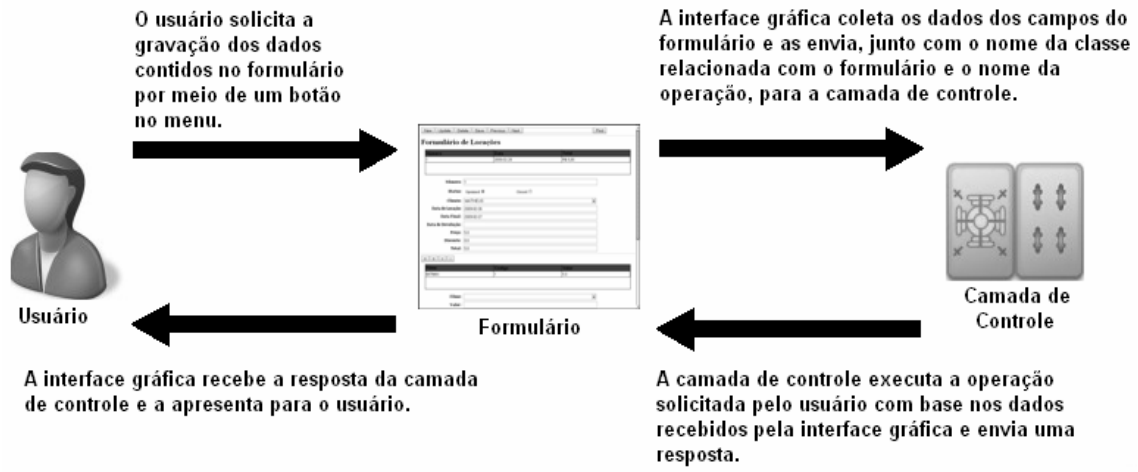

Figura 4.3. Passos de uma operação de gravação dos dados de um formulário.

Como foi definido que o uso da linguagem Java deve ser priorizado, os formulários, painéis, tabelas, relatórios e demais páginas são construídos por classes da camada de interface gráfica do framework Guiwe, enquanto que a camada de controle é responsável por manipular objetos da camada de negócios da aplicação ou do EAF, cuja camada de interface gráfica foi instanciada do framework Guiwe. A Tabela 4.1 contém a relação de responsabilidades das camadas de interface gráfica e de controle do framework Guiwe com base nas características descritas nesta Seção.

Tabela 4.1. Responsabilidades das camadas de interface gráfica e de controle do framework Guiwe.

Camada	**Responsabilidade**
Interface Gráfica	Prover estrutura da tela.
	Enviar mensagens para a camada de controle e obter resposta.
	Construir páginas, formulários, painéis, tabelas e relatórios.
Controle	Receber e responder mensagens da camada de interface gráfica
	Criar objetos da camada de negócios do sistema/EAF.
	Efetuar operações sobre objetos da camada de negócios do sistema/EAF.

A Figura 4.4 apresenta o modelo de classes em nível de análise da camada de interface gráfica do framework Guiwe. Por motivos de simplificação, os métodos das classes foram omitidos. A Tabela 4.2 apresenta uma relação, com nome e descrição, das classes que aparecem no modelo da Figura 4.4.

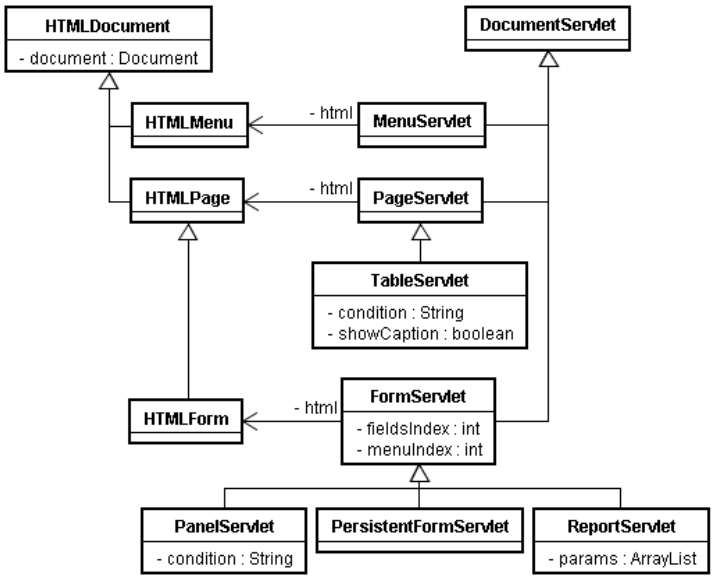

Figura 4.4. Modelo de classes da camada de interface gráfica do framework Guiwe.

Tabela 4.2. Relação das classes da camada de interface gráfica do framework Guiwe.

Nome	**Descrição**
HTMLDocument	Responsável pela geração de um documento HTML.
HTMLMenu	Permite construir páginas que contêm um menu interativo.
HTMLPage	Permite construir páginas que podem conter textos, tabelas e imagens.
HTMLForm	Permite construir páginas que contêm formulários com campos de texto, de seleção, de checagem (*checkboxes*) e de marcação exclusiva (*radio*).
DocumentServlet	Recebe uma requisição do navegador e envia código HTML.
MenuServlet	Recebe uma requisição do navegador e envia o código HTML de uma página que contém um menu interativo.
PageServlet	Recebe uma requisição do navegador e envia o código HTML de uma página que pode conter textos, tabelas e imagens.
TableServlet	Recebe uma requisição do navegador e envia o código HTML de uma tabela com informações recuperadas da base de dados.
FormServlet	Recebe uma requisição do navegador e envia o código HTML de um formulário.
PersistentFormServlet	Recebe uma requisição do navegador e envia o código HTML de um formulário relacionado com uma classe persistente.
PanelServlet	Recebe uma requisição do navegador e envia o código HTML de um painel.
ReportServlet	Recebe uma requisição do navegador e envia o código HTML de um relatório.

A Figura 4.5 apresenta o modelo de classes em nível de análise da camada de controle do framework Guiwe e a Tabela 4.3 apresenta a descrição dessas classes.

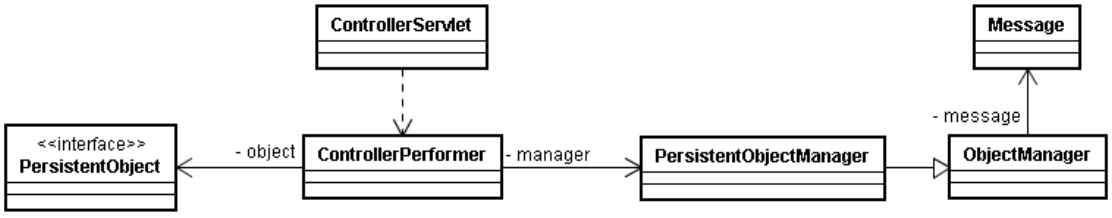

Figura 4.5. Modelo de classes da camada de controle do framework Guiwe.

Tabela 4.3. **Relação das classes da camada de controle do framework Guiwe.**

Nome	Descrição
ControlerServlet	Recebe as mensagens da camada de interface gráfica e envia as respostas.
ControlerPerformer	Efetua a operação solicitada sobre um objeto da classe indicada na mensagem enviada pela camada de interface gráfica.
PersistentObject	Interface que define os métodos obrigatórios para as classes persistentes.
ObjectManager	Permite ler e alterar, dinamicamente, os valores dos atributos de um objeto.
PersistentObjectManager	Permite carregar informações da base de dados.
Message	Gerencia as mensagens que são enviadas à camada de interface gráfica e apresentadas ao usuário do sistema.

4.2.2. Projeto e Implementação do Framework Guiwe

A prática do desenvolvimento guiado por testes (do inglês, *Test-Driven Development* – TDD) preconiza que não é necessário criar um modelo detalhado das classes do software em desenvolvimento, pois os casos de teste criados em cada iteração servem de documentação sobre o escopo e as interfaces das classes do software (Janzen e Saiedian, 2008). Dessa forma, o primeiro passo do projeto e implementação do framework Guiwe foi definir que cada iteração seria responsável pela construção de uma classe. Essa decisão foi devido ao fato de que cada classe possui uma responsabilidade dentro do framework.

A construção das classes teve como base os passos dos ciclos TDD: criação de uma lista de casos de teste, implementação dos casos de teste, implementação das classes, execução do testes e refatoração. Para exemplificar esse processo, segue-se a descrição de como foi realizada a construção da classe *HTMLDocument*, que pertence à camada de interface gráfica. Essa classe é responsável por construir um documento HTML que possui um elemento *span* como sendo o raiz (principal). Esse elemento serve como um *container* para os demais elementos do documento e possui um atributo que o relaciona com o CSS que define a apresentação das páginas carregadas no painel principal da interface gráfica.

No projeto do framework Guiwe, foi definido que as listas de casos de teste devem conter as descrições e as assinaturas dos métodos da classe que está sendo construída. A descrição de um método indica o que deve ser testado e a sua assinatura define como se deseja que seja a sua interface. Por motivos de simplificação, construtores e métodos que apenas configuram ou retornam o valor de um atributo da classe não constam nas listas de casos de teste. Outro detalhe importante, a lista de

casos de testes sempre foi escrita no próprio arquivo Java da classe que estava sendo construída. Durante a implementação, as descrições dos métodos foram transformadas na documentação *javadoc* (Sun Mycrosystems, 2009f) e as assinaturas ganhavam o corpo do método. A classe *HTMLDocument* deve conter métodos para a inserção de elementos e para a obtenção do código do documento HTML que está sendo montado. A Figura 4.6 contém a lista de casos de teste da classe *HTMLDocument*.

```
public class HTMLDocument {

  /**Retorna o elemento raiz do documento HTML.*/
  public Element getContainer(){}

  /**Adiciona um elemento ao documento HTML como filho do elemento raiz.*/
  public void addTag( Element tag ){}

  /**Adiciona um elemento ao documento HTML como filho do elemento identificado por id.*/
  public void addTag( String id, Element tag ){}

  /**Cria e retorna um elemento.*/
  public Element createTag( String name, Map<String, String> attributes, ArrayList<Node> childs ){}

  /**Retorna o código do documento HTML.*/
  public String toString(){}

}
```

Figura 4.6. Lista de Casos de Teste da classe *HTMLDocument*.

Ao invés de escrever todos os testes e, em seguida, implementar o código de todos os métodos e testá-los, optou-se por escrever os testes e implementar o código de um método por vez. Por exemplo, o método *getContainer* necessita de apenas um teste que verifica se o seu retorno é o elemento *span* do documento HTML. A implementação desse método contém somente uma instrução que retorna o primeiro elemento do documento HTML que, supostamente, deve ser o elemento *span*. A Figura 4.7(a) mostra o código do teste e a Figura 4.7(b) o código do método *getContainer*.

```
@Test
public void getContainer() {
    System.out.println( "getContainer" );
    HTMLDocument instance = new HTMLDocument();
    Element tag = instance.getContainer();
    assertEquals( "span", tag.getNodeName() );
}
```

(a)

```
/**
 * Returns the container of the document. In HTMLDocument class, the container
 * is the root element span.
 * @return the Element that is the container of this document.
 */
public Element getContainer() {
    return (Element) document.getFirstChild();
}
```

(b)

Figura 4.7. Implementação do teste (a) e do código (b) do método *getContainer*.

Após a implementação de um método, seus testes são executados. Caso ocorra um erro, a implementação deve ser corrigida até que passe pelo teste. Refatorações foram realizadas somente após a implementação de todos os métodos da classe. Vale salientar que alguns métodos não foram identificados durante a criação da lista dos casos de testes da sua classe e sua necessidade foi percebida somente durante a refatoração ou a construção de outra classe. Por exemplo, durante a refatoração da classe *HTMLDocument*, percebeu-se que seu código se tornaria mais flexível se existisse um método que retornasse o código HTML de um elemento e não somente de todo o documento. Assim, foi criado o método *tagToString* e o método *toString* passou a chamar esse método, informando como parâmetro o elemento raiz. A lista de casos de teste de uma classe também era alterada quando era detectada a necessidade de inserção de um novo método nessa classe. Um exemplo desse caso foi a criação do método *copyElement* na classe *HTMLDocument*. Esse método permite criar uma cópia de um elemento que pertence ao documento de outra instância de *HTMLDocument* e sua necessidade foi percebida durante a construção da classe *PanelServlet*.

As mensagens enviadas pela camada de interface gráfica para a de controle devem conter o nome da classe, o nome da operação a ser efetuada e os nomes e os valores dos atributos dessa classe. Essas informações são suficientes para que a classe *ConrollerPerformer* crie um objeto da classe indicada com os valores dos atributos configurados e efetue a operação. As operações possíveis correspondem àquelas determinadas pelo padrão *Persistent Object* (Yoder *et al.*, 1998): salvar, atualizar, remover e recuperar o registro dos dados de um objeto na base de dados.

A criação e atribuição dos valores do objeto da classe indicada na mensagem é definida pelas classes *ObjectManager* e *PersistentObjectManager*. A classe *ObjectManager* possui um método, chamado *setFields*, que recebe uma lista de nomes e valores que devem ser atribuídos aos atributos de um objeto de qualquer classe e outro método, chamado *getFields*, que lê um objeto e retorna uma mensagem com os nomes e valores de seus atributos. Para que isso funcione, é imprescindível que os objetos tenham métodos *getters* e *setters* para todos os seus atributos. A Figura 4.8 apresenta o trecho de código do método *getFields* que obtém a lista de atributos (*fields*) de um objeto e invoca os métodos *getters* de cada um deles, acumulando seus nomes e valores na mensagem de resposta (*result*) no formato *&nome_do_atributo=valor_do_atributo*.

```
Field fields[] = object.getClass().getDeclaredFields();

for ( Field field : fields ) {
    fieldName = field.getName();
    String methodName = "get" + capitalize( fieldName );
    Method method = object.getClass().getMethod( methodName );
    result += "&" + field.getName() + "=" + method.invoke( object );
}
```

Figura 4.8. Trecho de código que obtém o nome e o valor dos atributos de um objeto.

A classe *PersistentObjectManager* é uma extensão de *ObjectManager* e inclui métodos que permitem buscar os registros da base de dados para, então, inseri-los em objetos que implementam o padrão *PersistentObject*. *PersistentObjectManager* também possui um método que retorna uma lista de registros da base de dados para ser utilizado na construção de tabelas na interface gráfica.

4.2.3. Construção da Apresentação e do Script da Interface Gráfica

A linguagem Java foi utilizada na construção das classes que geram o código HTML a ser carregado no navegador e das classes que controlam o funcionamento do framework Guiwe. Contudo, ainda foi necessário o uso de documentos CSS para definir a apresentação da interface gráfica e arquivos JavaScript para o carregamento das páginas no painel principal, o tratamento de eventos do usuário e para a manipulação dos dados dos formulários, painéis e relatórios.

Ao todo, foram criados sete documentos CSS para definir a apresentação padronizada dos sistemas instanciados a partir do framework Guiwe. Cada um é responsável pela apresentação de uma estrutura da interface gráfica. Esses documentos são:

- *gridlayout* – define o tamanho, as margens e bordas da tela da interface gráfica.
- *header*, *horizontalmenu* e *footer* – respectivamente, definem o tamanho, o formato, a fonte principal e as cores do cabeçalho, do menu principal e do rodapé.
- *home* – define as fontes e o formato do título e dos parágrafos da página inicial carregada no painel principal.
- *persistentform* - define o tamanho, as cores e o formato dos menus e dos campos de formulários, painéis e relatórios.

- *table* – define o formato, as fontes e as cores das tabelas que mostram informações existentes na base de dados.

Os documentos CSS são organizados em blocos de declaração que contém uma lista de declarações de estilo (W3C, 2009). Os elementos do código HTML gerado pelas classes Java possuem um atributo chamado *class* que indica qual bloco de declaração define sua apresentação. O navegador é responsável por apresentar os elementos do código HTML de acordo com as declarações de estilo dos blocos com os quais estão relacionados.

O desenvolvedor da aplicação não precisa, obrigatoriamente, modificar nem criar novos documentos CSS para desenvolver um sistema de informação utilizando o framework Guiwe. Isso só é necessário caso deseje personalizar a apresentação do sistema desenvolvido.

O framework Guiwe faz uso de quatro arquivos JavaScript. O arquivo *jquery* (Resig, 2009) fornece uma interface para execução dos comandos AJAX. O arquivo *main* controla o funcionamento do menu principal e utiliza a interface do arquivo *jquery* para carregar o cabeçalho, o menu principal e o rodapé, assim como os formulários, as tabelas, os relatórios, a página inicial e outras no painel principal. O arquivo *persistentform* realiza a coleta e envio dos dados do formulário, painel ou relatório que estiver carregado na tela para que a camada de controle efetue a operação indicada. O arquivo *persistentform* também é responsável por receber a resposta da operação efetuada e disponibilizá-la para o usuário. O arquivo *transaction* realiza funções semelhantes às do arquivo *persistentform*, porém, específicas dos painéis.

4.2.4. Finalização do Framework Guiwe

A prática do TDD não pôde ser aplicada na construção dos documentos CSS e JavaScript porque não há meios para verificar a apresentação da interface gráfica e os comandos AJAX com testes automatizados. Portanto, a verificação do código desses documentos, bem como a verificação do funcionamento geral do framework Guiwe, só poderiam ser realizadas com uma instanciação desse framework.

A apresentação e o script do cabeçalho, do menu principal, da página inicial do painel principal, do rodapé e de outras páginas com texto e imagem foram verificados com a instanciação do framework Guiwe após a construção das classes responsáveis por

essas páginas. Essa instanciação não constituía um sistema, mas, somente, um conjunto de páginas web.

Como o framework Guiwe foi, inicialmente, desenvolvido para ser utilizado na construção da camada de interface gráfica do framework GRENJ, o funcionamento da sua camada de controle e do script das páginas foi verificado após a construção das classes das camadas de interface gráfica e de controle do framework GRENJ e a instanciação de um sistema utilizando esse framework. À medida que cada padrão do framework GRENJ era implementado, avançava-se com a instanciação de um sistema para uma locadora de DVDs e, desse modo, o funcionamento geral dos dois frameworks era verificado. Maiores detalhes sobre a utilização do framework Guiwe na construção da camada de interface gráfica do framework GRENJ e da instanciação do sistema da locadora de DVDs são apresentados na Seção 4.3.

4.3. Construção da Camada de Interface Gráfica do Framework GRENJ

A construção da camada de interface gráfica do framework GRENJ foi realizada com a instanciação do framework Guiwe. Essa instanciação ocorreu com a construção de classes que se relacionam com as da camada de negócios e que herdam, direta ou indiretamente, classes das camadas de interface gráfica ou de controle. Por exemplo, foram construídas as classes *StaticObjectManager* e *StaticObjectFormServlet* que herdam, respectivamente, as classes *PersitentObjectManager* e *PesistentFormServlet* do framework Guiwe. Ambas estão relacionadas com a classe *StaticObject* da camada de negócios do GRENJ. Essa construção permite que o processo de instanciação de um sistema a partir do framework GRENJ continue seguindo a seqüência dos padrões da GRN. Nesse processo, o desenvolvedor deve estender as classes das camadas de negócios e de interface gráfica relativas aos padrões aplicados para a construção do sistema. Assim, como ocorre com as classes da camada de persistência, não é necessário estender as classes da camada de controle, uma vez que o comportamento dessas classes se mantém independente das características específicas da aplicação instanciada.

Assim como ocorreu com o framework Guiwe, o processo de desenvolvimento das camadas de interface gráfica e de controle do framework GRENJ também foi realizado com a aplicação da prática de desenvolvimento guiado por testes (TDD) (Janzen e Saiedian, 2008).

4.3.1. Análise da Camada de Interface Gráfica do Framework GRENJ

A camada de negócios do framework GRENJ fornece uma série de classes que representam a implementação dos padrões da GRN. Porém, o número de classes implementadas é maior que o existente nos modelos desses padrões. Isso ocorre porque os modelos da GRN estão em nível de análise. Classes extras tiveram que ser construídas com base em padrões de software (Gamma *et al.*, 1995) para apoiar a variabilidade do framework. A camada de interface gráfica deve prover formulários ou painéis somente para as classes que aparecem nos modelos dos padrões da GRN. Essas classes correspondem àquelas que são diretamente estendidas durante a instanciação da camada de negócios de um sistema com base no framework GRENJ. A Figura 4.9 ilustra os modelos de classes do primeiro padrão da GRN, Identificar o Recurso, e de sua implementação na camada de negócios do framework GRENJ. As classes do GRENJ que correspondem às classes da GRN estão destacadas pela cor cinza. Essas classes destacadas devem possuir um formulário na camada de interface gráfica.

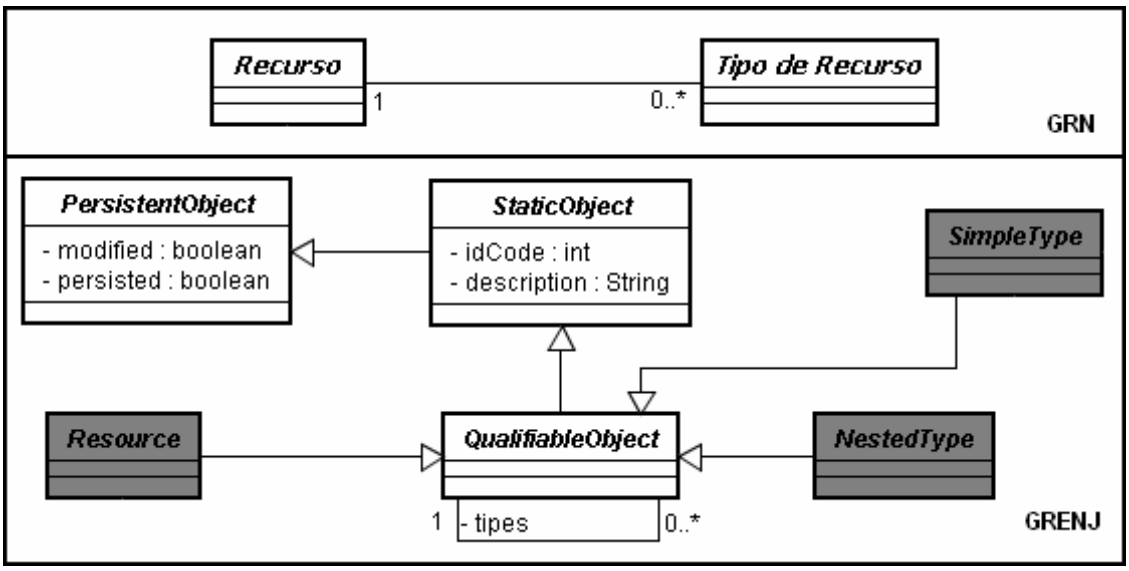

Figura 4.9. Modelo de classes do primeiro padrão da GRN e seu equivalente no GRENJ.

Algumas classes da camada de negócios devem estar relacionadas com classes da camada de interface gráfica que originam os formulários, enquanto que outras estão relacionadas com classes que dão origem aos painéis. Por exemplo, na camada de interface gráfica deve existir uma classe que constrói um formulário para *ResourceRental*, que representa uma transação de aluguel. Porém, a classe *TransactionItem*, que representa os itens de uma transação, está relacionada com uma classe que constrói um painel. Essa diferença é devido ao fato dos itens de uma

transação não existirem fora do contexto de uma transação. Os formulários e painéis contêm os campos para os atributos das classes da camada de negócio do GRENJ e o desenvolvedor deve acrescentar os campos específicos do sistema que está sendo instanciado.

A Figura 4.10 apresenta o modelo de classes da camada de interface gráfica do framework GRENJ. As classes destacadas na cor cinza provêm do framework Guiwe. As classes *TableServlet* e *ReportServlet* não possuem subclasses no framework GRENJ porque são utilizadas para criar tabelas e relatórios específicos dos sistemas instanciados com base nesse framework. As classes que estendem direta ou indiretamente *PersistentFormServlet* ou *PanelServlet* estão relacionadas com as classes da camada de negócios que possuem nomes semelhantes. Por exemplo, a classe *ResourceFormServlet* é utilizada para construir um formulário para a classe *Resource* da camada de negócios do framework GRENJ. As hierarquias das classes das camadas de interface gráfica e de negócios são similares.

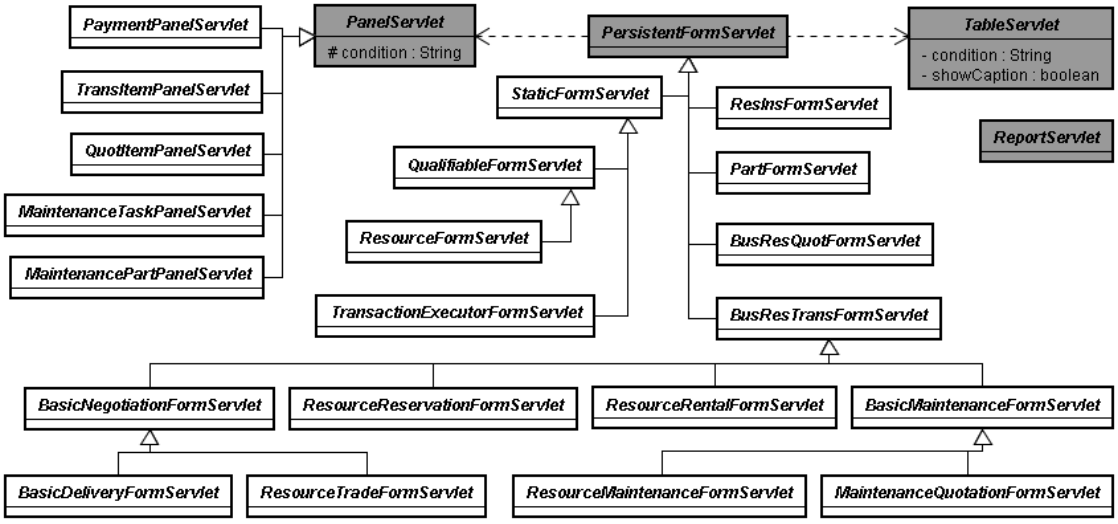

Figura 4.10. Modelo de classes da camada de interface gráfica do framework GRENJ.

A camada de controle do framework GRENJ deve prover um conjunto de classes que recebem e respondem às solicitações dos formulários e painéis da camada de interface gráfica. Essas classes também estão relacionadas às mesmas classes da camada de negócios que possuem formulários ou painéis. A Figura 4.11 apresenta o modelo de classes da camada de controle do framework GRENJ. Assim como ocorreu na camada de interface gráfica, a hierarquia das classes também se assemelha com a da camada de negócios.

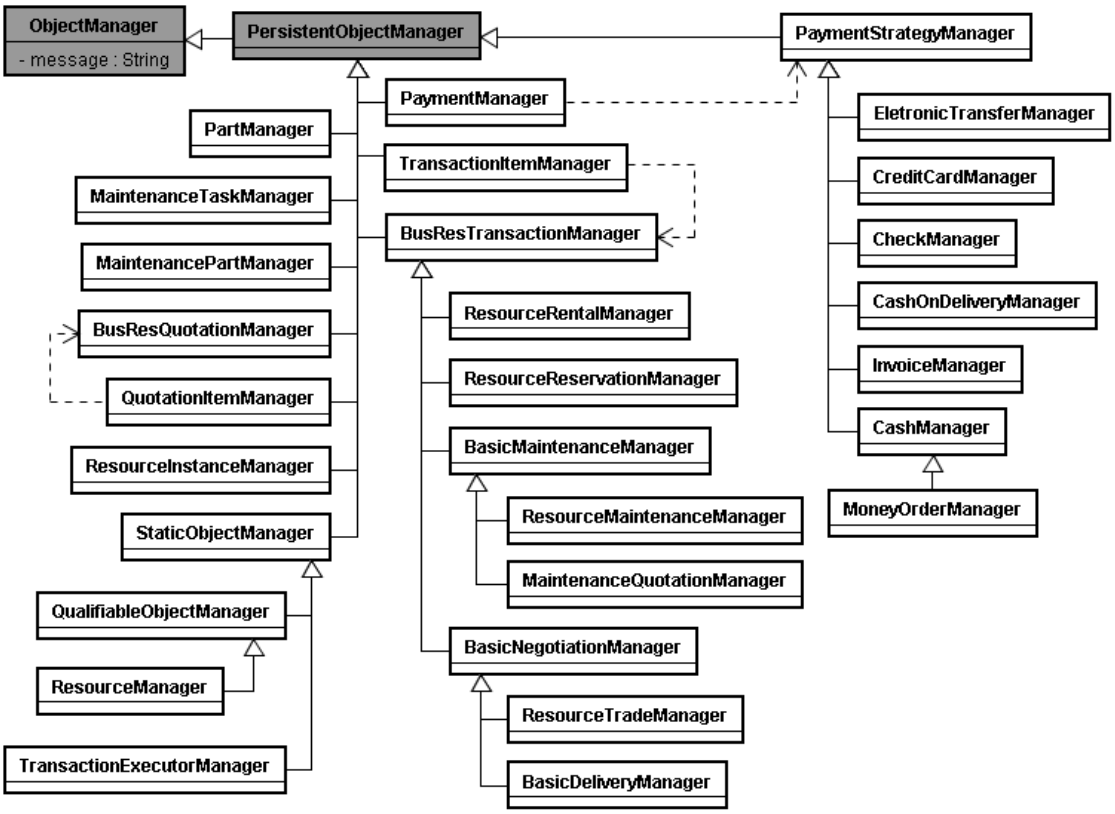

Figura 4.11. Modelo de classes da camada de controle do framework GRENJ.

4.3.2. Projeto e Implementação da Camada de Interface Gráfica do Framework GRENJ

O projeto e a implementação da camada de interface gráfica do framework GRENJ foram divididos em quatorze iterações. Essas iterações correspondem aos quatorze dos quinze padrões da GRN, uma vez que o terceiro padrão, Armazenar o Recurso, não foi implementado no framework GRENJ, pois a forma com foi implementado o tornou pouco utilizado. Cada iteração contemplou a implementação de um padrão na camada de controle e na de interface gráfica.

A seqüência de escolha dos padrões para serem implementados foi ligeiramente diferente da seqüência indicada pelo número dos padrões da GRN. Após o primeiro, o segundo e o quarto padrões terem sido implementados, o décimo primeiro padrão foi escolhido para ser o próximo. A razão dessa escolha foi que a interface gráfica desses padrões contempla todas as funções principais: montagem de formulários, montagem e inserção de tabelas e painéis nos formulários e efetuação de todas as operações de persistência. Essa seqüência permitiu que os detalhes do projeto fossem definidos com mais antecedência, o que diminuiu a ocorrência de modificações e de re-trabalho. Após

a implementação do décimo primeiro padrão, seguiu-se a ordem a partir do quinto padrão da GRN. A Tabela 4.4 lista as classes construídas em cada iteração nas camadas de controle e de interface gráfica.

Tabela 4.4. Lista das classes construídas para cada padrão da GRN.

#	Padrão	Camada de Controle	Camada de Interface Gráfica
1	1	*StaticObjectManager*, *QualifiableObjectManager*	*StaticFormServlet*, *QualifiableFormServlet*
2	2	*ResourceInstanceManager*	*ResInsFormServlet*
3	4	*BusResTransactionManager*, *ResourceRentalManager*	*BusResTransFormServlet*, *ResourceRentalFormServlet*
4	11	*TransactionItemManager*	*TransItemPanelServlet*
5	5	*ResourceReservationManager*	*ResourceReservationFormServlet*
6	6	*BasicNegotiationManager*, *ResourceTradeManager*	*BasicNegotiationFormServlet*, *ResourceTradeFormServlet*
7	7	*BusResQuotationManager*, *QuotationItemManager*	*BusResQuotFormServlet*, *QuotItemPanelServlet*
8	8	*BasicDeliveryManager*	*BasicDeliveryFormServlet*
9	9	*BasicMaintenanceManager*, *ResourceMaintenanceManager*	*BasicMaintenanceFormServlet*, *ResourceMaintenanceFormServlet*
10	10	*MaintenanceQuotationManager*	*MaintenanceQuotationFormServlet*
11	12	*PaymentManager*, *PaymentStrategyManager*, *CashManager*, *CheckManager*, *InvoiceManager*, *CashOnDeliveryManager*, *CreditCardManager*, *MoneyOrderManager*, *EletronicTransferManager*	*PaymentPanelServlet*
12	13	*TransactionExecutorManager*	*TransactionExecutorFormServlet*
13	14	*MaintenanceTaskManager*	*MaintenanceTaskPanelServlet*
14	15	*PartManager*, *MaintenancePartManager*	*PartFormServlet*, *MaintenancePartPanelServlet*

A forma de criação da lista de casos de testes, escrita dos testes, implementação das classes e execução dos testes foi a mesma da aplicada no projeto e na implementação do framework Guiwe, que foram descritos na Seção 4.2.2. Entretanto, no framework GRENJ, a implementação de cada padrão implicou na construção de, pelo menos, duas classes, uma na camada de controle e outra na camada de Interface gráfica. Assim, havia uma lista de casos de testes para cada classe a ser implementada dentro de uma iteração.

Como uma classe pode ser estendida por várias outras, o formulário dessa classe pode ser utilizado por suas subclasses. Por exemplo, a classe *QualifiableFormServlet* pode ser utilizada para montar formulários paras as classes *SimpleType* e *NestedType*, que são subclasses de *QualifiableObject*. De fato, as classes *PersistentFormServlet* e *PanelServlet* podem prover uma interface gráfica para todas as classes da camada de

negócios. Contudo, isso não é aconselhável porque as demais classes da camada de interface gráfica foram construídas para contemplar as características específicas das classes da camada de negócios e, assim, reduzir o esforço do desenvolvedor durante o desenvolvimento de um sistema baseado no framework GRENJ. Por exemplo, a classe *ResInsFormServlet* permite construir um formulário que já contém os campos relativos aos atributos da classe *ResourceInstance* e o desenvolvedor não precisa adicionar instruções em sua subclasse para que isso ocorra.

Na camada de controle, entretanto, não é possível utilizar a classe *PersistentObjectManager* para tratar de objetos das subclasses de *PersistentObject*. Os atributos das subclasses da camada de negócios tiveram que ser tratados individualmente em cada nível da hierarquia de classes para que seus valores pudessem ser atualizados ou obtidos em tempo de execução. Além disso, muitas classes possuem particularidades que necessitavam de métodos extras para que fossem contempladas. Um exemplo dessas particularidades são as diferentes estratégias de pagamento associadas com a classe *Payment* da camada de negócios. A classe *PaymentManager* possui métodos específicos para tratar da configuração dessas estratégias. Outro exemplo está na associação entre os recursos e seus tipos, implementada na classe *QualifiableObject* por meio de métodos que indicam quais são as classes dos tipos de recursos. *QualifiableObjectManager* precisa acessar esses métodos para poder configurar os dados dos tipos dos recursos.

Para exemplificar como é realizada a manipulação de objetos das classes da camada de negócios de um sistema instanciado com base no framework GRENJ, considere uma classe *Cliente* que estende *DestinationParty* que, por sua vez, estende *StaticObject*. *Cliente* possui os atributos *cpf* e *telefone*, além de *idCode* e de *description* que são herdados indiretamente de *StaticObject*, como mostra a Figura 4.12 (a). Como *DestinationParty* não possui atributos, nem outro tipo de particularidade, não existe um *manager* específico para essa classe. Assim, *StaticObjectManager* é utilizada para manipular objetos da classe *Cliente*. A Figura 4.12(b) ilustra os passos para a criação de um objeto da classe *Cliente* e para a atribuição de seus valores na seguinte ordem: 1) um objeto de *StaticObjectManager* recebe uma mensagem com o nome da classe *Cliente* e os nomes e os valores de seus atributos; 2) o objeto de *StaticObjectManager* cria um objeto da classe *Cliente* com os atributos sem valor; 3) o objeto de *StaticObjectManager* atribui os valores aos atributos *idCode* e *description*, provenientes da classe

StaticObject, por meio de seu método *setFields*; 4) o objeto de *StaticObjectManager* usa o método *setFields* de sua superclasse *ObjectManager* para atribuir os valores dos atributos *cpf* e *telefone* da classe *Cliente*.

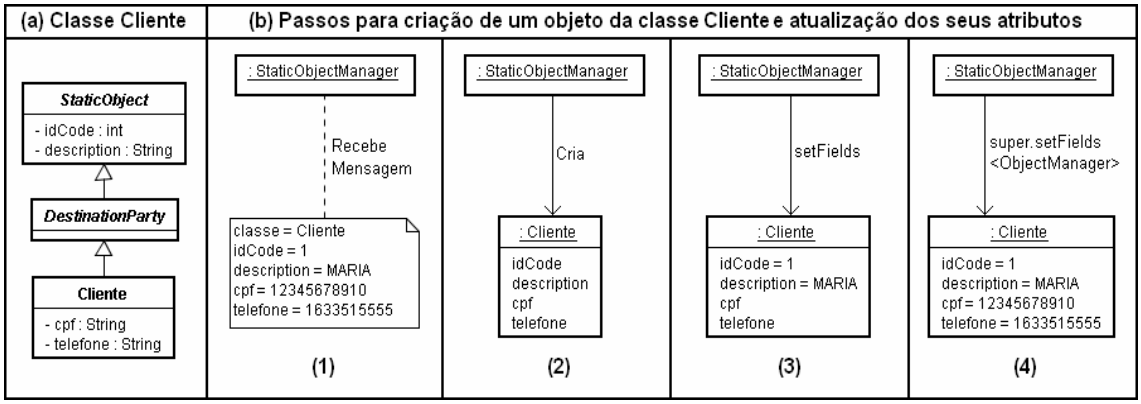

Figura 4.12. Exemplo de manipulação de um objeto da classe *Cliente*.

4.3.3. Finalização da Camada de Interface Gráfica do Framework GRENJ

A finalização da construção da camada de interface gráfica do framework GRENJ foi realizada com a sua instanciação para o sistema de locadora de DVDs, que foi apresentado na Seção 3.3.1. Essa instanciação permitiu testar o funcionamento geral tanto do GRENJ quanto do framework Guiwe.

Além das classes que montam a tela inicial do sistema com cabeçalho, menu principal, página inicial do painel principal e rodapé, foram construídas dez classes para montar os formulários, painéis, tabelas e relatórios do sistema da locadora de DVDs. A Tabela 4.5 contém a relação das classes construídas para a camada de interface gráfica desse sistema e suas classes correspondentes na camada de negócios.

Tabela 4.5. Classes específicas do sistema para uma locadora de DVDs.

Classes da Camada de Negócios	Classes da Camada de Interface Gráfica	Tipo de Interface
Categoria	*CategoriaFormServlet*	Formulário
Genero	*GeneroFormServlet*	Formulário
Filme	*FilmeFormServlet*	Formulário
DVD	*DVDFormServlet, DVDTableServlet*	Formulário, Tabela
Locacao	*LocacaoFormServlet, LocacaoTableServlet*	Formulário, Tabela
Cliente	*ClienteFormServlet*	Formulário
ItemLocacao	*ItemPanelServlet*	Painel
–	*LocacaoReportServlet*	Relatório

A Figura 4.13 mostra o código da classe *FilmeFormServlet* com quatro métodos: o primeiro indica para qual classe da Camada de Negócio o formulário é montado; o

segundo é opcional e especifica um título para o formulário; o terceiro retorna uma lista que contém os rótulos dos campos; e o quatro indica quais são os campos dos formulários. A superclasse de *FilmeFormServlet*, *ResourceFormServlet*, já inclui os campos para os atributos *idCode* e *description*, que foram herdados de suas superclasses, e para os tipos *Categoria* e *Genero* da classe *Filme*. Portanto, só é necessária uma instrução para acrescentar o campo de texto para o atributo *ano*, que é específico da classe *Filme*. Os códigos das demais classes que constroem formulários são semelhantes ao de *FilmeFormServlet*.

```
public class FilmeFormServlet extends QualifiableFormServlet {

   @Override
   public Class getFormClass() {
       return Filme.class;
   }

   @Override
   public String getFormTitle() {
       return "Formulário de Filmes";
   }

   @Override
   public String[] getFieldLabels() {
       return new String[] { "Código:", "Título:", "Categoria:", "Gênero", "Ano:" };
   }

   @Override
   public void mountHTML() {
       super.mountHTML();
       textField( "ano" );
   }

}
```

Figura 4.13. Código da classe *FilmeFormServlet*, que monta um formulário para a classe *Filme*.

A classe *DVDTableServlet* constrói uma tabela que carrega todos os registros armazenados na base de dados que estão relacionados com a classe *DVD*. Essa classe contém apenas dois métodos: *getDataList* e *getTableLabels*. O primeiro fornece o conteúdo da tabela, que pode ser carregado a partir do método *loadAll* de uma das classes *manager* da camada de controle. O método *getTableLabels* define o cabeçalho das colunas da tabela. Além dos métodos descritos na classe *FilmeFormServlet*, *DVDFormServlet* contém um método que indica que seu formulário deve conter a tabela montada por *DVDTableServlet*. A Figura 4.14 contém o código da classe *DVDTableServlet*. O código de *LocacaoTableServlet* é semelhante ao dessa classe.

```
public class DVDTableServlet extends TableServlet {

   @Override
   public ArrayList<ArrayList<String>> getDataList() {
       String[] fields = { "resource", "code", "location", "status" };
       return ResourceInstanceManager.loadAll( "", DVD.class, Movie.class, fields );
   }

   @Override
   public String[] getTableLabels() {
       return new String[] { "Filme", "Código", "Localização", "Status" };
   }
}
```

Figura 4.14. Código da classe *DVDTableServlet*.

Os métodos da classe *ItemPanelServlet*, que constrói um painel, se assemelham aos métodos existentes nas classes que constroem formulários e tabelas. Isso ocorre porque um painel é um pequeno formulário que contém uma tabela. A classe *LocacaoReportServet* também possui métodos que se assemelham aos de um formulário e aos de uma tabela. Porém, essa classe inclui um método, chamado *getReportQuery*, que define o código SQL que busca na base de dados os registros que compõem a tabela do relatório. O relatório montado por essa classe contém um campo de texto na qual o usuário do sistema pode informar uma data. Se esse campo estiver vazio, o relatório mostra todas as locações de DVDs registradas na base de dados do sistema. Caso uma data seja informada, o relatório carrega somente os registros das locações referentes a essa data. Portanto, o código da SQL pode variar de acordo com o preenchimento, ou não, do campo da data. O método *getParams*, proveniente da superclasse *ReportServlet*, é utilizado para obter uma lista com os nomes e valores dos campos de um relatório. A Figura 4.15 contém o código do método *getReportQuery* da classe *LocacaoReportServlet*.

```
@Override
public String getReportQuery() {

    String query = "SELECT number, date, description FROM MovieRental, Client " +
        "WHERE destinationParty = idCode";

    if ( !getParams().isEmpty() && !getParams().get( "date" ).equals( "" ) ) {
        query += " AND date = '" + getParams().get( "date" ) + "'";
    }

    return query;

}
```

Figura 4.15. Código do método *getReportQuery* da classe *LocacaoReportServlet*.

O Formulário de Locações é construído pela classe *LocacaoFormServlet*. Essa classe utiliza o método *getFormClassTableServlet* para inserir nesse formulário uma tabela que é criada pela classe *LocacaoTableServlet* e lista todas as locações registradas no sistema. O método *getTransItemPanelServlet* também é utilizado para inserir no Formulário de Locações o painel dos itens da locação, que é criado pela classe *ItemPanelServlet*. Para registrar uma locação, é necessário preencher os campos e clicar no botão de gravação existente no menu do formulário. Com a locação registrada, os campos do painel de itens da locação podem ser preenchidos para que seus dados sejam adicionados à locação. Os itens da locação são apresentados na tabela do painel. A Figura 4.16 apresenta a tela do sistema para um locadora de DVDs com o Formulário de Locações carregado no painel principal, em que são destacados: a) a tabela de locações construída pela classe *LocacaoTableServlet* e b) o painel de itens da locação construído pela classe *ItemPanelServlet*.

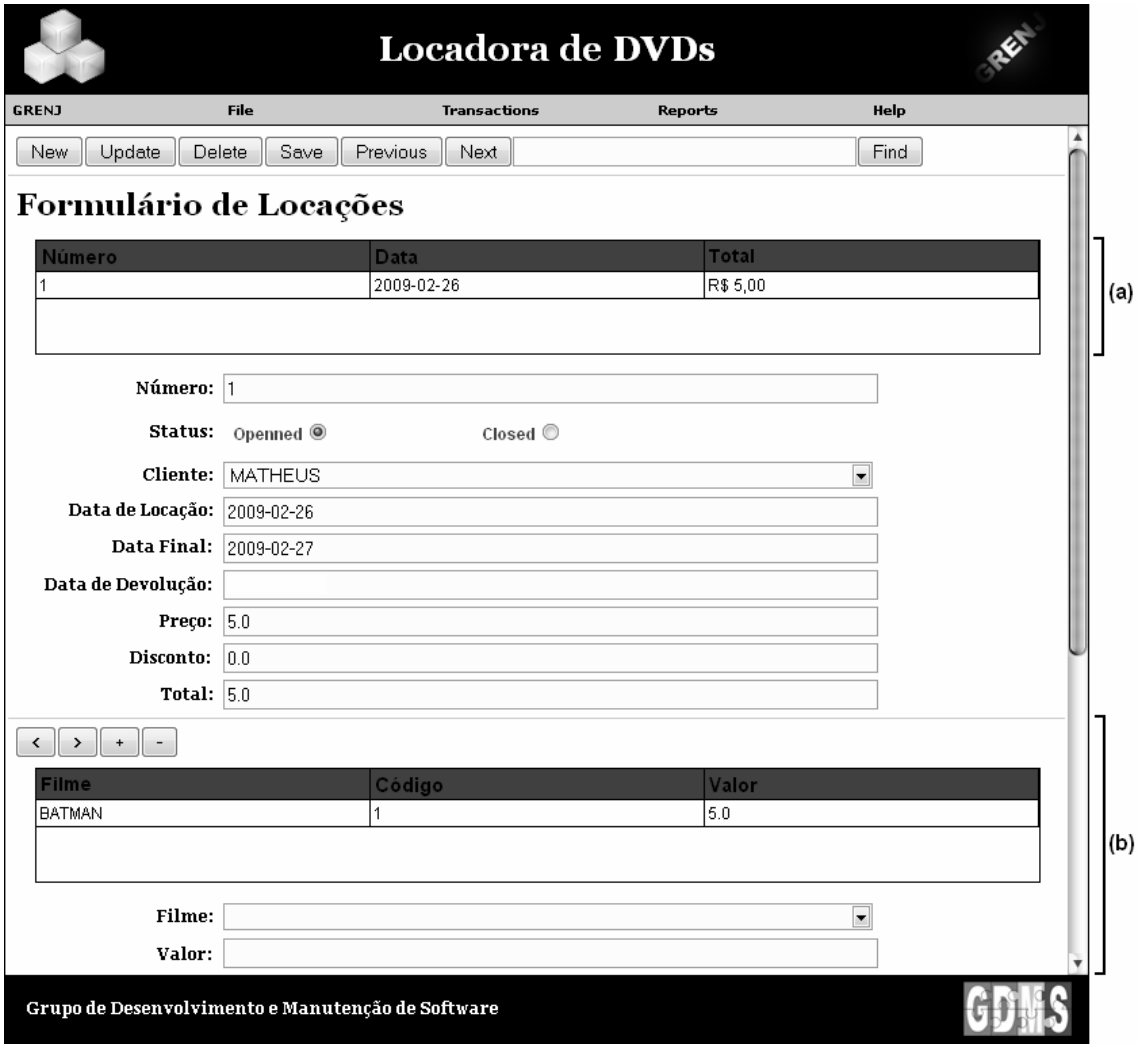

Figura 4.16. Apresentação do sistema para uma locadora de DVDs com o formulário de locação.

4.4. Considerações Finais

O framework Guiwe pode ser utilizado para apoiar a construção da interface gráfica web de um sistema de informação ou EAF. Seu número reduzido de classes o torna simples de ser aprendido e de ser utilizado. Contudo, apesar de sua simplicidade, o framework Guiwe permite que suas classes sejam estendidas, o que lhe garante flexibilidade no número de operações, tipos de dados e formulários contemplados. Seus arquivos CSS e de script podem ser modificados para que a apresentação de sua interface gráfica seja aperfeiçoada e componentes gráficos com mais recursos sejam adicionados. As mensagens retornadas ao usuário estão armazenadas em um arquivo de propriedades e podem ser customizadas ou traduzidas para outras línguas.

Durante o desenvolvimento do framework Guiwe e da interface gráfica do framework GRENJ, a abordagem utilizada para a criação das listas de casos de testes do TDD reduziu o número de artefatos de documentação. Pois as listas eram transformadas nas próprias classes do código-fonte dos frameworks. Além de descrever os métodos, documentação *javadoc* define o que os testes devem verificar e quais as interfaces do projeto. Essa abordagem tornou o trabalho de construção das classes mais eficiente, uma vez que não era necessário consultar e manter a consistência de muitos artefatos.

Apesar de ser utilizado na instanciação de sistemas voltados para a web, a maior parte do framework GRENJ está implementado em linguagem Java. Dessa forma, há redução do número de tecnologias que o desenvolvedor necessita ter domínio para utilizá-lo, bem como permite que muitos dos defeitos existentes no código do sistema em desenvolvimento sejam detectados em tempo de compilação. Diferentemente do que ocorreria com o uso de linguagens interpretadas como, por exemplo, JSP e HTML.

A instanciação de um sistema com base no framework GRENJ é realizada por meio da extensão de suas classes, o que caracteriza o reúso caixa branca. Em cada camada, os métodos que precisam ser sobrescritos se repetem em diversas classes. Essa característica representa um padrão idiomático inerente ao framework e facilita o trabalho do desenvolvedor. Além disso, tanto na camada de negócios quanto na de interface gráfica, a maioria dos métodos sobrescritos possui uma ou duas instruções. Esse número reduzido de instruções representa um aumento na eficiência de desenvolvimento e uma garantia de melhor qualidade do sistema, uma vez que um número menor de instruções reduz a probabilidade de inserção de erros.

O Capítulo 5 apresenta a configuração do framework GRENJ em um gerador de aplicações, o que permite que sistemas baseados nesse framework sejam desenvolvidos a partir de uma interface gráfica e sem a necessidade de implementação de código.

5 Criação de um Wizard para o Framework GRENJ

5.1. Considerações Iniciais

Com o advento da camada de interface gráfica, o framework GRENJ passa a apoiar o desenvolvimento de sistemas web completos. Por ser um framework caixa branca, sua instanciação é realizada por meio da extensão de suas classes e, desse modo, as classes construídas pelo desenvolvedor sobrescrevem métodos das classes estendidas do framework para indicar as características específicas do sistema que está sendo desenvolvido.

O domínio de gestão de recursos de negócios provê apoio ao desenvolvimento de sistemas que tratam de transações de aluguel, comercialização e manutenção de recursos e possui inúmeras variabilidades. No framework GRENJ essas variabilidades são representadas por diversos pontos variáveis. Um desenvolvedor pode sentir dificuldades em saber quais pontos variáveis devem ser configurados e como fazer isso para a instanciação de determinado sistema, principalmente, nas primeiras vezes que utiliza o framework.

Essa dificuldade pode ser amenizada com a utilização de um wizard no qual o desenvolvedor informa as características específicas do sistema que deseja desenvolver e esse sistema é gerado de maneira automatizada. Isso pode ser obtido por meio da configuração de um gerador de aplicações para a instanciação de sistemas baseados no framework GRENJ. Dessa forma, os pontos variáveis do framework são abstraídos e o desenvolvedor necessita conhecer somente a linguagem de padrões GRN.

Entretanto, para não tornar a geração de sistemas baseados no framework GRENJ dependente de um gerador específico, foi desenvolvida uma ferramenta, denominada GRENJ-Gens, que permite ao desenvolvedor registrar diversos geradores de aplicações e, então, selecionar um para instanciar um sistema pertencente ao domínio do gerador selecionado. Essa ferramenta fornece aos geradores os gabaritos que devem ser utilizados para a geração das classes e outros artefatos necessários na instanciação de um sistema baseado no framework GRENJ.

As demais Seções deste Capítulo, que descrevem o processo de criação do wizard para uso do framework GRENJ, estão organizadas da seguinte forma: na Seção 5.2 é descrita a criação de uma LMA do framework GRENJ com a utilização do gerador de aplicações configurável Captor; na Seção 5.3 é tratada a construção dos gabaritos XSL e na Seção 5.4 é abordada a construção da ferramenta GRENJ-Gens; na Seção 5.5 são apresentadas as considerações finais deste Capítulo.

5.2. Criação de uma LMA do Framework GRENJ

Uma LMA define as características comuns e variáveis dos sistemas que pertencem a determinando domínio (Weiss e Lai, 1999). A criação de uma LMA do framework GRENJ foi realizada com o apoio do gerador de aplicações configurável Captor (Shimabukuro Junior, 2006). No Captor, a criação de uma LMA referente a um domínio é realizada por meio da construção de formulários, cuja estrutura é persistida em um documento XML (ver Seção 3.4).

Os formulários da LMA do framework GRENJ foram construídos com base nos padrões da linguagem de padrões GRN. Com exceção do primeiro formulário, cada um dos demais representa uma classe da camada de negócios do framework GRENJ como, por exemplo, *Resource*, *DestinationParty*, *ResourceMaintenance*, entre outras. Durante a fase de engenharia da aplicação, esses formulários solicitam ao desenvolvedor as informações que são utilizadas pelos gabaritos para gerar as classes das camadas de modelo e de interface gráfica específicas da aplicação que está sendo desenvolvida.

A construção dos formulários e dos gabaritos XSL foi realizada durante as iterações do projeto e implementação das classes da interface gráfica do framework GRENJ. Ou seja, cada iteração compreendeu a construção das classes da camada de interface gráfica do framework, dos formulários do gerador de aplicações Captor e dos gabaritos XSL de um padrão da GRN.

O primeiro formulário construído para o domínio do framework GRENJ está relacionado com as informações gerais do sistema a ser gerado. Possui campos para o nome, o pacote e a uma descrição do sistema. Essas informações são utilizadas na geração das classes que montam a página inicial da interface gráfica do framework GRENJ e nas ações executadas pelo Captor antes e depois da geração das classes. A Figura 5.1 apresenta o formulário inicial para geração de sistemas baseados no framework GRENJ.

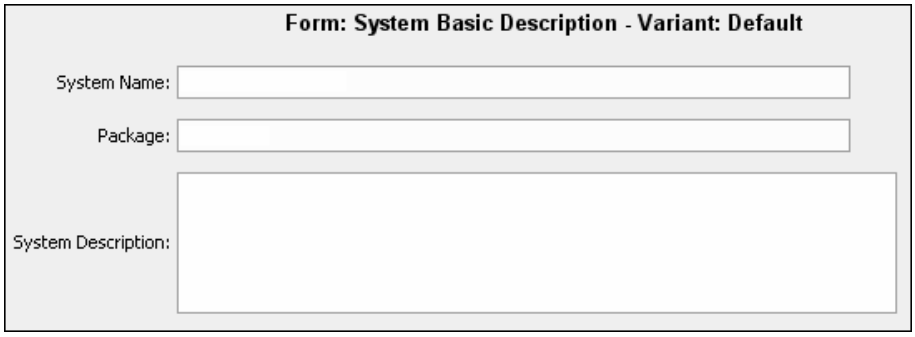

Figura 5.1. Formulário inicial para geração de sistemas baseados no framework GRENJ.

O formulário que solicita os dados de um recurso contém um campo de texto para o nome da classe e outro para o título do formulário da interface gráfica do sistema, além de uma tabela que solicita os tipos e nomes dos atributos adicionais. Os dados desse formulário são utilizados para criação das classes que estendem *Resource* na camada de modelo e *QualifiableFormServlet* e *TableServlet* na camada de interface gráfica. A Figura 5.2 apresenta o formulário relativo à identificação do recurso. Os formulários relativos aos tipos de recursos do framework GRENJ são semelhantes ao formulário relativo ao recurso, uma vez que suas classes são extensões de *QualifiableObject.*

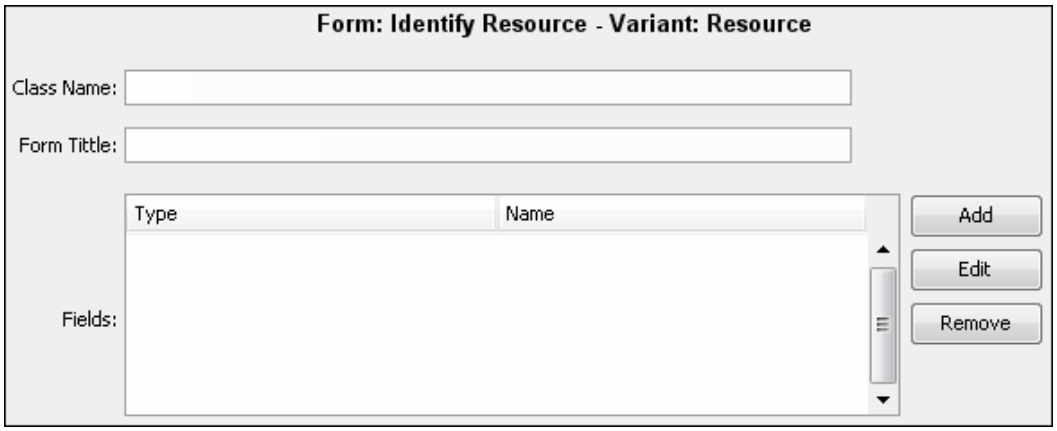

Figura 5.2. Formulário relativo à identificação do recurso.

Como a aplicação dos dois primeiros padrões da GRN é obrigatória, os formulários relativos à identificação e à quantificação do recurso são carregados automaticamente quando o projeto de um novo sistema é iniciado. Cabe ao desenvolvedor preencher os seus campos e selecionar quais os demais formulários a serem carregados.

Para o segundo padrão da GRN, Quantificar o Recurso, foi criado um formulário que possui quatro variantes. A primeira variante indica que o recurso é simples e

nenhuma classe é gerada para a sua quantificação. A segunda variante indica que o recurso é instanciável e, portanto, solicita os dados para a geração de subclasses de *ResourceInstance*, *ResInsFormServlet* e *TableServlet*. As terceira e quarta variantes indicam que o recurso é mensurável ou em lote, respectivamente, e solicitam dados para subclasses de *MeasureUnity*. O uso de uma variante ou outra desse formulário influencia tanto na geração das classes do padrão de quantificação do recurso quanto no conteúdo de alguns métodos da classe que representa o recurso.

Os padrões relativos às transações do sistema que está sendo gerado são contemplados por formulários que podem ser carregados a partir do formulário de um recurso. Em geral, os formulários das transações solicitam o nome da classe da transação, o título do formulário da interface gráfica do sistema, os atributos adicionais, a quantificação do recurso e o nome das classes associadas. A Figura 5.3 contém o formulário relativo a uma transação de aluguel. A única diferença desse formulário em relação aos demais formulários das transações é a presença do campo *Has Associated Sale*, que indica se podem existir, ou não, transações de venda associadas às transações de aluguel. Além disso, o formulário da transação de aluguel pode ser seguido pelo formulário de taxa de multa.

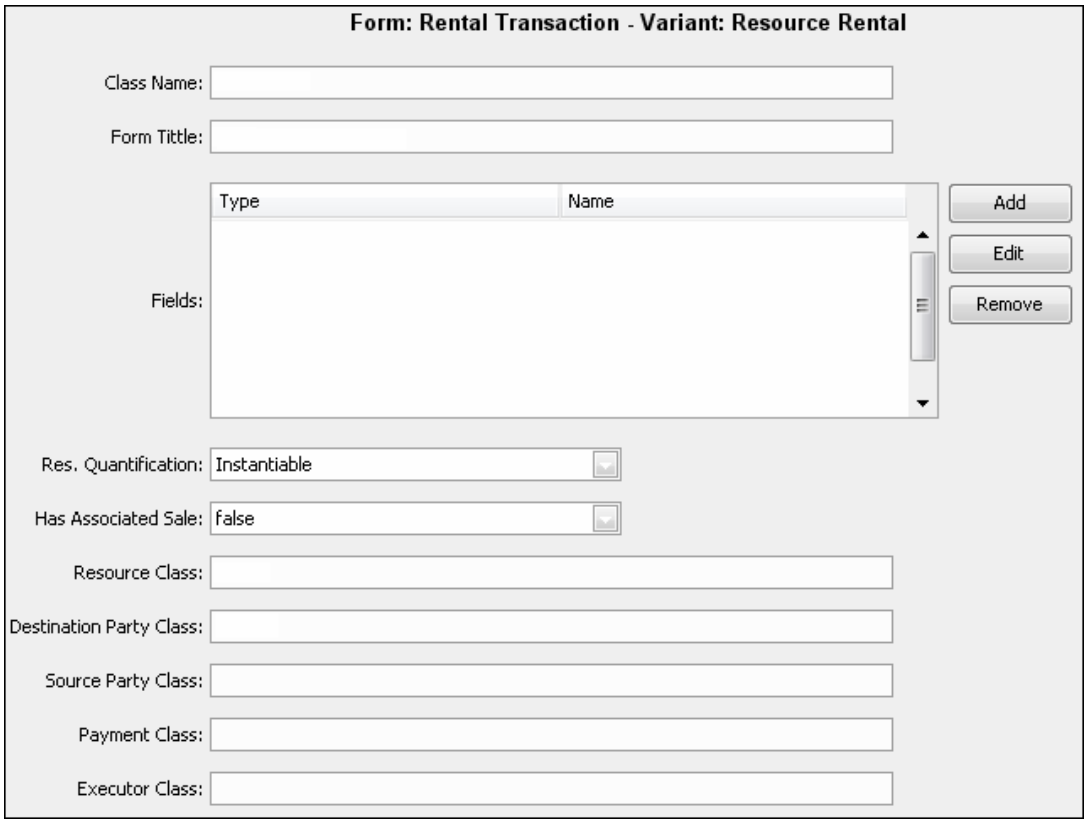

Figura 5.3. Formulário relativo a uma transação de aluguel.

Os formulários relacionados com a origem e com o destino dos recursos de uma transação solicitam os mesmos dados que o formulário relativo à identificação do recurso. Porém, suas informações são utilizadas para gerar subclasses de *SourceParty* (origem) e *DestinationParty* (destino) na camada de negócios do sistema e de *StaticFormServlet* na camada de interface gráfica do sistema.

Os formulários das transações, com exceção do de cotação de recursos, podem ser seguidos por um formulário que representa o padrão Itemizar a Transação do Recurso da GRN. Esse formulário indica que devem ser geradas subclasses de *TransactionItem* na camada de negócios e de *TransItemPanelServlet* na camada de interface gráfica do sistema que está sendo desenvolvido. O formulário equivalente à itemização dos recursos de uma cotação de venda é o de itens a cotação, que está relacionado com as classes *QuotationItem* e *QuotItemPanelServlet*.

O formulário relativo ao pagamento de uma transação solicita o nome da classe, os atributos adicionais, o nome da classe da transação, as estratégias de pagamento contempladas pelo sistema e qual dessas estratégias é a principal. As estratégias que podem ser contempladas pelo sistema são baseadas nas subclasses de *PaymentStrategy* existentes na camada de negócios do framework GRENJ. A Figura 5.4 contém o formulário relativo ao pagamento de uma transação, cujas informações são utilizadas para a geração de subclasses de *Payment* e *PaymentPanelServlet*.

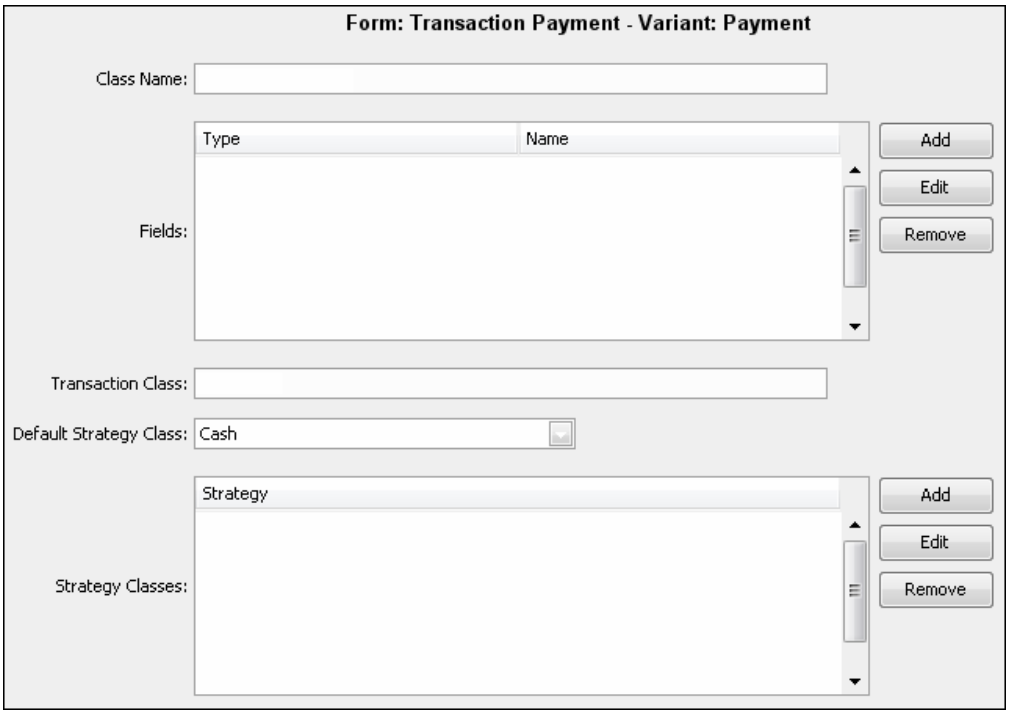

Figura 5.4. Formulário relativo ao pagamento de uma transação.

O formulário relativo ao executor de uma transação é semelhante ao formulário de identificação do recurso, exceto por conter um campo de seleção que indica o recebimento, ou não, de comissão. Assim como o formulário de pagamento, o formulário do executor da transação pode ser carregado após os formulários das transações.

Os formulários relacionados com os dois últimos padrões da GRN, Identificar as Tarefas da Manutenção e Identificar as Peças da Manutenção, podem ser carregados após o de uma transação de manutenção. Respectivamente, estão relacionados com as classes *MaintenanceTask* e *MaintenancePart* da camada de negócios e *MainTaskPanelServlet* e *MainPartPanelServlet* da camada de interface gráfica do framework GRENJ. O padrão Identificar as Peças da Manutenção indica, ainda, a necessidade de mais um formulário para a geração da classe que define as peças da manutenção, que está relacionado com as classes *Part* e *PartFormServlet* do framework GRENJ. A Figura 5.5 apresenta o formulário para geração da classe que define as tarefas de uma manutenção. O formulário relativo à classe *MaintenancePart* difere desse somente por não conter um campo que solicita o nome da classe do executor.

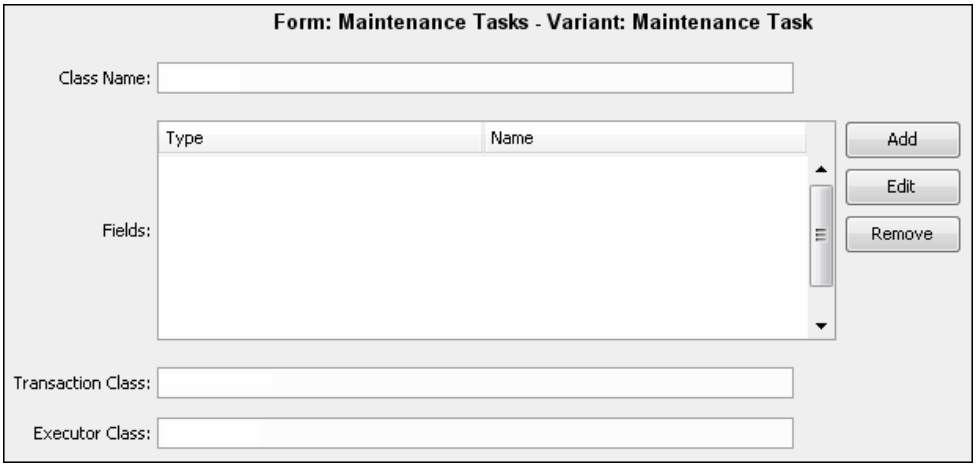

Figura 5.5. Formulário relativo ao padrão Identificar as Tarefas da Manutenção.

5.3. Construção dos Gabaritos XSL

No Captor, quando o desenvolvedor inicia o projeto de um sistema baseado no framework GRENJ e preenche os dados dos formulários, é gerada uma instância da LMA do framework GRENJ com as informações específicas do sistema que está sendo desenvolvido. O mapeamento dos dados dessa instância da LMA sobre os gabaritos XSL resulta na geração das classes e de outros artefatos específicos do sistema.

Os gabaritos XSL foram organizados em pacotes com base na mesma hierarquia do framework GRENJ e possuem os mesmos nomes das classes que representam. Para cada padrão da GRN, eram construídos os gabaritos para as classes das camadas de negócios e de interface gráfica e o script da base de dados era atualizado para contemplar essas classes. O arquivo *rules* também foi atualizado para associar os formulários da LMA com os gabaritos sobre os quais deviam ser mapeados.

Um gabarito XSL contém código imutável do artefato a ser gerado e instruções que selecionam o conteúdo dos elementos de uma instância da LMA. Variáveis foram criadas para armazenar os valores dos campos dos formulários. Caso os formulários sejam modificados, somente o conteúdo das variáveis necessita ser modificado, ao invés das instruções em que são utilizadas. A Figura 5.6 contém o trecho de código XSL que declara uma variável, chamada *className*, que armazena o nome de uma classe informado no formulário associado a esse gabarito.

```
<xsl:variable name="className">
    <xsl:value-of select="/formsData/current/form/data/textatt[@name='className']"/>
</xsl:variable>
```

Figura 5.6. Declaração de uma variável em que armazena o nome de uma classe.

Os nomes das classes informados nos formulários da LMA do framework GRENJ são utilizados na declaração das classes e de seus construtores. A Figura 5.7 apresenta um exemplo em que o desenvolvedor informa que o nome da classe que representa o recurso é *Filme*. Assim, na camada de negócios é gerada uma classe *Filme*, que estende *Resource*, e na camada de interface gráfica são geradas *FilmeFormServlet*, que estende *ResourceFormServlet*, e *FilmeTableServlet*, que estende *TableServlet*.

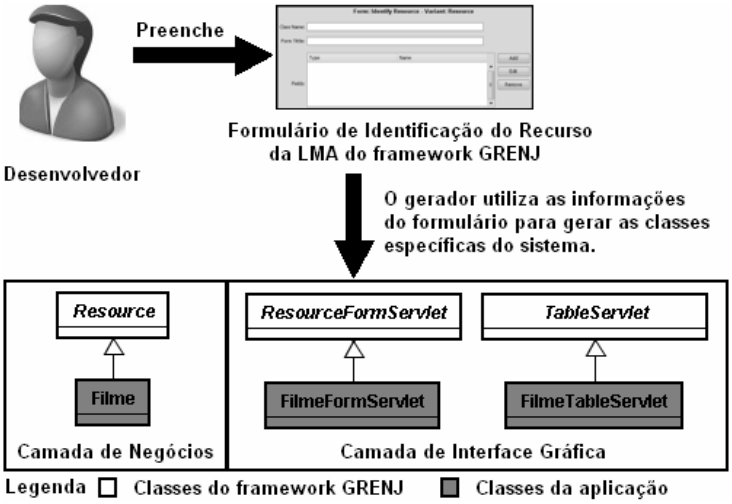

Figura 5.7. Exemplo de geração das classes relacionadas com o recurso do negócio.

Nos gabaritos referentes às classes da camada de negócios, os tipos e nomes dos atributos adicionais, que são informados em tabelas existentes nos formulários da LMA, são utilizados para a geração das instruções de declaração desses atributos, das instruções contidas nos métodos construtores e de seus métodos *setters* e *getters*. Nos gabaritos referentes às classes da camada de interface gráfica, os tipos e nomes dos atributos adicionais são utilizados na montagem dos campos dos formulários da tela do sistema, bem como na montagem do cabeçalho das tabelas. A Figura 5.8 contém um trecho do código do gabarito que dá origem a uma subclasse de *Resource*. Esse trecho de código é responsável pela declaração da classe, de seus atributos e de um de seus construtores. As instruções em negrito correspondem ao conteúdo imutável do gabarito.

```
public class <xsl:value-of select="$className"/> extends Resource {
   <xsl:for-each select="/formsData/current/form/data/table[@id='classFields']/row">
   private <xsl:value-of select="col[@number='0']/value"/>
   <xsl:text> </xsl:text> <xsl:value-of select="col[@number='1']/value"/>;</xsl:for-each>

   public <xsl:value-of select="$className"/>() {

       super();
       <xsl:for-each select="/formsData/current/form/data/table[@id='classFields']/row">
       <xsl:text>
       </xsl:text>
       <xsl:value-of select="col[@number='1']/value"/><xsl:text> = </xsl:text>
       <xsl:if test="col[@number='0']/value='String'">"";</xsl:if>
       <xsl:if test="col[@number='0']/value='int'">0;</xsl:if>
       <xsl:if test="col[@number='0']/value='char'">'\0';</xsl:if>
       <xsl:if test="col[@number='0']/value='float'">0.0;</xsl:if>
       <xsl:if test="col[@number='0']/value='double'">0.0D;</xsl:if>
       <xsl:if test="col[@number='0']/value='boolean'">true;</xsl:if>
       <xsl:if test="col[@number='0']/value='Money'">new Money( 0.0D );</xsl:if>
       <xsl:if test="col[@number='0']/value='Date'">Period.getToday();</xsl:if>
       </xsl:for-each>
       super.setChanged( false );

   }
```

Figura 5.8. Código do gabarito que origina uma subclasse de *Resource*.

Os valores dos campos que aparecem nos formulários da LMA para solicitar os nomes de outras classes são utilizados para compor as instruções dos métodos que realizam as associações entre as classes do sistema que está sendo gerado. Por exemplo, no formulário relacionado com a transação de aluguel, existe um campo que solicita o nome da classe que é o executor da transação. O conteúdo desse campo é utilizado pelo gabarito das subclasses de *BusinessResourceTransaction* para a geração dos métodos *getExecutorClass* e *hasExecutor*. A Figura 5.9 contém o trecho do código XSL correspondente a esses métodos.

```
@Override
public Class&lt;? extends TransactionExecutor &gt; getExecutorClass() {
    return <xsl:choose>
        <xsl:when test="$executorClass=''">null</xsl:when>
        <xsl:otherwise><xsl:value-of select="$executorClass"/>.class</xsl:otherwise>
    </xsl:choose>;
}

@Override
public boolean hasExecutor() {
    return <xsl:choose>
        <xsl:when test="$executorClass=''">false</xsl:when>
        <xsl:otherwise>true</xsl:otherwise>
    </xsl:choose>;
}
```

Figura 5.9. Trecho do código de um gabarito que gera os métodos *getExecutorClass* e *hasExecutor*.

Além dos gabaritos referentes às classes das camadas de negócios e de interface gráfica, foram criados outros que correspondem à geração de artefatos necessários ao funcionamento dos sistemas baseados no framework GRENJ. Um desses gabaritos é responsável pela geração do script da base de dados, que gera uma tabela para cada classe da camada de negócios por meio de uma função chamada *createTable*. A Figura 5.10 contém o código XSL dessa função.

```
<xsl:template name="createTable">
    <xsl:param name="name"/>
    <xsl:param name="rows"/>
    <xsl:param name="defaultFields"/>
DROP TABLE IF EXISTS <xsl:value-of select="$name"/>;
CREATE TABLE <xsl:value-of select="$name"/> (
<xsl:value-of select="$defaultFields"/><xsl:if test="$rows!=''">
   <xsl:for-each select="$rows/row">,
   <xsl:value-of select="col[@number='1']/value"/>
      <xsl:if test="col[@number='0']/value='String'"> VARCHAR(50)</xsl:if>
      <xsl:if test="col[@number='0']/value='int'"> INTEGER</xsl:if>
      <xsl:if test="col[@number='0']/value='char'"> CHAR</xsl:if>
      <xsl:if test="col[@number='0']/value='float'"> FLOAT</xsl:if>
      <xsl:if test="col[@number='0']/value='double'"> FLOAT</xsl:if>
      <xsl:if test="col[@number='0']/value='boolean'"> BOOLEAN</xsl:if>
      <xsl:if test="col[@number='0']/value='Money'"> FLOAT</xsl:if>
      <xsl:if test="col[@number='0']/value='Date'"> DATE</xsl:if>
   </xsl:for-each></xsl:if>
);
</xsl:template>
```

Figura 5.10. Código da função *createTable* do gabarito que gera o script da base de dados.

Os demais gabaritos são responsáveis pela geração da pagina inicial do sistema baseado no framework GRENJ, dos arquivos de configuração do Apache Tomcat (The Apache Software Foundation, 2009a) e dos arquivos que definem o sistema gerado como um projeto da IDE NetBeans (Sun Mycrosystems, 2009g). Os arquivos de

configuração do servidor Apache Tomcat gerenciam a comunicação entre as páginas do sistema e a camada de controle. Os arquivos que definem o sistema gerado como sendo um projeto da IDE Netbeans foram criados para fornecer o apoio dessa IDE, caso o desenvolvedor decida realizar customizações específicas sem utilizar o gerador ou acrescentar funções não contempladas pelo framework GRENJ.

Após a criação dos gabaritos, foi realizada a construção dos arquivos de configuração do domínio. O arquivo *rules*, que associa os formulários do wizard com os gabaritos correspondentes, foi construído ao longo da construção dos gabaritos de cada padrão da GRN. Os arquivos *pre-build* e *pos-build*, que definem as ações executadas antes e depois da geração dos artefatos, foram construídos somente no final da configuração do domínio do GRENJ no Captor, pois as ações que deveriam ser realizadas por esses arquivos só poderiam ser conhecidas após a identificação de todos os artefatos. No arquivo *pre-build* do domínio do GRENJ no Captor, foi acrescentada uma instrução que copia a estrutura de arquivos de um projeto de sistema web feito na IDE Netbeans para a pasta do projeto do sistema que está sendo gerado no Captor. Essa estrutura de arquivos se mantém independentemente do sistema que está sendo gerado. No arquivo *pos-build*, foram incluídas instruções que executam as seguintes ações:

1. Alguns arquivos gerados nas pastas *src* e *web* do projeto do sistema são copiados para a pasta *build*, que é a pasta da estrutura de arquivos de um sistema web que contém os arquivos que são necessários para a execução do sistema.
2. Os arquivos das classes da camada de negócios e de interface gráfica, que foram gerados dentro da pasta *src*, são compilados e os arquivos originados dessa compilação são armazenados na pasta *build/web/WEB-INF/classes*.
3. A estrutura de arquivos do projeto do sistema gerado é copiada para a pasta base das aplicações web gerenciadas pelo servidor Apache Tomcat. O local dessa pasta varia de um computador para outro e, por isso, é previamente configurado pela ferramenta GRENJ-Gens, Seção 5.4.
4. O arquivo de contexto do sistema web é copiado para a pasta *catalina/localhost*, dentro do diretório de instalação do servidor Apache Tomcat. O local desse diretório varia de um computador para outro e, por isso, é previamente configurado pela ferramenta GRENJ-Gens, Seção 5.4.
5. O script de criação da base de dados do sistema gerado é executado.

A Figura 5.11 ilustra os passos executados pelo Captor para a geração do projeto de um sistema chamado *Locadora* com a execução das instruções do arquivo *pre-build*, a geração dos artefatos pelo gerador e a execução das instruções do arquivo *pos-build*.

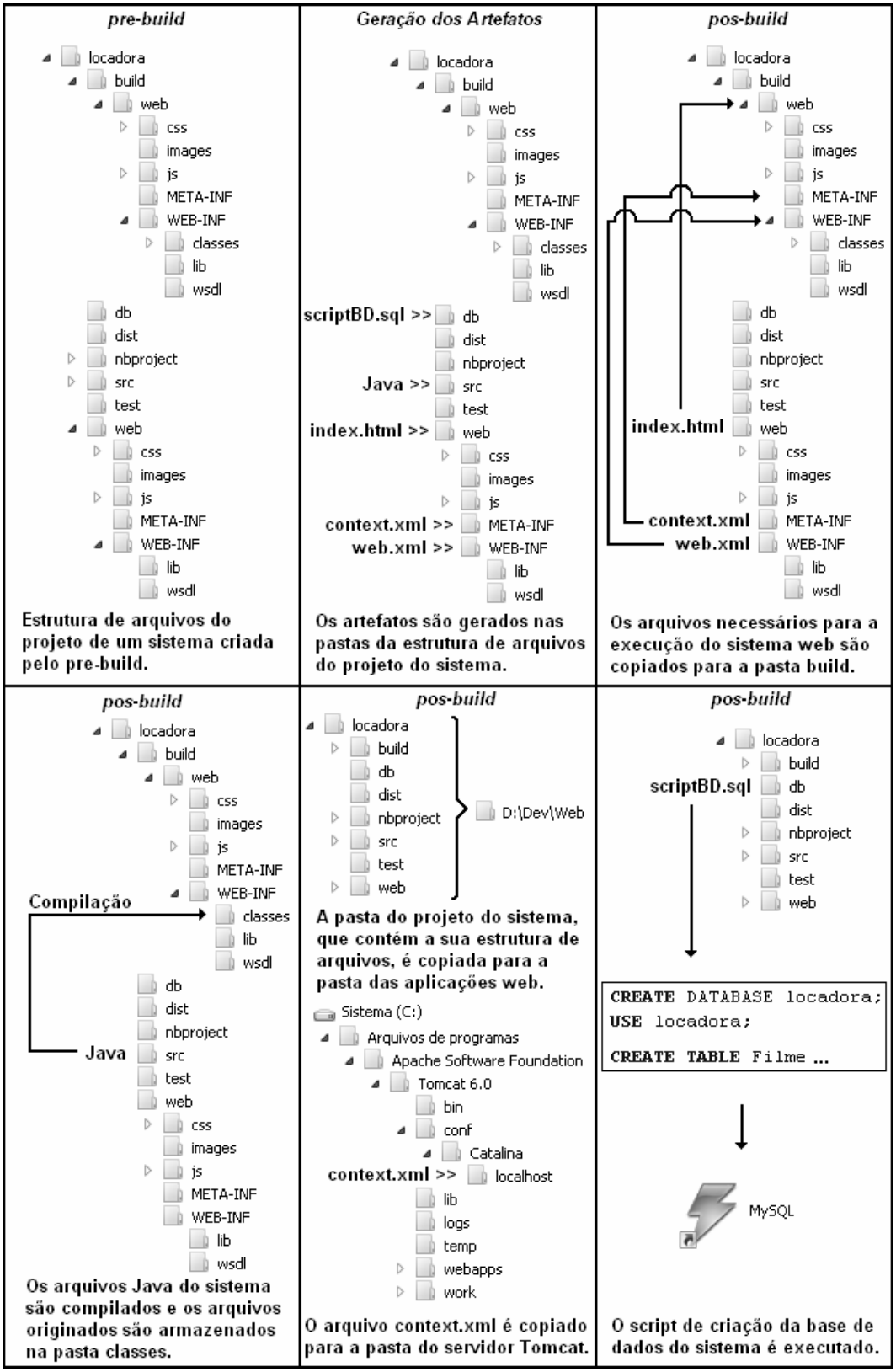

Figura 5.11. Geração de um sistema desenvolvido com o uso do wizard do framework GRENJ.

5.4. Construção da Ferramenta GRENJ-Gens

A criação da LMA, dos gabaritos XSL e dos arquivos de configuração definem a configuração do gerador de aplicações Captor para o domínio do framework GRENJ. Porém, os gabaritos XSL estão relacionados com o domínio para o qual geram artefatos e, não, com o gerador de aplicações configurável que os utiliza. Portanto, podem ser utilizados por outros geradores de aplicações configuráveis que fazem uso desse tipo de gabarito na configuração de seus domínios.

As ações que são executadas após a geração dos artefatos, como a compilação das classes e a criação da base de dados, dependem de outros softwares para serem executadas. As localizações desses softwares devem constar nos arquivos de configurações do domínio, entretanto, podem variar de um computador para outro. Assim, é necessário atualizar essas localizações nos arquivos de configuração a cada instalação do gerador em um computador diferente. Uma solução poderia ser solicitar do desenvolvedor essas informações nos formulários da LMA, contudo, essa abordagem possui duas desvantagens: 1) as localizações dos softwares teriam de ser informados a cada instanciação de um sistema; 2) a solicitação dessas informações vai de encontro à premissa de que, com o uso do gerador, o desenvolvedor precisa conhecer somente a linguagem de padrões GRN para instanciar um sistema baseado no framework GRENJ.

Com o intuito de evitar que o framework GRENJ se torne dependente de um único gerador de aplicações e de resolver o problema da localização dos softwares auxiliares, foi criada uma ferramenta chamada **GRENJ-Gens**. Essa ferramenta permite atualizar os arquivos de configuração com as localizações dos softwares auxiliares por meio de uma interface gráfica e cadastrar diversos geradores de aplicações baseados em gabaritos XSL. Esses geradores devem ser configurados para utilizar os gabaritos fornecidos pelo GRENJ-Gens. Para desenvolver um novo sistema, o desenvolvedor seleciona um dos geradores cadastrados para executá-lo e iniciar um novo projeto de sistema baseado no framework GRENJ.

5.4.1. Arquitetura da Ferramenta GRENJ-Gens

A ferramenta GRENJ-Gens deve armazenar informações sobre a localização do Java Development Kit (JDK), do diretório de instalação do Apache Tomcat e do diretório base das aplicações web, bem como o usuário e a senha da base de dados. Também deve atualizar os gabaritos e arquivos de configuração que dependem dessas informações.

Deve ser possível cadastrar geradores de aplicações a partir do seu nome e da localização do seu arquivo executável. Em alguns casos, o executável do gerador de aplicações é um arquivo Jar e, portanto, deve ser informado o caminho da sua classe principal dentro desse arquivo. Um gerador de aplicações pode ser selecionado na lista de cadastrados e executado. A Figura 5.12 ilustra os componentes da arquitetura da ferramenta GRENJ-Gens.

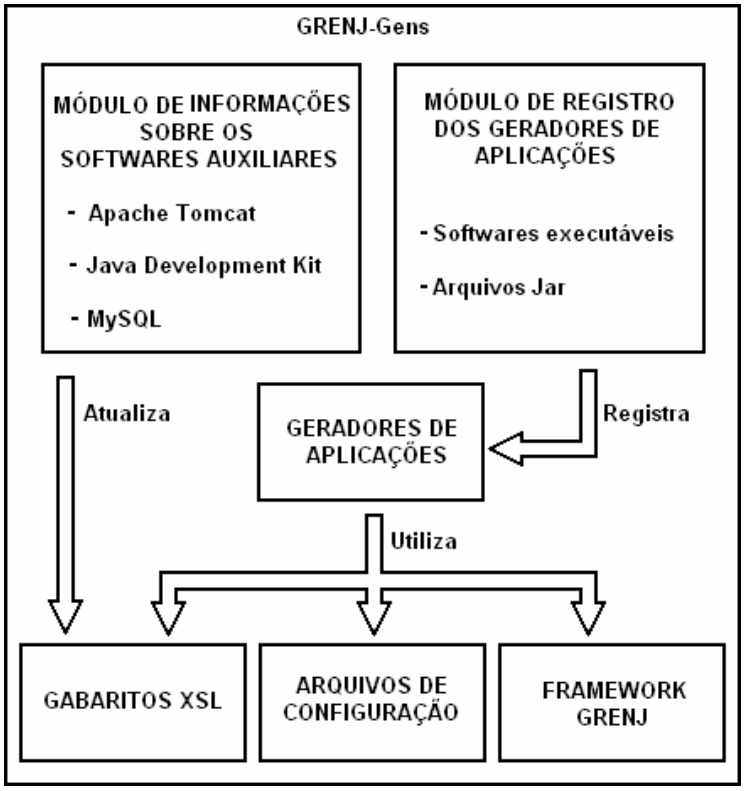

Figura 5.12. Arquitetura da ferramenta GRENJ-Gens.

5.4.2. Projeto e Implementação do GRENJ-Gens

O projeto e a implementação da ferramenta GRENJ-Gens foram realizados com a prática do TDD, com a criação de testes antes da implementação das classes. Por se tratar de um número reduzido de informações, os dados sobre os softwares auxiliares e os geradores foram armazenados em um documento XML chamado *grenjgens*. Uma classe, chamada *Controller*, foi construída para gerenciar a leitura e a escrita desse XML. O GRENJ-Gens possui uma janela principal com um menu que direciona aos painéis de configuração dos dados e de cadastro dos geradores de aplicações. Além das informações dos softwares auxiliares, o painel de configurações permite selecionar a linguagem da ferramenta (inglês ou português) e sua aparência (*Look and Feel*). A Figura 5.13 apresenta o painel de configuração do GRENJ-Gens.

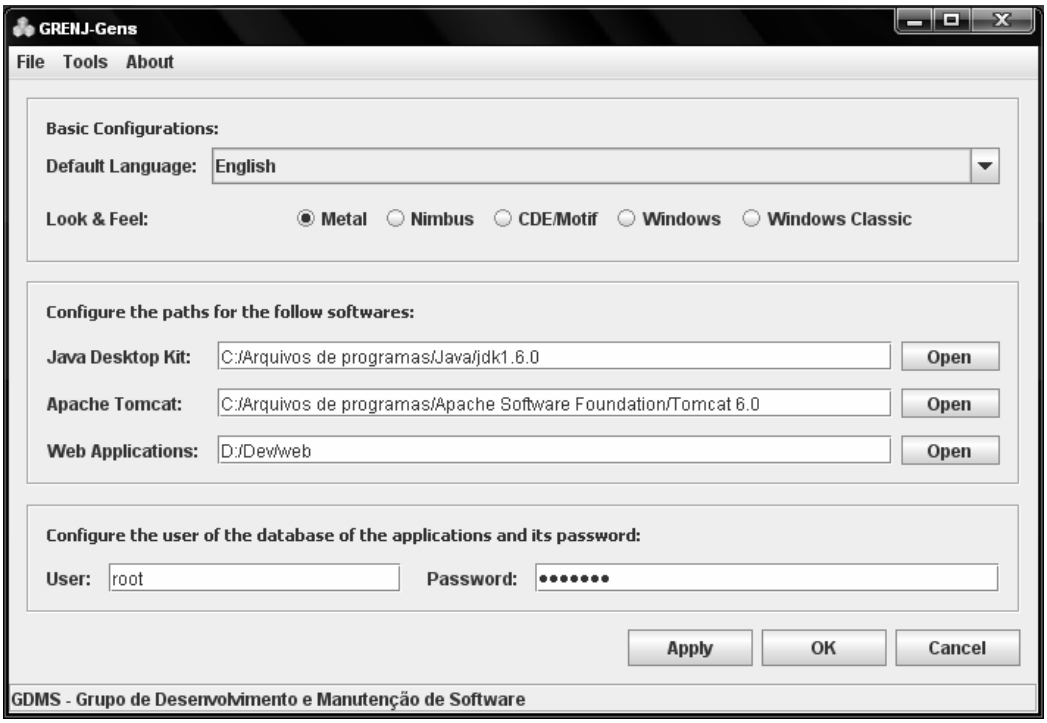

Figura 5.13. Painel de configurações do GRENJ-Gens.

O painel de cadastro de geradores de aplicações permite ao desenvolvedor selecionar e executar um dos geradores cadastrados na ferramenta. Também é nesse painel que geradores de aplicações podem ser adicionados ou removidos do cadastro da ferramenta. A Figura 5.14 apresenta o painel de cadastro de geradores de aplicações do GRENJ-Gens.

Figura 5.14. Painel de cadastro de geradores de aplicações do GRENJ-Gens.

5.4.3. Finalização do GRENJ-Gens

A finalização da ferramenta GRENJ-Gens ocorreu com a criação do diretório da ferramenta, que possui seu arquivo Jar, seu ícone e a pasta *data*, que contém o documento XML *grenjgens*. Para utilizar o GRENJ-Gens, é necessário possuir esse diretório no computador. Os dados dos softwares necessários para a execução dos sistemas baseados no framework GRENJ, como o JDK e o Apache Tomcat, foram registrados na ferramenta. O gerador de aplicações Captor foi cadastrado e seus arquivos/diretórios foram inseridos no diretório da ferramenta GRENJ-Gens. Foram realizados estudos de caso da ferramenta com o desenvolvimento de sistemas de informações baseados no framework GRENJ com o apoio do gerador de aplicações Captor. Esses estudos de caso são descritos no Capítulo 6 deste trabalho.

5.5. Considerações Finais

A Figura 5.15(a) ilustra que, com a construção das camadas de controle, de interface gráfica e do wizard, a arquitetura do framework GRENJ passa a ser composta de cinco camadas: persistência, negócios, controle, interface gráfica e wizard. A Figura 5.15(b) apresenta a arquitetura de uma aplicação instanciada do framework GRENJ. A camada do wizard tem como função facilitar a instanciação de aplicações com base no framework GRENJ, porém não faz parte das aplicações instanciadas.

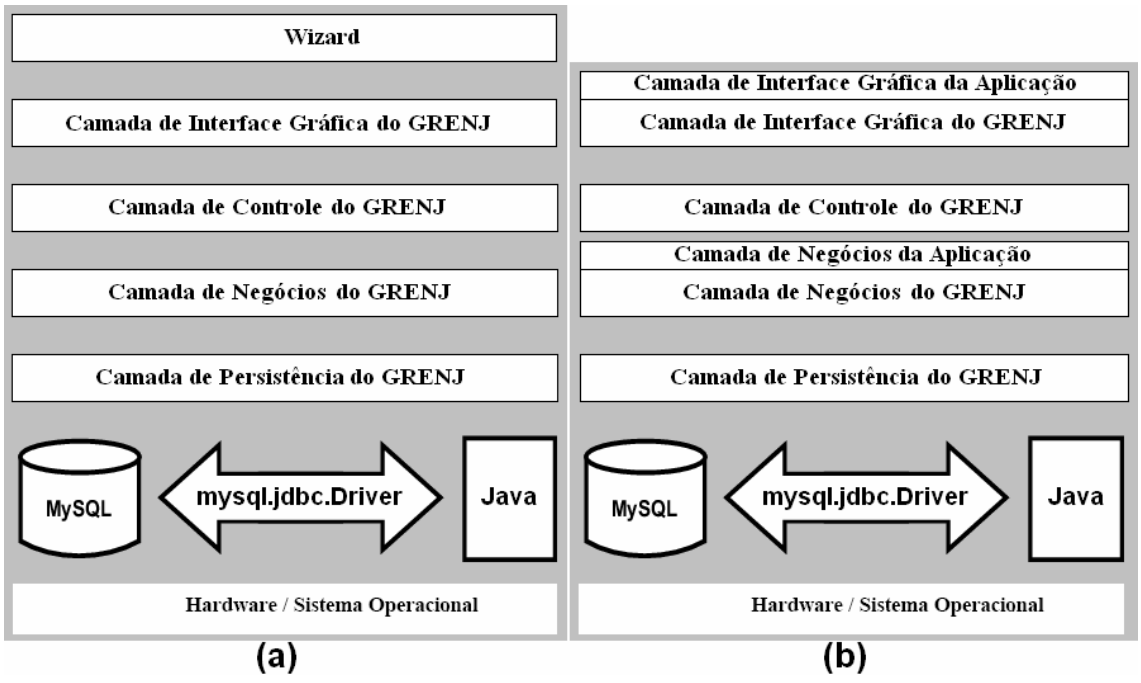

Figura 5.15. A arquitetura do framework GRENJ e a das aplicações instanciadas desse framework.

A geração automatizada torna o desenvolvimento de sistemas baseados no framework GRENJ mais eficiente e ameniza as dificuldades de configuração dos pontos variáveis desse framework. A análise do sistema que está sendo desenvolvido se resume em o desenvolvedor seguir os padrões da GRN e informar os seus dados específicos em uma interface gráfica baseada em formulários. O projeto e construção são realizados pelo gerador e pelo framework. Os testes foram realizados durante o desenvolvimento do framework e do gerador e da configuração do domínio. As principais vantagens desse processo são:

- A redução do tempo de desenvolvimento.
- A redução do custo de desenvolvimento, devido à redução do tempo.
- O desenvolvimento não depende de inúmeras ferramentas para escrita dos requisitos, modelagem do sistema, implementação do código, entre outras coisas.
- Redução dos defeitos, uma vez que o código do framework GRENJ e a geração foram testados. Além disso, quanto mais o framework e o gerador são utilizados, mais a qualidade de seu código e dos gabaritos XSL é atestada.

A principal desvantagem do uso de geradores de aplicações no desenvolvimento dos sistemas baseados no framework GRENJ é a redução da flexibilidade. Algumas opções precisaram ser predefinidas e fixadas para tornar possível a geração automatizada dos artefatos. Por exemplo, os rótulos dos campos relacionados com os atributos das classes do framework não podem ser customizados na geração automática. Essa customização pode ser realizada com a instanciação do framework GRENJ sem o apoio do wizard ou com a modificação, via IDE, dos artefatos gerados pelo wizard. Os artefatos de um sistema gerado com o apoio do wizard do framework GRENJ são organizados no formato de um projeto da IDE NetBeans (Sun Microsystems, 2009g) para facilitar a realização de modificações no código desse sistema.

6 Estudos de Caso com a Geração de Sistemas Baseados no Framework GRENJ

6.1. Considerações Iniciais

A construção da camada de interface gráfica do framework GRENJ favorece a instanciação completa de sistemas web pertencentes ao domínio de gestão de recursos de negócios. Além disso, com a criação de um wizard por meio da utilização de um gerador de aplicações configurável, é possível a geração automatizada de sistemas baseados nesse framework.

Três estudos de caso foram elaborados para avaliar e, possivelmente, corrigir o funcionamento da geração automática de sistemas baseados no framework GRENJ com o apoio da ferramenta GRENJ-Gens em conjunto com o gerador de aplicações Captor. O primeiro contempla o sistema para uma locadora de DVDs, que foi descrito nas Seções 3.3.1 e 4.3.3. Além de verificar o funcionamento do wizard, esse estudo de caso objetiva comparar a geração automática do sistema com a sua instanciação com base no framework GRENJ sem o apoio de um gerador de aplicações. O segundo estudo de caso aborda o desenvolvimento de um sistema para registro das matrículas dos alunos de uma instituição de ensino privada, enquanto que o terceiro trata de um sistema para uma oficina mecânica.

Antes da realização dos três estudos de caso, as informações sobre os softwares auxiliares (Java, Tomcat e MySQL) foram registradas na ferramenta GRENJ, para que esses dados fossem atualizados nos gabaritos e nos arquivos de configuração do wizard do framework GRENJ.

Além desta Seção, este Capítulo está organizado da seguinte maneira: na Seção 6.2 é abordado o desenvolvimento do primeiro estudo de caso com a geração do sistema para uma locadora de DVDs; na Seção 6.3 é tratado o estudo de caso para o sistema de matrícula de alunos em uma instituição de ensino privada; na Seção 6.4 é descrita a realização do terceiro estudo de caso com a geração do sistema para uma oficina mecânica; e, finalmente, na Seção 6.5 estão as considerações finais deste Capítulo.

6.2. Estudo de Caso 1: Geração de um Sistema para uma Locadora de DVDs

Uma versão do sistema para uma locadora de DVDs foi desenvolvida com o apoio do wizard criado no gerador de aplicações Captor. A Tabela 6.1 contém uma relação dos requisitos do sistema para uma locadora de DVDs.

Tabela 6.1. Relação dos requisitos do sistema para uma locadora de DVDs.

#	Descrição
1	A locadora realiza o aluguel de DVDs de filmes que podem ter uma ou mais cópias.
2	Cada filme possui um código, título e ano.
3	Cada DVD pertence a um filme e possui código, localização, status que indica se está disponível ou não.
4	Os filmes são classificados por categoria que indica o valor diário da locação.
5	Os filmes também são classificados por gênero (comédia, terror, ação, etc.).
6	Os DVDs são alugados para os clientes cadastrados da locadora. As informações que o sistema deve manter sobre o cliente são: código, nome, telefone e CPF.
7	As informações de locação são: código, data de locação, data de devolução prevista, código do cliente, DVDs alugados, data de devolução efetiva e valor. Um cliente pode alugar mais de um DVD em uma mesma locação.

No desenvolvimento do sistema para uma locadora de DVDs foram aplicados quatro padrões GRN: Identificar o Recurso, Quantificar o Recurso, Alugar o Recurso e Itemizar a Transação do Recurso. A Figura 6.1 apresenta o modelo de classes do sistema resultante da aplicação desses padrões, em que as classes do framework GRENJ são estendidas pelas classes específicas do sistema e estão destacadas com fundo cinza. Nas classes específicas do sistema, sem cor de fundo na Figura 6.1, assim como nos formulários do wizard, são acrescentados somente os atributos que não são fornecidos pelas classes do framework.

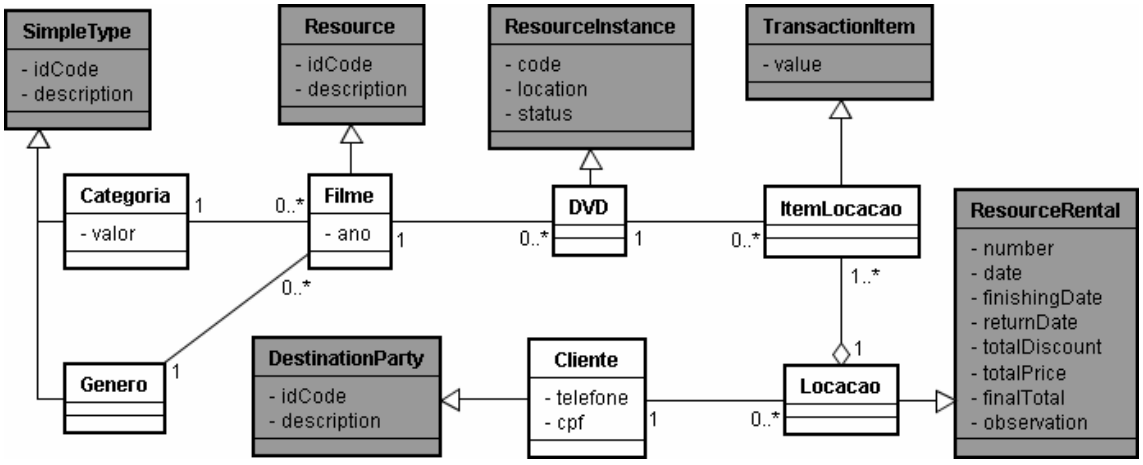

Figura 6.1. Modelo de classes do sistema para uma locadora de DVDs utilizando GRN e GRENJ.

O sistema para uma locadora de DVDs foi gerado com o preenchimento do formulário que identifica o sistema e dos formulários relativos às classes existentes no modelo obtido da aplicação dos padrões da GRN, Figura 6.1. A Figura 6.2(a) apresenta o formulário de descrição do sistema (*System Basic Description*), que é o primeiro formulário apresentado pelo wizard ao desenvolvedor. As informações desse formulário são utilizadas para gerar o título no cabeçalho da tela do sistema e sua página inicial, Figura 6.2(b).

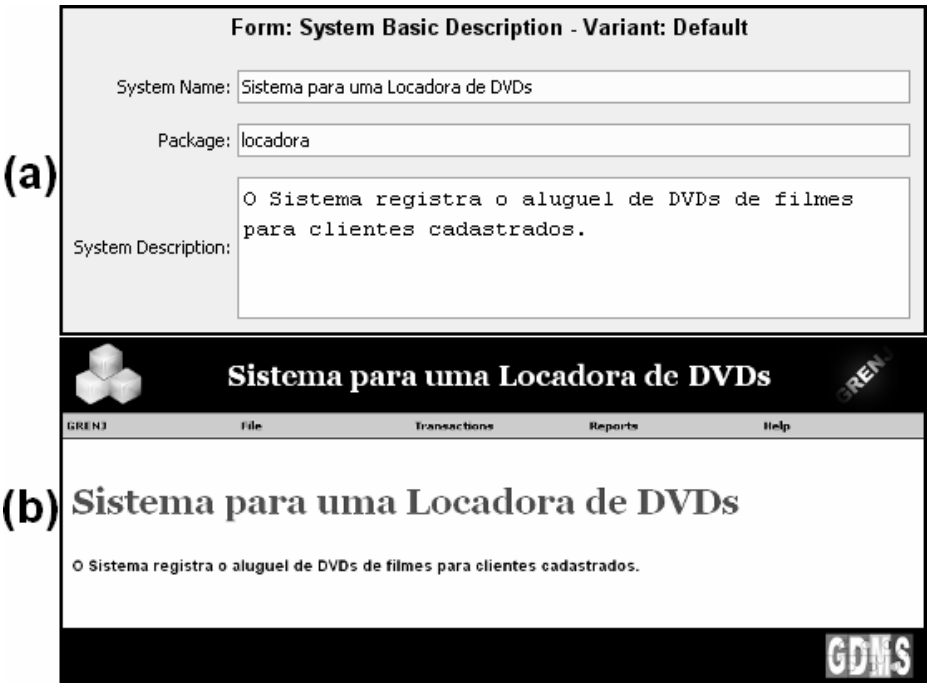

Figura 6.2. Criação da página inicial do Sistema para uma Locadora de DVDs.

O segundo formulário preenchido no wizard do framework GRENJ foi o de identificação do recurso, denominado *Identify Resource*. Os dados preenchidos nesse formulário são utilizados para gerar a classe *Filme* na camada de negócios do sistema para uma locadora de DVDs. A classe *Filme* estende a classe *Resource* do framework GRENJ e, além dos atributos herdados dessa classe, possui um atributo do tipo *int* (inteiro) chamado *ano*. Na camada de interface gráfica do sistema é gerada a classe *FilmeFormServlet*, que é responsável por carregar o formulário de Cadastro de Filmes na tela do sistema quando solicitado pelo usuário.

A Figura 6.3(a) apresenta o formulário *Identify Resource* do wizard do framework GRENJ com os dados específicos da classe *Filme*, enquanto que a Figura 6.3(b) apresenta o formulário de Cadastro de Filmes do sistema para uma locadora de DVDs.

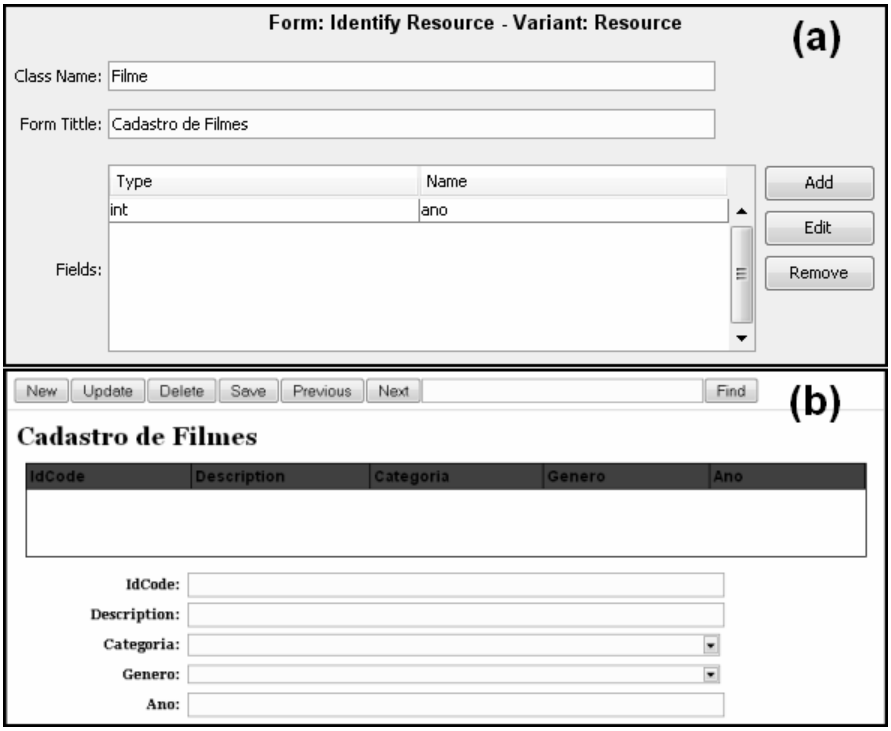

Figura 6.3. Criação do formulário de Cadastro de Filmes.

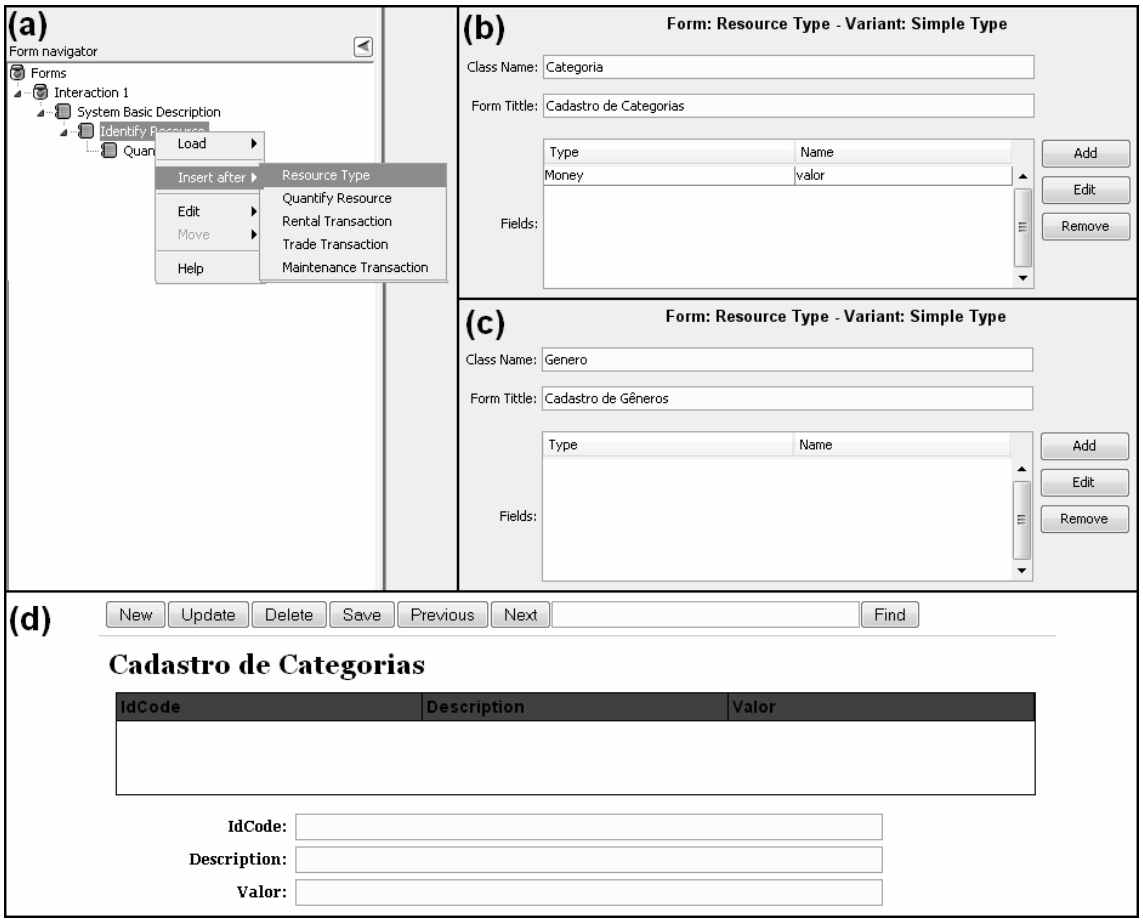

Figura 6.4. a) Carregamento dos formulários *Resource Type* do wizard; b) Preenchimento dos dados da classe *Categoria*; c) Preenchimento dos dados da classe *Genero*; e d) o formulário de Cadastro de Categorias da interface gráfica do sistema para uma locadora de DVDs.

A Figura 6.4(a) ilustra como um formulário de *Resource Type* é selecionado no wizard do framework GRENJ. As Figuras 6.4(b) e 6.4(c) apresentam duas instâncias desse formulário com os dados das classes *Categoria* e *Genero*, respectivamente. A Figura 6.4(d) ilustra o formulário de Cadastro de Categorias do sistema para uma locadora de DVDs, construído pela classe *CategoriaFormServlet*, que foi gerada a partir dos dados sobre a classe *Categoria* informados no formulário de Tipo de Recurso. O formulário de Cadastro de Gêneros, relacionado com a classe *Genero*, é semelhante ao formulário de Cadastro de Categorias.

Como a aplicação do segundo padrão da GRN, Quantificar o Recurso, é obrigatória, o wizard do framework GRENJ inicia o projeto de um sistema com o formulário *Quantify Resource* carregado. Entretanto, o desenvolvedor necessita indicar qual variante desse padrão deve ser aplicada. No caso do sistema para uma locadora de DVDs, foi selecionada a variante de recurso instanciável (*Instantiable Resource*) para a definição da classe *DVD*, que representa as instâncias do recurso *Filme*. A Figura 6.5(a) apresenta a variante *Instantiable Resource* do formulário *Quantify Resource* com os dados da classe *DVD*. Além da classe *DVD*, gerada na camada de negócios do sistema, as classes *DVDFormServlet* e *DVDTableServlet* foram geradas na camada de interface gráfica do sistema e são responsáveis por construir o formulário de Cadastro de DVDs do sistema, apresentado pela Figura 6.5(b).

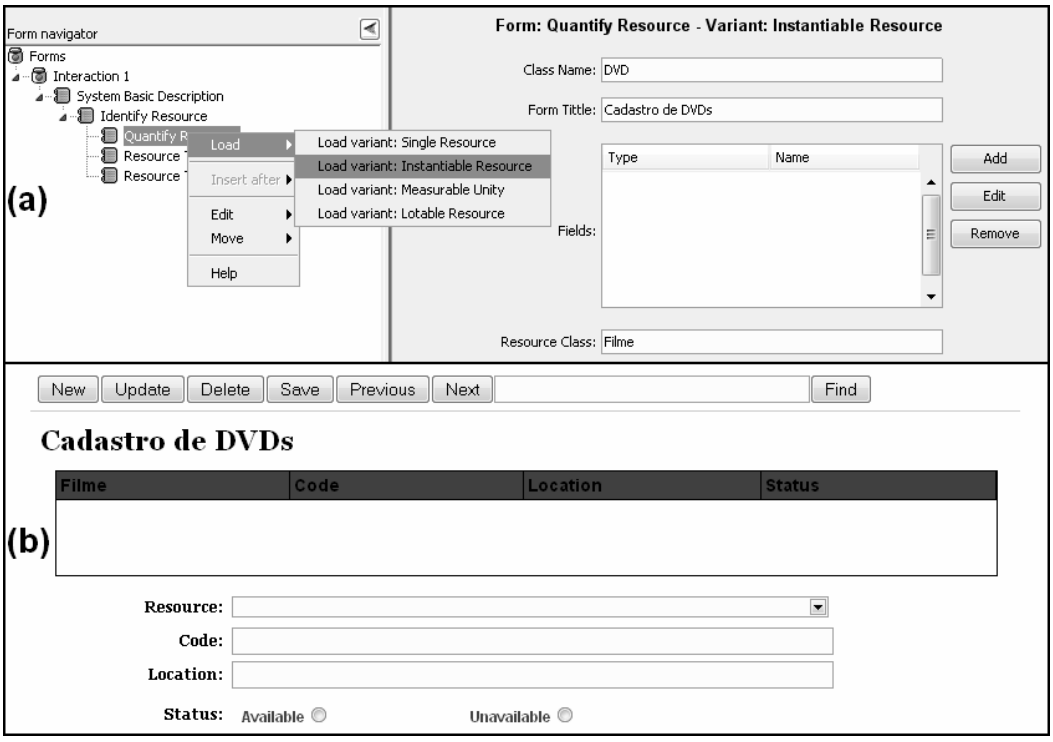

Figura 6.5. Criação do formulário de Cadastro de DVDs.

O passo seguinte foi o preenchimento do formulário que define as transações de aluguel (*Rental Transaction*) e representa a aplicação do quarto padrão da GRN, Alugar o Recurso. A Figura 6.6 apresenta formulário de transações de aluguel que definiu a classe *Locacao* no sistema para uma locadora de DVDs.

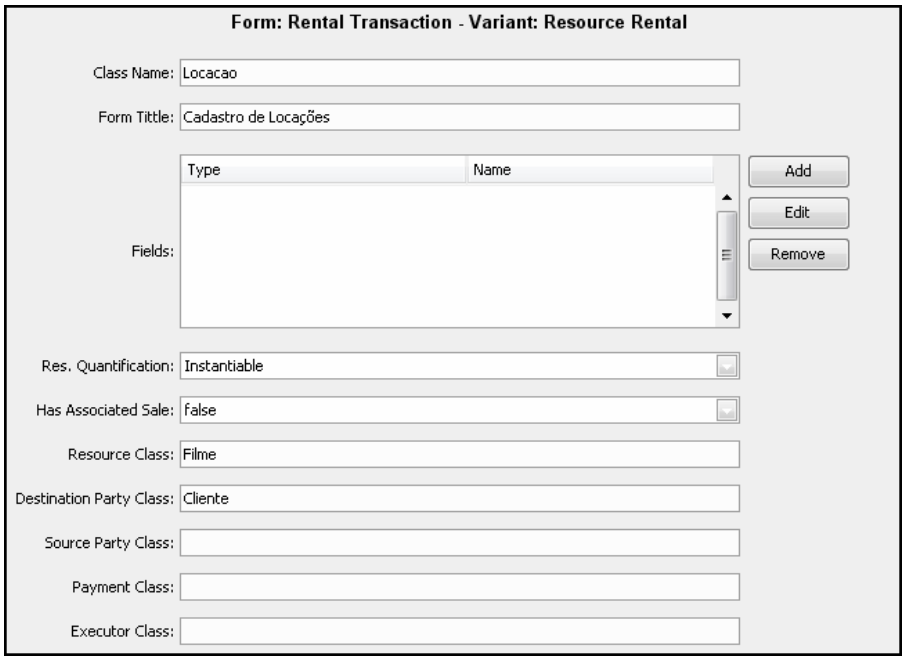

Figura 6.6. Formulário de transações de aluguel com os dados da classe *Locacao*.

Os clientes da locadora de DVDs representam o destino dos recursos envolvidos nas transações de aluguel. A Figura 6.7(a) ilustra o formulário de destino (*Destination Party*) com os dados da classe *Cliente*, enquanto que a Figura 6.7(b) apresenta o formulário de Cadastro de Clientes da interface gráfica do sistema.

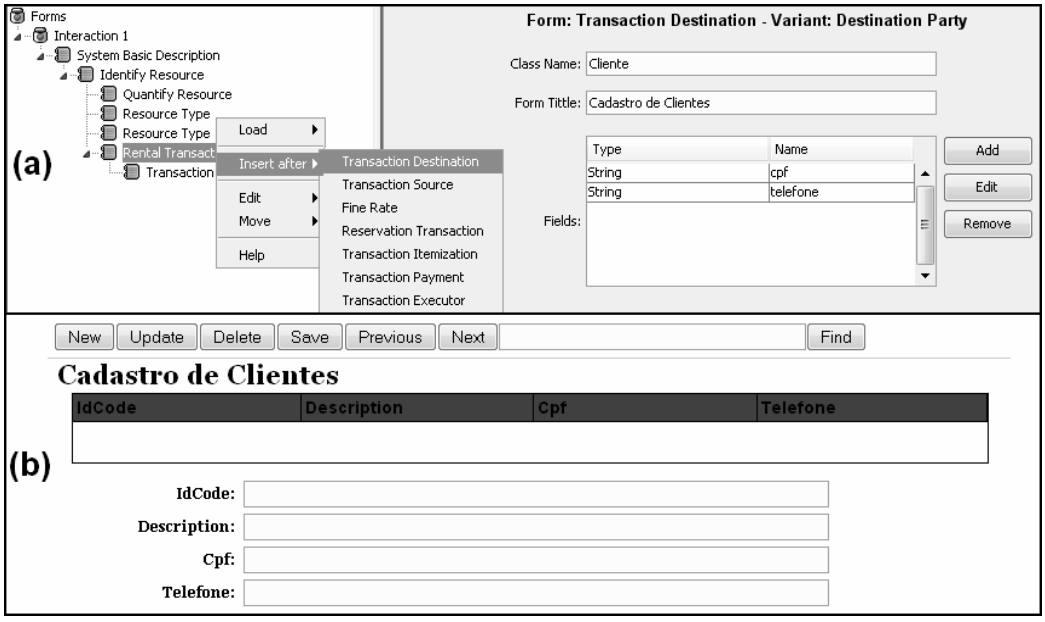

Figura 6.7. Criação do formulário de Cadastro de Clientes.

O formulário *Transaction Itemization* está relacionado com o padrão Itemizar a Transação do Recurso e deu origem a classe *ItemLocacao* na camada de negócios do sistema. A Figura 6.8(a) ilustra a hierarquia dos formulários preenchidos no wizard para o sistema para uma locadora de DVDs, enquanto que a Figura 6.8(b) apresenta o formulário *Transaction Itemization* com os dados da classe *ItemLocacao*.

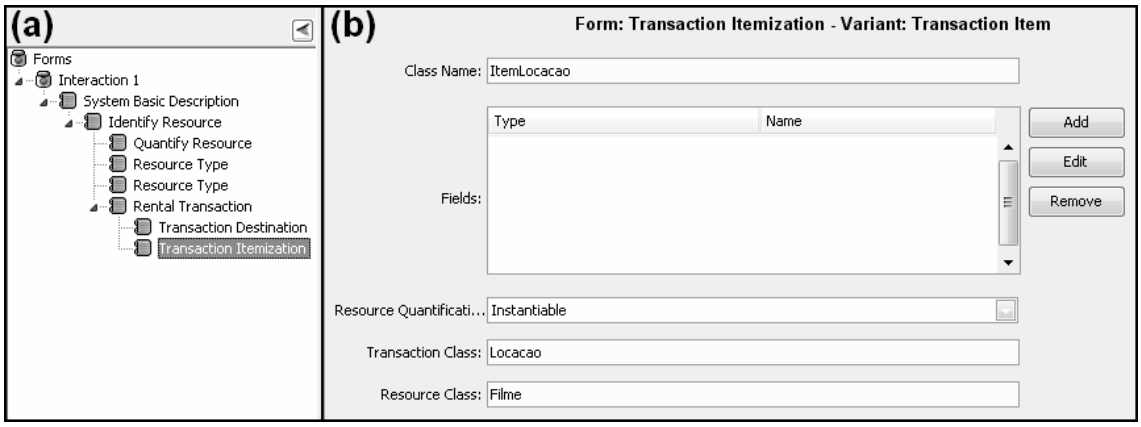

Figura 6.8. a) Hierarquia dos formulários utilizados para a geração do sistema para uma locadora de DVDs e b) o formulário *Transaction Itemization* preenchido com os dados da classe *ItemLocacao*.

Os dados sobre a classe *Locacao* preenchidos no formulário *Resource Rental* do wizard também são utilizados para gerar as classes *LocacaoFormServlet* e *LocacaoTableServlet* na camada de interface gráfica do sistema. Da mesma forma, os dados sobre a classe *ItemLocacao* preenchidos no formulário *Transaction Itemization* foram utilizados para criar a classe *ItemLocacaoPanelServlet*. Essas classes dão origem ao formulário de Cadastro de Locações, apresentado pela Figura 6.9.

Figura 6.9. Formulário de Cadastro de Locações.

Após o preenchimento dos formulários, iniciou-se a geração dos artefatos do sistema para uma locadora de DVDs. Falhas ocorreram na compilação das classes geradas, ação que é determinada no arquivo *pos-build* do domínio do framework GRENJ. Constatou-se que algumas classes possuíam defeitos em seus códigos causados por erros de digitação nos gabaritos XSL ou por leitura incorreta de um valor dos formulários. Defeitos também foram detectados no código do script da base de dados. Com a correção desses defeitos, o sistema para uma locadora de DVDs foi finalmente gerado.

Por determinação do arquivo *pos-build,* a pasta do projeto do sistema para uma locadora de DVDs foi copiada para o diretório das aplicações web configurado no Apache Tomcat. A execução do sistema exige a ativação do servidor Apache Tomcat. No navegador, o endereço do sistema, que possui o formato *localhost/nome_do_pacote_do_sistema,* deve ser informado. Para esse caso, o endereço é *localhost/locadora,* pois o nome do pacote informado no formulário de descrição do sistema é *locadora*.

A maior parte do esforço gasto no desenvolvimento de uma versão desse sistema com base nos padrões da GRN, porém, sem o apoio do framework GRENJ ou de um gerador de aplicações (Durelli *et al.*, 2008) correspondeu ao desenvolvimento da interface gráfica do sistema e de sua interligação com a camada de negócios. O desenvolvimento de uma versão do mesmo sistema com base na GRN e no framework GRENJ sem o apoio de um gerador de aplicações foi realizado em um período consideravelmente inferior. Entretanto, deve-se levar em consideração que o desenvolvedor possuía conhecimento total sobre a arquitetura do framework e, portanto, sabia exatamente quais classes deveriam ser estendidas e quais métodos deveriam ser sobrecarregados. Finalmente, houve uma redução ainda maior no esforço gasto no desenvolvimento de uma versão desse sistema com base na GRN e no framework GRENJ com o apoio do wizard.

Apesar de ter um tempo consideravelmente superior às demais versões, a versão do sistema para uma locadora de DVDs desenvolvida somente com o apoio da GRN possui maior flexibilidade. A implementação de requisitos específicos como, por exemplo, a criação de um método que define a data de devolução dos DVDs de uma locação, nessa versão foi facilitada, pois o sistema foi completamente desenvolvido com noção prévia da existência desses requisitos. O framework GRENJ não prevê requisitos

específicos de uma única aplicação e, consequentemente, a inclusão desses métodos na versão baseada nesse framework exige maior esforço do desenvolvedor. Na versão desenvolvida com o apoio do gerador de aplicações, a inclusão desse tipo de requisito representa um processo de manutenção adaptativa, uma vez que o sistema já se encontra desenvolvido.

6.3. Estudo de Caso 2: Geração de um Sistema para Registro de Matrícula de Alunos

O segundo estudo de caso aborda o desenvolvimento de um sistema para registro das matrículas dos alunos de uma instituição de ensino privada. A Tabela 6.2 contém os requisitos desse sistema.

Tabela 6.2. Relação dos requisitos do sistema para registro de matrícula de alunos.

#	Descrição
1	O sistema deve permitir a inclusão, alteração e remoção de departamentos com os seguintes atributos: código, descrição, sigla.
2	O sistema deve permitir a inclusão, alteração e remoção de disciplinas com os seguintes atributos: código, descrição, carga horária, nome do professor que a ministra e o departamento a qual a disciplina pertence.
3	O sistema deve permitir a inclusão, alteração e remoção de alunos. Cada aluno possui os seguintes atributos: código, nome, endereço, email, telefone e RG.
4	O sistema deve permitir a inclusão, alteração e remoção de cursos com os seguintes atributos: código, nome, número de semestres, nome do coordenador.
5	O sistema deve permitir a inclusão, alteração e remoção de matrícula. A matrícula é feita semestralmente e possui os seguintes atributos: número, semestre (por exemplo, 2009.1), aluno, curso e valor total. Associada à matrícula deve existir uma relação das disciplinas que o aluno cursará, o valor e o turno. O valor total da matrícula é igual à soma do valor de cada disciplina.
6	O sistema deve permitir as seguintes opções de pagamento para a matrícula: 1) dinheiro; 2) cheque.

Nesse sistema, a matricula é entendida como uma transação de comercialização em que as disciplinas representam recursos os quais são vendidos aos alunos. Um departamento se relaciona com uma disciplina de maneira semelhante à que um tipo de recurso se relaciona com um recurso, como definido no primeiro padrão da GRN, enquanto que um curso pode ser entendido como a origem das disciplinas de uma matricula. Com base nessas informações, a Tabela 6.3 contém a relação de padrões da GRN aplicados no desenvolvimento do sistema para registro de matrícula de alunos, das classes identificadas na camada de negócios do sistema e dos respectivos formulários do wizard utilizados para gerá-las.

Tabela 6.3. Padrões da GRN aplicados na geração do sistema para registro de matrícula de alunos.

Padrão da GRN	Classe Gerada na Camada de Negócios	Formulário do Wizard
Identificar o Recurso	Disciplina	Identify Resource
	Departamento	Simple Type
Quantificar o Recurso	-	Single Resource
Comercializar o Recurso	Matricula	Trade Transaction
	Curso	Source Party
	Aluno	Destination Party
Itemizar a Transação do Recurso	ItemMatricula	Transaction Itemization
Pagar pela Transação do Recurso	FormaPagamento	Transaction Payment

A Figura 6.10 apresenta o modelo de classes da camada de negócios do sistema para registro de matrícula de alunos criado a partir da aplicação dos padrões da GRN. As classes destacadas com fundo na cor cinza são provenientes da camada de negócios do framework GRENJ.

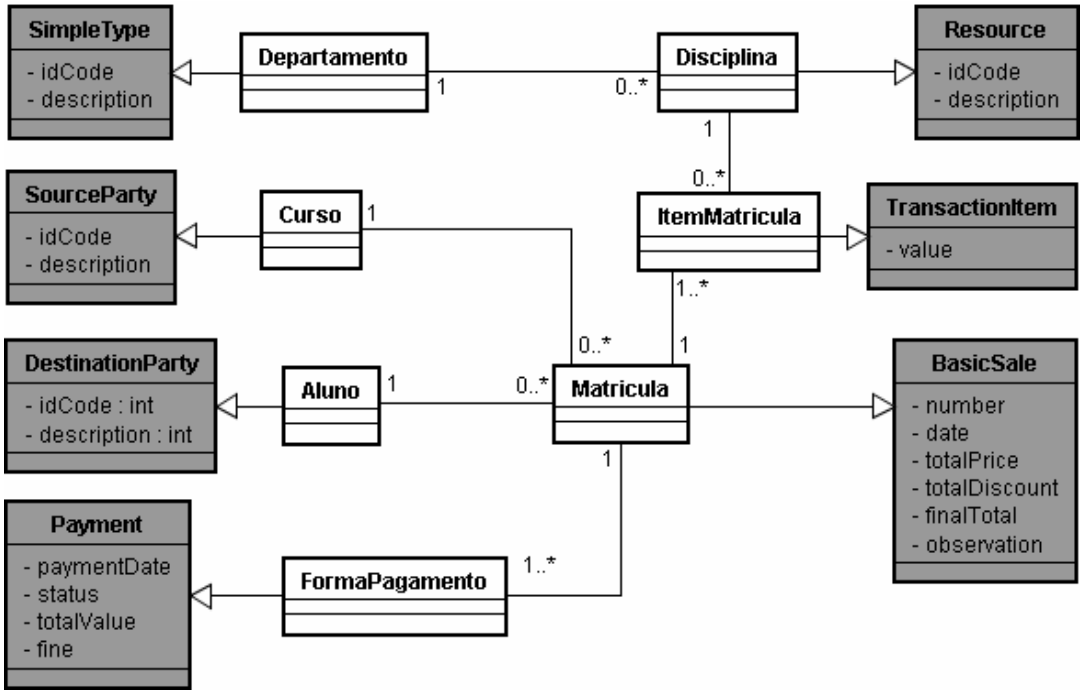

Figura 6.10. Modelo de classes do sistema para registro de matricula dos alunos.

Os formulários do wizard do framework GRENJ referentes às transações de aluguel, de reserva, de comercialização, de entrega e de manutenção são semelhantes por possuírem, praticamente, os mesmos campos. A Figura 6.11 apresenta a variante *Basic Sale* do formulário do wizard para as transações de comercialização, denominado *Trade Transaction*, cujos campos foram preenchidos com os dados da classe que representa a matrícula dos alunos.

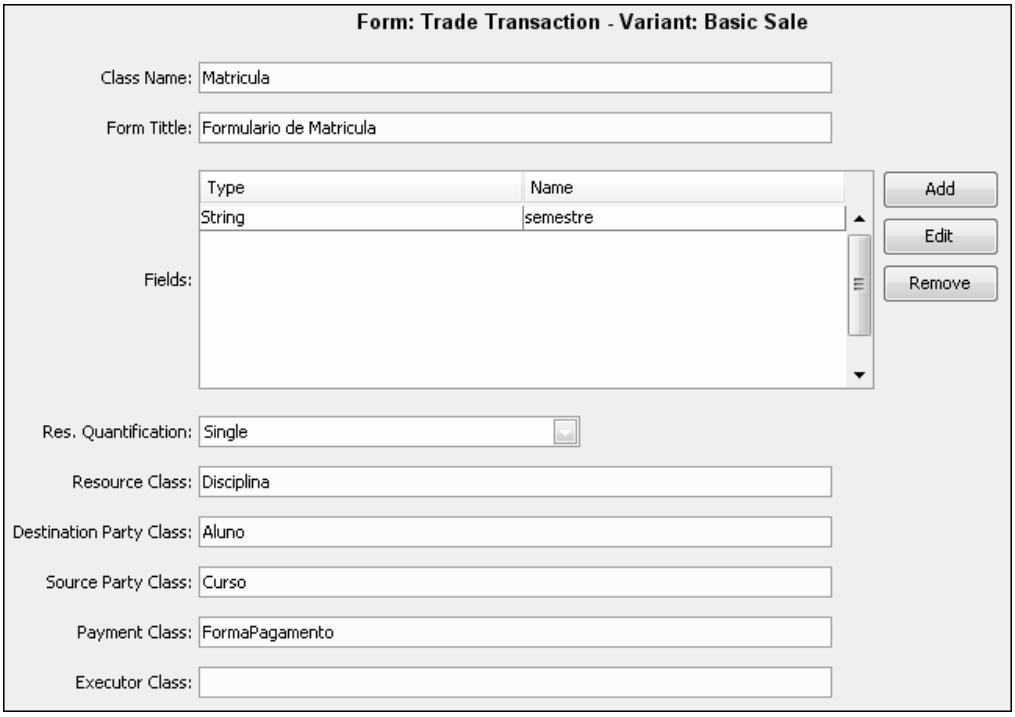

Figura 6.11. Formulário *Trade Transaction* com os dados da classe *Matricula*.

O formulário *Transaction Payment* define a classe *FormaPagamento* para controle da forma de pagamento de cada matrícula. Esse formulário define também que, inicialmente, toda matrícula é paga em dinheiro (*Cash*), porém o sistema permite ao aluno que efetue o pagamento via cheque (*Check*). A Figura 6.12 apresenta o formulário que define a classe *FormaPagamento* na camada de negócios do sistema.

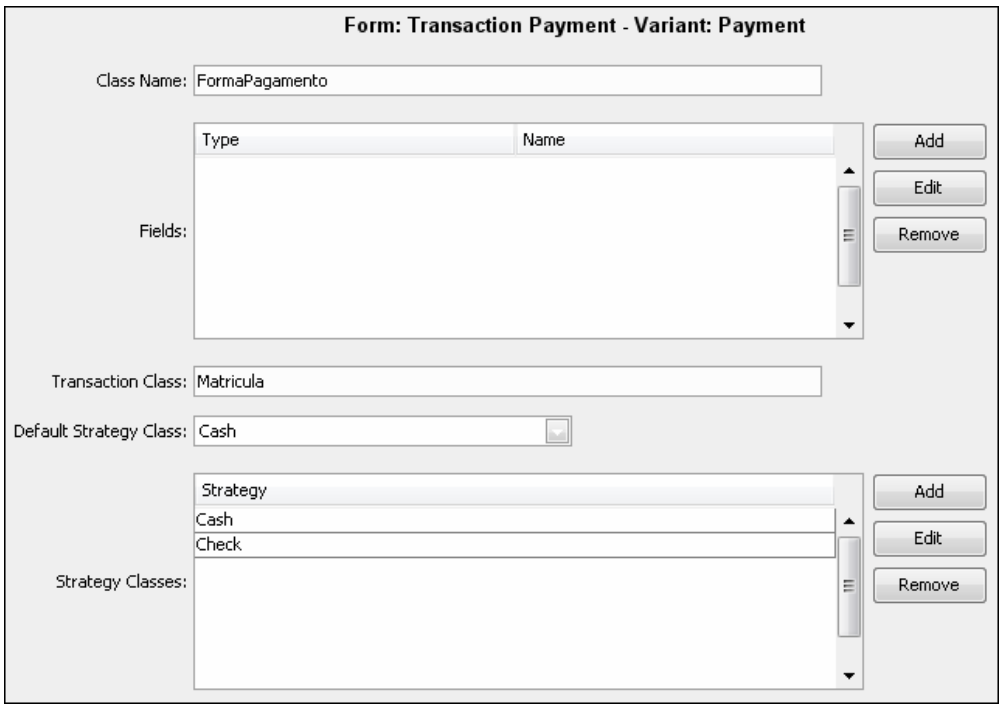

Figura 6.12. Formulário que define os dados da classe *FormaPagamento*.

Na primeira vez em que o sistema para registro de matrículas de alunos foi gerado ocorreram falhas na compilação do seu código, devido a defeitos existentes nos gabaritos XSL. O processo de correção foi o mesmo aplicado no estudo de caso do sistema para uma locadora de DVDs. Porém, após a geração do sistema, a sua execução apresentou falhas, uma vez que foram aplicados padrões que não haviam sido utilizados anteriormente com o framework GRENJ completo. A maior parte dessas falhas ocorreu no painel de pagamento do sistema, montado pela classe *FormaPagamentoPanelServlet*, que estende *PaymentPanelServlet* da camada de interface gráfica do framework GRENJ.

A Figura 6.13 apresenta o Formulário de Matrícula da interface gráfica do sistema para registro de matrículas de alunos dividido em duas partes: a) a tabela que mostra todos os registros de matrículas de alunos e os campos do formulário; e b) o painel para a inclusão das disciplinas a serem cursadas e o painel de pagamento da matrícula. O Formulário de Matrícula da Figura 6.13 foi gerado com base nas informações sobre a classe *Matricula* preenchidas no formulário *Resource Trade* do wizard, Figura 6.11.

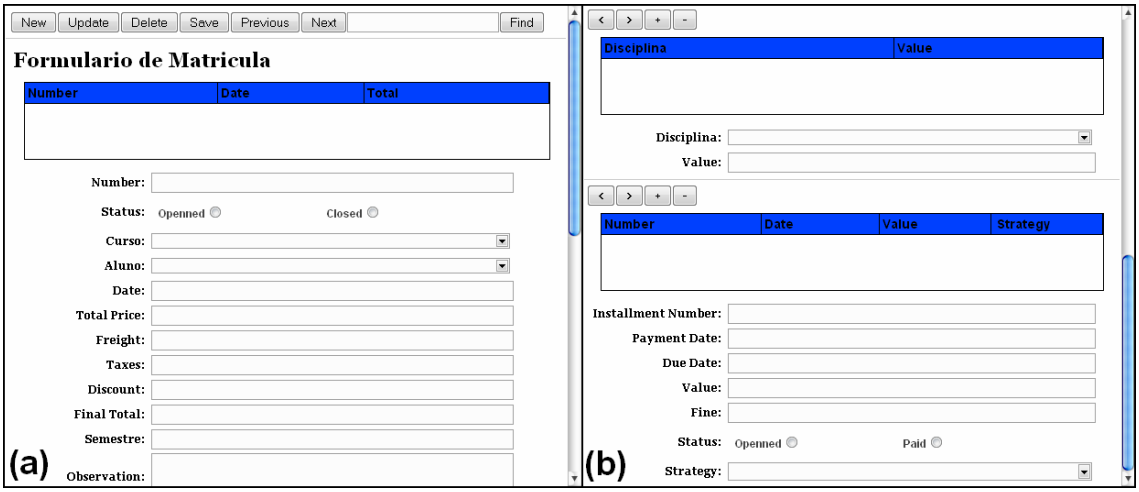

Figura 6.13. Formulário de Matrícula do sistema para registro de matrículas de alunos.

O processo de geração do sistema para registro de matrículas de alunos foi realizado com a criação de um modelo de classes com base nos requisitos e com a aplicação dos padrões da GRN e o preenchimento de oito formulários do wizard do framework GRENJ. O tempo necessário para o desenvolvimento do sistema com o uso do wizard é inferior quando comparado ao desenvolvimento sem o apoio do wizard.

6.4. Estudo de Caso 3: Geração de um Sistema para uma Oficina Mecânica

O terceiro estudo de caso refere-se a um sistema para uma oficina mecânica e foi elaborado para contemplar os padrões da GRN que são associados a uma transação de manutenção. A Tabela 6.4 contém a relação dos requisitos desse sistema.

Tabela 6.4. Relação dos requisitos do sistema para uma oficina mecânica.

#	Descrição
1	O sistema deve permitir a inclusão, alteração e remoção de clientes da oficina, com os seguintes atributos: código, nome, endereço, telefone e RG.
2	O sistema deve permitir a inclusão, alteração e remoção de veículos pertencentes a um cliente, com os seguintes atributos: código, modelo, ano, cor e placa.
3	O sistema deve permitir a inclusão, alteração e remoção de tipos de veículo (por exemplo, automóvel, caminhão, motocicleta, etc.).
4	O sistema deve permitir a inclusão, alteração e remoção de mecânicos da oficina, com os seguintes atributos: código, nome, endereço, telefone, RG, data de nascimento, especialidade e salário.
5	O sistema deve permitir a inclusão, alteração e remoção de peças, que são utilizadas na manutenção dos veículos, com os seguintes atributos: código, nome, fabricante e localização na prateleira.
6	O sistema deve permitir a inclusão, alteração e remoção de ordens de serviço que possuem os seguintes atributos: data de entrada, data de saída, cliente, veículo, mecânico atendente, valor do serviço, desconto concedido e valor total.
7	Cada ordem de serviço possui uma lista de tarefas a serem realizadas, informando os defeitos encontrados, a solução aplicada, o custo e o mecânico responsável.
8	Cada ordem de serviço também possui uma lista das peças utilizadas na manutenção do veículo, que ainda inclui o custo e a quantidade utilizada dessas peças.

Os recursos tratados pelo sistema para uma oficina mecânica são os veículos de seus clientes, que são recursos únicos. Uma ordem de serviço representa uma transação de manutenção, definida no padrão Manter o Recurso da GRN. Uma ordem de serviço é atribuída ao mecânico que atende o cliente e cada tarefa pode ser realizada por um mecânico diferente. A Tabela 6.5 contém a relação de padrões da GRN aplicados para esse sistema, os formulários utilizados e as classes geradas na camada de negócios.

Tabela 6.5. Padrões da GRN aplicados na geração do sistema para uma oficina mecânica.

Padrão da GRN	Classe Gerada na Camada de Negócios	Formulário do Wizard
Identificar o Recurso	Veiculo	Identify Resource
Identificar o Recurso	TipoVeiculo	Simple Type
Quantificar o Recurso	-	Single Resource
Manter o Recurso	OrdemServico	Maintenance Transaction
Manter o Recurso	Cliente	Destination Party
Identificar o Executor da Transação	Mecanico	Transaction Executor
Identificar as Tarefas da Manutenção	Tarefa	Maintenance Task
Identificar as Peças da Manutenção	Peca	Part
Identificar as Peças da Manutenção	PecaManutencao	Maintenance Part

A Figura 6.14 apresenta o modelo de classes do sistema para uma oficina mecânica criado com a aplicação dos padrões da GRN. As classes do framework GRENJ estendidas pelas classes específicas do sistema estão destacadas com fundo cinza e as classes da aplicação estão sem cor de fundo.

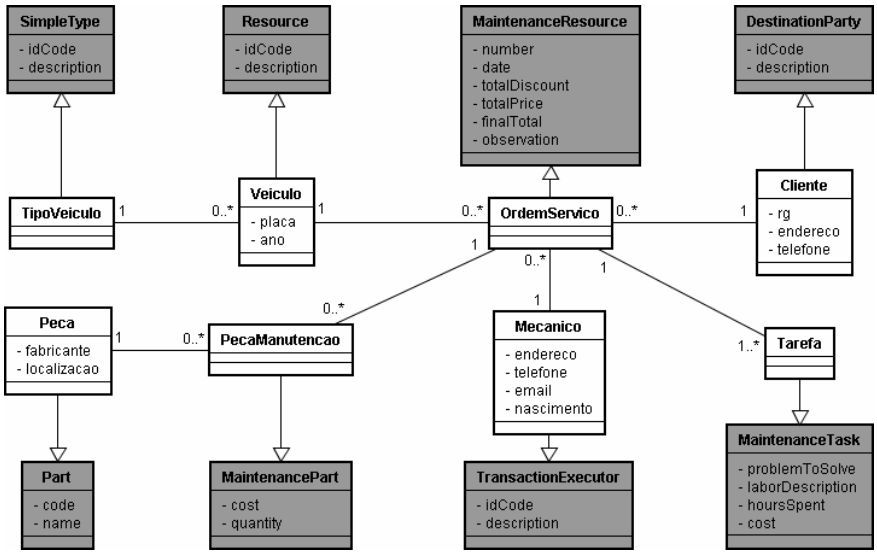

Figura 6.14. Modelo de classes do sistema para uma oficina mecânica usando GRN e GRENJ.

O preenchimento dos formulários do wizard foi realizado com base nos requisitos apresentados na Tabela 6.4. O formulário que define as classes responsáveis pelas transações de manutenção dos recursos é denominado *Maintenance Transaction*. Os dados preenchidos nesse formulário deram origem à classe *OrdemServico* na camada de negócios e às classes *OrdemServicoFormServlet* e *OrdemServicoTableServlet* na camada de interface gráfica do sistema. A Figura 6.15 ilustra o formulário *Maintenance Transaction* do wizard com os dados da classe *OrdemServico*.

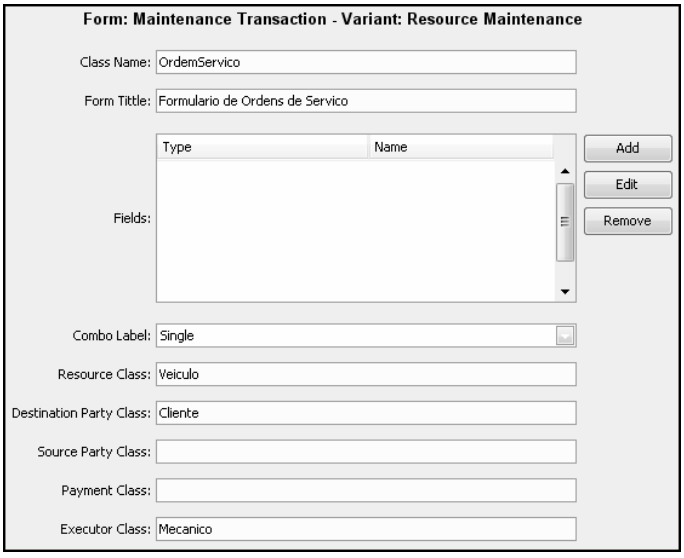

Figura 6.15. Formulário com os dados da classe *OrdemServico*.

O formulário *Transaction Executor* contém as informações necessárias para a geração da classe *Mecanico* na camada de negócios e das classes *MecanicoFormServlet* e *MecanicoTableServlet* na camada de interface gráfica do sistema para uma oficina mecânica. No wizard, esse formulário foi carregado a partir do formulário *Maintenance Transaction*, que definiu a classe *OrdemServico* na camada de negócios. A Figura 6.16(a) apresenta a tela do wizard que contém a hierarquia dos formulários carregados para a geração do sistema para uma oficina mecânica e o formulário *Transaction Executor* com os dados da classe *Mecânico*, enquanto que a Figura 6.16(b) apresenta o formulário de Cadastro de Mecânicos do sistema após ter sido gerado.

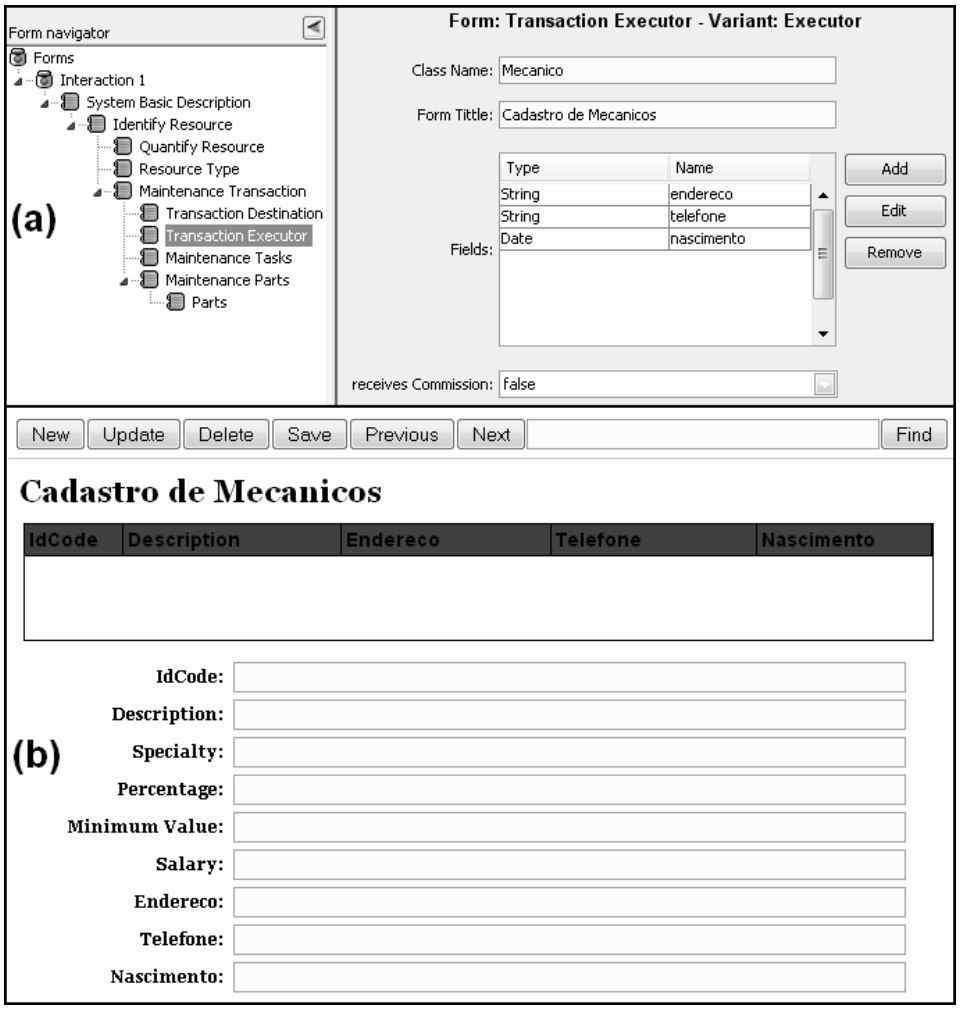

Figura 6.16. Criação do formulário de Cadastro de Mecânicos.

O formulário *Maintenance Tasks* foi utilizado em correspondência com as informações indicadas nos requisitos do sistema sobre as tarefas das ordens de serviço e dá origem às classes *Tarefa*, na camada de negócios, e *TarefaPanelServlet*, na camada de interface gráfica do sistema. A classe *Mecanico* também é utilizada para indicar o executor de cada tarefa. Nenhum atributo adicional foi definido na tabela do formulário

porque a classe *MaintenanceTask* do framework GRENJ já fornece os atributos necessários ao sistema. A Figura 6.17 apresenta o formulário *Maintenance Tasks*.

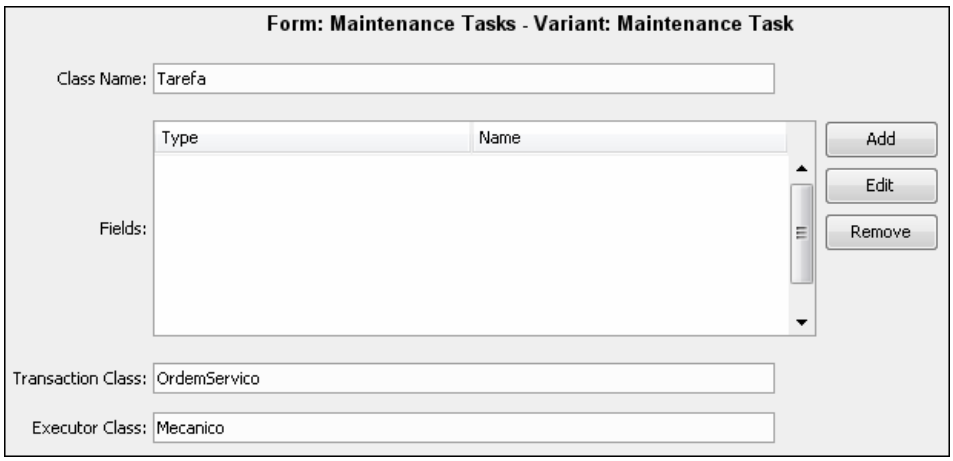

Figura 6.17. Formulário com os dados da classe que identifica as tarefas das ordens de serviço.

A Figura 6.18 ilustra o formulário *Maintenance Parts* que define as classes *PecaManutencao* e *PecaManutencaoPanelServlet*, responsáveis pela identificação das peças utilizadas nas ordens de serviço.

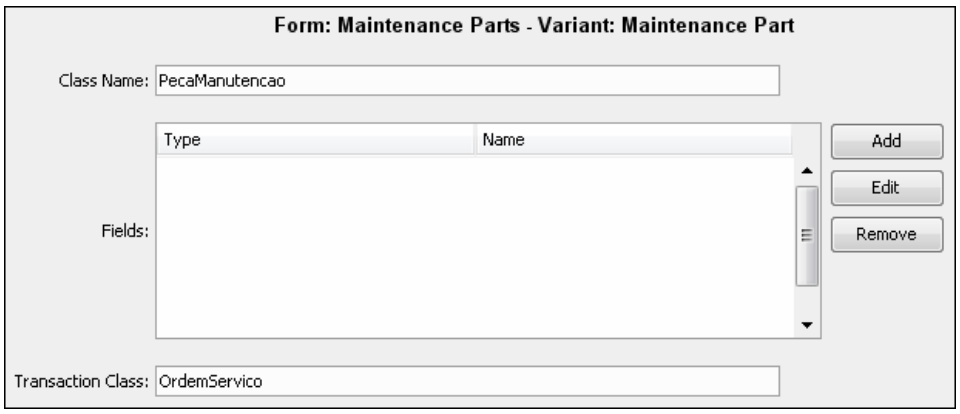

Figura 6.18. Formulário que define as classes que identificam as peças das ordens de serviço.

O Formulário de Ordens de Serviços da interface gráfica do sistema é composto pelas seguintes classes: *OrdemServicoFormServlet* e *OrdemServicoTableServlet*, geradas com os dados do formulário *Maintenance Transaction* do wizard, apresentado na Figura 6.15; *TarefaPanelServlet*, gerada pelo formulário *Maintenance Tasks* do wizard, apresentado na Figura 6.17; e *PecaManutencaoPanelServlet*, gerada pelo formulário *Maintenance Parts* do wizard, apresentado na Figura 6.18. A Figura 6.19 apresenta o Formulário de Ordens de Serviços dividido em duas partes: a) a tabela que mostra todas as ordens de serviço registradas e os campos do formulário; e b) os painéis para registro das tarefas realizadas e das peças utilizadas na ordem de serviço.

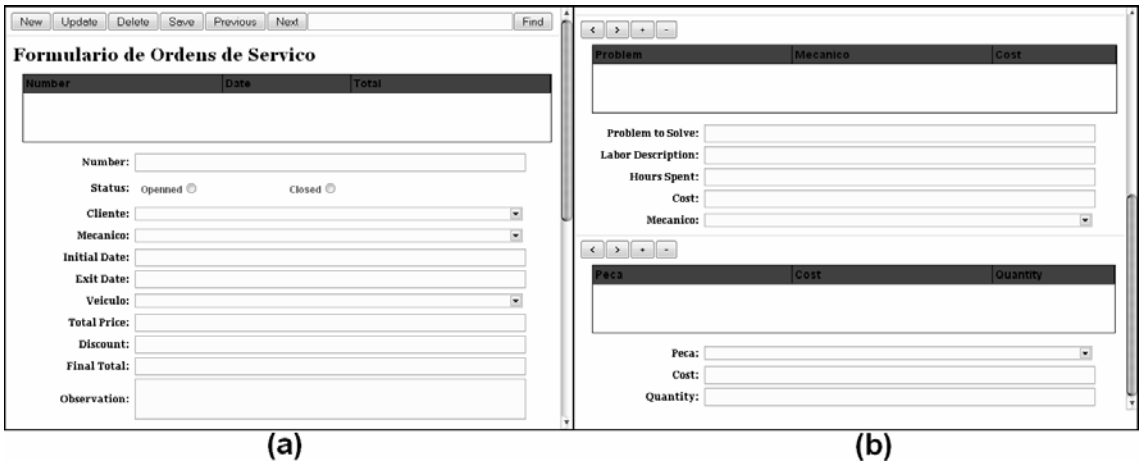

Figura 6.19. Formulário de Ordens de Serviço do sistema para uma oficina mecânica.

O formulário *Parts* foi utilizado para definir as classes responsáveis pelo registro das peças existentes na oficina mecânica. As informações preenchidas nesse formulário deram origem a classe *Peca* na camada de negócios e às classes *PecaFormServlet* e *PecaTableServlet* na camada de interface gráfica do sistema. A Figura 6.20(a) ilustra o formulário *Parts* preenchido com os dados sobre as peças, enquanto que a Figura 6.20 (b) apresenta o formulário de Cadastro de Peças da interface gráfica do sistema.

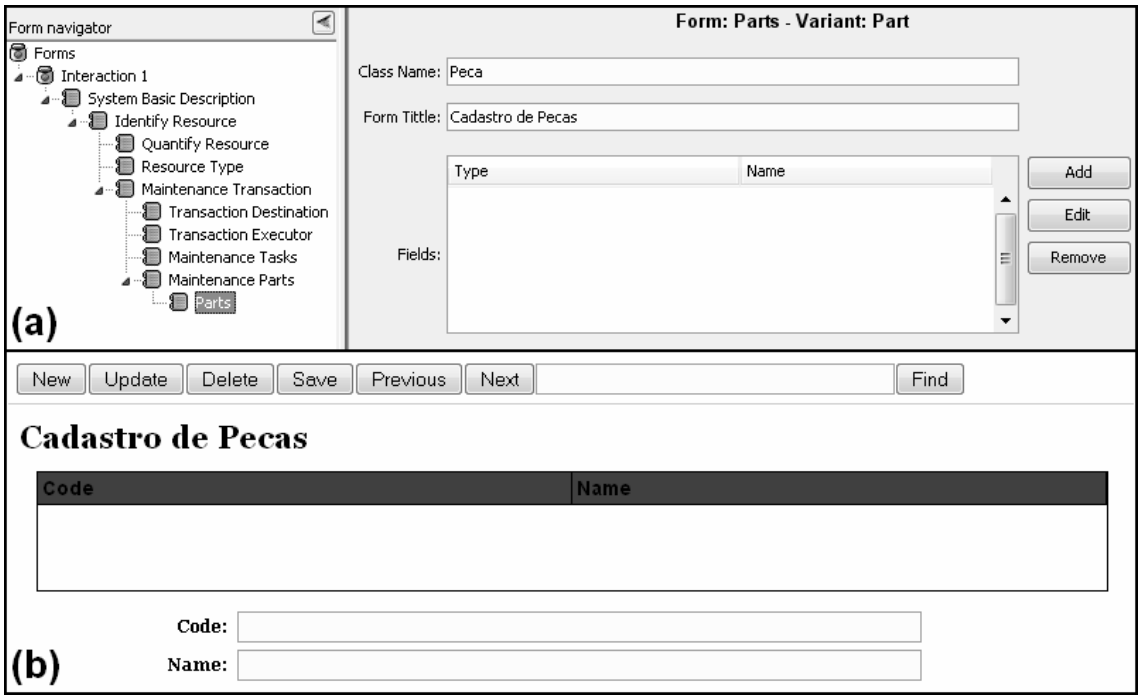

Figura 6.20. Criação do formulário de Cadastro de Peças.

A primeira tentativa de geração do sistema para uma oficina mecânica foi interrompida, devido a erros de compilação das classes geradas. Esses erros foram provocados por defeitos existentes nos gabaritos XSL referentes às classes dos padrões

Identificar o Executor da Transação, Identificar as Tarefas da Manutenção e Identificar as Peças da Manutenção. Após a correção desses defeitos, o sistema foi gerado corretamente. Além disso, a experiência acumulada no desenvolvimento e na correção das classes utilizadas nos sistemas dos estudos de caso anteriores reduziu consideravelmente a ocorrência de falhas durante a execução do sistema para uma oficina mecânica.

6.5. Considerações Finais

Os três estudos de caso apresentados neste Capítulo permitiram analisar as vantagens da utilização do framework GRENJ e de seu wizard gerador de aplicações, bem como verificar e corrigir problemas em seu funcionamento. Porém, outros estudos de caso podem ser futuramente realizados com o intuito de garantir ainda mais a qualidade dos sistemas desenvolvidos com o apoio dessas ferramentas.

Nos três estudos de caso foram encontrados defeitos no código dos gabaritos. Uma das razões para a ocorrência desses defeitos é a sintaxe da linguagem XSL, que provoca amontoamento das instruções e prejudica a legibilidade de seus documentos. Apesar disso, o uso da linguagem XSL é vantajoso por ser uma tecnologia de fácil aprendizagem e altamente portável por ser baseada na linguagem XML (W3C, 2009).

O fato de suas classes do framework GRENJ estarem organizadas em uma hierarquia definida por relacionamentos de herança facilita a identificação e a correção dos trechos com defeito e evidencia as vantagens do conceito de reúso. Isso pôde ser verificado no segundo e no terceiro estudos de caso, em que não foram encontrados defeitos nas classes que realizam as transações, apesar de não terem sido previamente verificadas com a instanciação de um sistema, como ocorreu com a classe *ResourceRental*. O motivo é que todas estendem a mesma classe em cada camada do framework GRENJ. Assim, os defeitos encontrados durante os testes dessas classes e durante a instanciação da versão do sistema para uma locadora de DVDs com o apoio do framework GRENJ foram suficientes para a correção dos problemas que poderiam ocorrer nas classes relacionadas com as transações.

Com o uso do wizard do framework GRENJ, o desenvolvimento de sistemas pertencentes ao domínio de gestão de recursos de negócios é realizado com o preenchimento de formulários. Dessa forma, o processo se torna mais rápido e mais intuitivo, pois o desenvolvedor não precisa ter conhecimento aprofundado sobre

linguagens de programação e técnicas da Engenharia de Software. Também não é necessário conhecer a arquitetura interna do framework GRENJ nem se preocupar com a implementação de inúmeros métodos.

Houve uma constatação de que o esforço para o desenvolvimento dos sistemas com o apoio do wizard do framework GRENJ é inferior ao gasto no desenvolvimento realizado sem esse apoio. Ressalva-se que essa eficiência é obtida com prejuízo à flexibilidade. Entretanto, além de gerar sistemas prontos para serem utilizados, o wizard do framework GRENJ cria uma estrutura de arquivos no formato de um projeto da IDE Netbeans (Sun Microsystems, 2009g). Dessa forma, podem ser modificados para contemplar requisitos que não são previstos pelo framework ou que foram pré-fixados pelos gabaritos XSL do wizard.

7 Conclusões

7.1. Considerações Finais

Este trabalho apresentou a construção da camada de interface gráfica e de um wizard para o framework GRENJ com o uso de padrões, da prática do TDD e de geradores de aplicações configuráveis. Com a sua camada de interface gráfica, o framework GRENJ pode ser utilizado no desenvolvimento de sistemas voltados para a web que pertencem ao domínio de gestão de recursos de negócios. Além disso, essa camada é constituída de outro framework, o Guiwe (do inglês, ***G****raphical* ***u****ser* ***i****nterface for* ***we****b*), que pode ser instanciado para apoiar a construção da camada de interface gráfica web de EAFs ou de sistemas de informação pertencentes a qualquer domínio. A ferramenta GRENJ-Gens permite o uso de um wizard baseado em geradores de aplicações para o desenvolvimento automatizado de sistemas de informações instanciados a partir do framework GRENJ. Dessa forma, a complexidade do uso desse framework é amenizada e o processo de construção de sistemas torna-se mais eficiente. Existe, ainda, a possibilidade dessas ferramentas serem aplicadas na realização de diversos experimentos em conjunto com outras técnicas ou produtos de pesquisa relacionados com a linguagem Java e com as tecnologias voltadas para a web.

As contribuições e limitações deste trabalho são descritas nas Seções 7.2 e 7.3, respectivamente. Na Seção 7.4 são indicadas sugestões de trabalhos futuros que podem ser realizados como complementação deste aqui apresentado.

7.2. Contribuições

Este trabalho propõe o uso do framework Guiwe para a construção da camada de interface gráfica de sistemas de informação e de EAFs voltados para web. A aplicação desse framework define um processo em que suas classes são estendidas para contemplar as características específicas das classes da camada de negócios do software alvo. Esse processo pode ser realizado por meio de iterações e com a aplicação de prática do desenvolvimento guiado por testes.

O framework Guiwe possui também uma camada de controle que interliga a sua camada de interface gráfica com a de negócios do sistema ou do EAF sobre o qual foi aplicado. Essa camada é independente de sua camada de interface gráfica, de modo que pode ser utilizada isoladamente no desenvolvimento de um sistema de informação. Sua interface exige que o nome da operação, o nome da classe e os nomes e os valores dos atributos sejam informados. Em geral, os objetos da camada de negócios necessitam implementar as quatro operações para a persistência de dados: inserção, atualização, remoção e seleção. Entretanto, a classe responsável por essa restrição pode ser estendida para que outras operações possam ser contempladas. Dessa forma, o desenvolvedor pode optar em criar sua própria camada interface gráfica e utilizar a camada de controle do framework Guiwe.

O desenvolvimento de sistemas instanciados a partir do framework GRENJ é realizado por um processo constituído de quatro passos principais: 1) a criação de um modelo de classes com base nos requisitos do sistema e com a aplicação dos padrões da GRN; 2) o preenchimento dos formulários do wizard relacionados com as classes que foram criadas no modelo; 3) a geração do sistema; e 4) caso os requisitos do sistema indiquem uma funcionalidade que não é prevista pelo framework, essa funcionalidade deve ser acrescentada ao código do sistema gerado.

O framework GRENJ apóia o desenvolvimento de sistemas voltados para a web que pertencem ao domínio definido pela linguagem de padrões GRN. Porém, o domínio dessa linguagem de padrões é extenso e pode ser abstraído para outros subdomínios como, por exemplo, o de clínicas de saúde, o de clínicas veterinárias, o de leilões, o de hotelarias, entre outros. Dessa forma, o framework GRENJ também pode ser estendido para esses subdomínios com a criação de classes abstratas, com base naquelas existentes no framework, para que sejam abordadas as características específicas e variantes desses subdomínios.

Com o uso do framework GRENJ o desenvolvimento de sistemas se torna mais eficiente. Os artefatos de uma aplicação gerada com o uso do wizard do framework GRENJ são organizados no formato de um projeto da IDE Netbeans (Sun Microsystems, 2009g) e podem ser modificados para contemplar requisitos que não são previstos pelo framework. Essa característica permite maior flexibilidade às aplicações desenvolvidas com o uso do framework GRENJ.

A documentação interna do código do framework GRENJ segue, como ocorreu nas camadas de persistência e de negócios, com a utilização de *tags javadoc* (Sun Myconsystems, 2009g). Nas camadas de interface gráfica e de controle, essas informações foram originadas das listas de casos de testes das classes do framework e descrevem as finalidades de seus métodos e suas interfaces, além de detalhes como o tipo e o significado dos valores de retorno, o autor da classe e a sua versão. Dessa forma, o desenvolvedor tem acesso a uma lista dos métodos existentes em cada classe, o que pode auxiliá-lo, em conjunto com o manual do framework, a identificar e a configurar os pontos variáveis corretamente.

Quando o usuário de um sistema instanciado a partir do framework GRENJ solicita que uma operação seja efetuada como, por exemplo, a gravação ou a alteração dos dados de um registro da base de dados, a sua camada de interface gráfica lhe retorna uma mensagem de confirmação ou de falha na execução dessa operação. Por definição, todas as mensagens que podem ser apresentadas aos usuários dos sistemas estão escritas, em inglês, no arquivo *app.propertities*. Por meio desse arquivo, é possível alterar o texto das mensagens. Além dessa possibilidade, os rótulos, textos e cabeçalhos da interface gráfica do framework podem ser customizados nas classes do sistema que está sendo instanciado. Com isso, garante-se a internalicionalização da linguagem natural dos textos da interface gráfica dos sistemas web instanciados com base no framework GRENJ.

O framework GRENJ foi desenvolvido com o uso da linguagem Java (Sun Mycrosystems, 2009a) e com tecnologias voltadas para a web. Essas tecnologias garantem a característica multiplataforma dos sistemas instanciados a partir desse framework. Além disso, as customizações realizadas pela ferramenta GRENJ-Gens permitem a configuração dos caminhos dos softwares auxiliares em qualquer computador, independentemente de seu hardware ou de seu sistema operacional, bem como a seleção de diferentes geradores de aplicações que possam utilizar os gabaritos XSL do framework GRENJ.

O registro e a seleção de diferentes geradores de aplicações na ferramenta GRENJ-Gens para a instanciação automatizada de sistemas baseados no framework GRENJ propicia maior facilidade na evolução do seu wizard. Pois novos geradores de aplicações, configuráveis ou não, podem ser desenvolvidos para utilizarem os gabaritos XSL, os arquivos de configuração e a LMA do domínio do framework GRENJ. A

independência de gerador permite que manutenções no wizard possam ser realizadas com maior facilidade.

A geração automatizada de sistemas instanciados a partir do framework GRENJ pode ser utilizada para a criação de protótipos. Como um sistema pode ser desenvolvido em menor tempo com o uso do wizard, é possível oferecer uma versão para que os usuários possam utilizar e avaliar. Assim, obtém-se *feedback* mais rapidamente, os requisitos são definidos com maior clareza, ocorre redução no tempo de desenvolvimento e a versão final do sistema tende a possuir maior qualidade. Além disso, o protótipo gerado pode evoluir para contemplar características não previstas pelo wizard ou pelo framework GRENJ.

7.3. Limitações

O framework GRENJ foi implementado para utilizar o Sistema Gerenciador de Banco de Dados (SGBD) MySQL (MySQL, 2009). A alteração desse SGBD requer o uso de outro *driver* de banco de dados para a linguagem Java e de mudanças na sintaxe do código SQL definido dentro das classes do framework. Esse código SQL se encontra espalhado na hierarquia de classes das camadas de persistência e de negócios, portanto, uma alteração do SGBD do framework representa um esforço para modificação do código interno de diversos métodos.

Os geradores de aplicações cadastrados na ferramenta GRENJ-Gens devem, pelo menos, fazer uso dos gabaritos XSL definidos para o domínio do framework GRENJ. Na prática, um gerador de aplicações pode ser construído, ou configurado, para desenvolver sistemas baseados no framework GRENJ sem o uso desses gabaritos e ser cadastrado no GRENJ-Gens. Entretanto, nessa abordagem não é aplicado o conceito de reúso. A situação ideal é que os geradores de aplicações reutilizem tanto os gabaritos XSL quanto a LMA do framework GRENJ.

Outra restrição está relacionada com a impossibilidade de customizar os rótulos e cabeçalhos da interface gráfica dos sistemas gerados a partir do wizard do framework GRENJ, bem como as mensagens apresentadas aos usuários desses sistemas. As customizações desses itens só podem ser realizadas por meio de alterações nas classes do sistema sem o apoio de um gerador de aplicações. Essa restrição também se aplica à apresentação da interface gráfica dos sistemas, que é definida nos documentos CSS do framework GRENJ.

Os testes realizados neste trabalho GRENJ não cobrem todas as possíveis combinações da linguagem de padrões GRN, devido às restrições de tempo do projeto. Além disso, os documentos CSS e JavaScript da camada de interface gráfica e os gabaritos XSL do wizard tiveram de ser verificados sem o apoio de uma ferramenta de testes automatizados, pois não existem ferramentas que permitem a realização desse tipo de teste sobre essas tecnologias.

7.4. Sugestões de Trabalhos Futuros

Os frameworks Guiwe e GRENJ e a ferramenta GRENJ-Gens podem ser utilizados para apoiar o desenvolvimento de sistemas de informação ou outros projetos de pesquisa. Nesta Seção, são apresentadas sugestões de trabalhos futuros que podem amenizar as restrições citadas na Seção anterior ou aumentar os recursos e a qualidade desses produtos.

Adicionar componentes com mais recursos na camada de interface gráfica dos frameworks Guiwe e GRENJ: instruções podem ser adicionadas nos documentos JavaScript para fornecer aos campos de texto máscaras para números de telefone, números de documentos, dados monetários e outros valores. Podem, inclusive, ser implementados componentes com calendários para os campos com datas.

Verificar a compatibilidade dos códigos JavaScript e CSS com diversos navegadores: algumas instruções dessas tecnologias são incompatíveis com um navegador ou outro. No JavaScript, essas instruções devem ser substituídas por outra que seja compatível com a maioria dos navegadores ou estar agrupadas em cláusulas que verificam qual navegador está sendo utilizado. Nos documentos CSS, são adicionadas instruções que definem a mesma apresentação para cada navegador.

Implementar suporte a outros SGBDs no framework GRENJ: para permitir que o desenvolvedor possa escolher o SGBD utilizado por sistema que foi instanciado com base no framework GRENJ é necessário que sejam fornecidos diversos drivers de comunicação entre o sistema e o SGBD. Além disso, devem sem implementados mecanismos que permitem alterar as instruções SQL existentes dentro das classes Java. Uma forma de se fazer isso é por meio de Aspectos, que podem agrupar o código SQL espalhado do framework. Outra solução pode ser o uso de frameworks voltados para o domínio de persistência.

Implementar suporte a outros SGBDs no wizard: os gabaritos XSL, os arquivos de configuração e a LMA do framework GRENJ foram construídos para oferecer suporte apenas para bases de dados implementadas em MySQL. Outros gabaritos XSL podem ser construídos para dar origem ao script de criação da base de dados dos sistemas para diferentes SGBDs e os formulários do wizard podem ser modificados para oferecer a opção de escolha. A ferramenta GRENJ-Gens também deve ser modificada para armazenar informações sobre diversos SGBDs.

Alterar o wizard do framework GRENJ para permitir customizações na interface gráfica dos sistemas: não é possível customizar a apresentação da interface gráfica dos sistemas nem os rótulos dos campos e de algumas opções no menu principal dos sistemas instanciados com base no framework GRENJ gerados com o apoio do wizard. Essas e outras opções podem ser gradualmente incrementadas na LMA do framework GRENJ.

Alterar o wizard do framework GRENJ para aumentar a flexibilidade dos relatórios dos sistemas: os relatórios gerados pelo wizard do framework GRENJ são limitados. Customizações e filtros nos relatórios exigem uma implementação complexa para geração dos códigos SQL com seleção de campos e de tabelas e com definição de condições.

Eliminar as dependências dos testes com a base de dados: os testes criados durante o desenvolvimento do framework GRENJ comunicam-se com a base de dados a fim de avaliar a inserção, remoção e atualização dos dados contidos nos objetos. Porém, é necessário que a base de dados seja previamente construída para que esses testes funcionem. Esse problema poderia ser eliminado por meio da utilização de objetos *mock* para a simulação da interação com o banco de dados. Dessa forma, a execução dos testes do framework GRENJ seria facilitada, pois eliminaria a necessidade de se configurar o banco de dados com as tabelas avaliadas pelos testes.

Adicionar o conceito de serviços ao framework GRENJ: diversas funções realizadas pelos sistemas instanciados a partir do framework GRENJ podem ser estendidas para uma arquitetura orientada a serviços. Isso pode garantir maior flexibilidade na configuração dos pontos variáveis do framework e na inclusão de novas funções nos sistemas.

Referências

Aarsten, A., Brugali, D. e Menga, G. (1997). The Framework Life Span: a Case Study for Flexible Manufacturing Systems. Communications of the ACM. Disponível em: http://www.polito.it/~cim/Articles/Articles/cacm97fw.ps.

Abi-Antoun, M. (2007). *Making Frameworks Work: A Project Retrospective*. In: Conference on Object Oriented Programming Systems Languages and Applications, p. 1004-1018.

Asleson, R. e Schutta, N. T. (2006). *Foundations of AJAX*. Apress L. P.

Beck, K. (2001). *Aim, Fire*. Thought Works, IEEE Computer Society, p. 87-89.

Braga, R. T. V. (2002). *Um Processo para Construção e Instanciação de Frameworks Baseados em uma Linguagem de Padrões para um Domínio Específico*. Tese de doutorado, ICMC/USP, São Carlos - SP.

Braga, R. T. V., Germano, F. S. R. e Masiero, P. C. (1999). *GRN: Uma Linguagem de Padrões para Gerenciamento de Recursos de Negócio*. Disponível em: *http://www.icmc.usp.br/~rtvb/*. Acessado em 06 de Novembro de 2007.

Braga, R. T. V. e Masiero, P. C. (2002). *A Process for Framework Construction Based on a Pattern Language*. In: 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment (COMPSAC'02), p. 615-622.

Braga, R. T. V. e Masiero, P. C. (2004a). *Building a Wizard for Framework Instantiation Based on a Pattern Language*. In: 9th International Conference on Object-Oriented Information Systems (OOIS 03), Genebra, Suiça. Lecture Notes on Computer Science, LNCS 2817, Springer, Setembro, p. 95-106.

Braga, R. T. V. e Masiero, P. C. (2004b). *Finding Frameworks Hot Spots in Pattern Languages*. Journal of Object Technology, vol. 3, no. 1, p. 123-142. Disponível em: http://www.jot.fm/issues/issue_2004_01/article2. Data de Acesso: 05 de Novembro de 2007.

Brugali, D. e Sycara, K. (2000). *Frameworks and Pattern Languages: an Intriguing Relationship.* ACM Computing Surveys, vol. 32, article no. 2.

Bushmann, F., Henney, K. e Schmidt, D. C. (2007). *Past, Present and Future Trends in Software Patterns*. IEEE Software, vol. 24, p. 31-37.

Chaves, A.P.; Leal, G.C.L.; Huzita, E.H.M. (2008). *An Experimental Study of the FIB Framework Driven by the PDCA Cycle.* In: International Conference of the Chilean Computer Science Society (SCCC '08), p. 23-31.

Cleaveland, J. C. (2001). *Program Generators with XML and Java*. Prentice Hall.

Clements, P. e Northrop, L. M. (2001). *Software Product Lines: Practices and Patterns.* SEI Series in Software Engineering. Addison-Wesley Professional.

Crane, D., Pascarello, E. e James, D. (2006). *Ajax in Action*. Manning Publications.

Coplien, J. (1996). *Software Patterns*. SIGS Books. Disponível em: http://users.rcn.com/jcoplien/Patterns/WhitePaper/SoftwarePatterns.pdf. Data de acesso: 26 de Agosto de 2007.

Durelli, V. H. S. (2008). *Reengenharia Iterativa do Framework GREN*. Dissertação de Mestrado. PPGCC – UFSCAR, São Carlos – SP.

Durelli, V. H. S., Viana, M. C. e Penteado, R. A. D. (2008). *Uma Proposta de Reúso de Interface Gráfica com o Usuário Baseada no Padrão Arquitetural MVC*. In: IV Simpósio Brasileiro de Sistemas de Informação, Rio de Janeiro - RJ. Anais SBSI 2008, p. 48-59.

Fayad, M. E. e Schmidt, D. C. (1997). *Object-Oriented Application Frameworks*. Communications of the ACM, vol. 40, no. 10, p. 32-38.

Fowler, M. (1997). *Analysis Patterns: Reusable Object Models.* Addison-Wesley.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.

Freeman, E., Freeman, E., Sierra, K. e Bates, B. (2004). *Head First Design Patterns*. O'Reilly Media, Inc.

Gamma, E., Helm, R., Johnson, R. e Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gomaa, H. (2005). *Designing Software Product Lines with UML*. Addison-Wesley.

Harrison, N. B., Avgeriou, P. e Zdun, U. (2007). *Using Patterns to Capture Architectural Decisions*. IEEE Software, vol. 24, no. 4, p. 38-45.

Herrington, J. (2004). *Extensible Code Generation with Java*. Part 2. Disponível em: http://today.java.net/pub/a/today/2004/05/31/generation-pt2.html. Data de Acesso: 22 de Janeiro de 2008.

Janzen, D. e Saiedian, H. (2005). *Test-Driven Development: Concepts, Taxonomy and Future Direction*. IEEE Software, vol. 38, no. 9, p. 43-50.

Janzen, D. e Saiedian, H. (2008). *Does Test-Driven Development Really Improve Software Quality*. IEEE Software, vol. 25, no. 2, p. 77-84.

Jensen, P. (2009). *Experiences with Software Product Line Development.* CrossTalk: The Journal of Defense Software Engineering, janeiro 2009. Disponível em: http://tactical.overwatch.com/pdfs/news/2009/ExperiencesWithSPLdevelopment_Jan09.pdf. Data de acesso: 08 de Março de 2009.

Johnson, R. E. (1997a). *Frameworks = (Components + Patterns)*. Communications of the ACM, vol. 40, no. 10, p. 39-42.

Johnson, R. E. (1997b). *Components, Frameworks, Patterns*. In: 1997 Symposium on Software Reusability, ACM, p. 10-17.

Kerievsky, J. (2004). *Refactoring to Patterns*. Addison-Wesley.

Kircher, M. e Völter, M. (2007). *Guest Editors' Introduction: Software Patterns*. IEEE Software, vol. 24, no. 4, p. 28-30.

Kirk, D., Roper, M. e Wood, M. (2002). *On the Creation of Pattern Languages for Framework Reuse*. Disponível em: http://citeseer.comp.nus.edu.sg/649055.html. Data de Acesso: 01 de Dezembro de 2007.

Larman, C. (2004). *Utilizando UML e Padrões: Uma introdução à análise e ao projeto orientados a objetos e ao Processo Unificado*. 2ª Edição, Bookman.

Liem, I. e Nugroho, Y. (2008). An application generator framelet. In: Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD '08), p. 794-799.

Manolescu, D., Kozaczynski, W., Miller, A. e Hogg, J. (2007). *The Growing Divide in the Patterns World*. IEEE Software, vol. 24, no. 4, p. 61-67.

Myers, G. J., Sandler, C., Badgett, T. e Thomas, T. M. (2004). *The Art of Software Testing*. Wiley.

MySQL (2009). *MySQL: The World's Most Popular Open Source Database*. Disponível em: http://www.mysql.com. Data de acesso: 15 de Fevereiro de 2009.

Pacios, S. F. (2006). *Uma abordagem Orientada a Aspectos para Desenvolvimento de Linhas de Produtos de Software*. Dissertação de Mestrado, USP-ICMC, São Carlos – SP.

Pazin, A. (2004). *GAwCRe: Um Gerador de Aplicações baseadas na Web para o Domínio de Clínicas de Reabilitação*. Dissertação de Mestrado, PPGCC-UFSCar, São Carlos – SP.

Pressman, R. S. (2006). *Engenharia de Software*. 6ª Edição, McGraw-Hill.

Ré, R., Braga, R. T. V. e Masiero, P. C. (2001). *A Pattern Language for Online Auctions Management*. In: PLoP'2001, 8th Conference on Pattern Language of Programs, Monticello, IL, EUA, p. 1-18.

Red Rat (2009). *Hibernate*. Disponível em: http://www.hibernate.org. Data de acesso: 22 de Fevereiro de 2009.

Resig, J. (2009). *JQuery JavaScript Library*. Disponível em: http://jquery.com. Data de acesso: 22 de Fevereiro de 2009.

Roberts, D. e Johnson, R. E. (1997). *Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks*. Disponível em: http://citeseer.ist.psu.edu/roberts96evolving.html. Data de acesso: 03 de Dezembro de 2007.

Santos, M. (2004). *Uma Proposta para a Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas*. Dissertação de Mestrado. Departamento de Computação - UFC, Fortaleza – CE.

Schmidt, D. C., Fayad, M. e Johnson, R. E. (1996). *Software Patterns*. Communications of the ACM, vol. 39, no. 10, p. 37-39.

Shalloway, A. e Trott, J. (2001). *Design Patterns Explained: A New Perspective On Object-Oriented Design*. Addison-Wesley.

Shimabukuro Junior, E. K. (2006). *Um Gerador de Aplicações Configurável*. Dissertação de Mestrado. Instituto de Ciências Matemáticas e de Computação – ICMC/USP, São Carlos – SP.

Shiva, S. G. e Shala, L. A. (2007). *Software Reuse: Research and Pratice*. Fourth International Conference on Information Technology, pages 603–609.

Smalltalk Dot Org (2009). *Smalltalk Dot Org: Community and industry meet inventing the future*. Disponível em: http://www.smalltalk.org/main. Data de Acesso: 02 de Março de 2009.

Spring Source (2009). *Spring Source*. Disponível em: http://www.springsource.com. Data de Acesso: 15 de Fevereiro de 2009.

Sugumaran, V., Park, S. e Kang, K. C. (2006). *Software Product Line Engineering*. Communications of the ACM, vol. 49, no. 12, p. 29-32.

Sun Microsystems (2009a). *The Source for Java Developers*. Disponível em: http://java.sun.com. Data de Acesso: 15 de Fevereiro de 2009.

Sun Microsystems (2009b). *JavaServer Pages Technology*. Disponível em: http://java.sun.com/products/jsp. Data de Acesso: 15 de Fevereiro de 2009.

Sun Microsystems (2009c). *Java Servlet Technology*. Disponível em: http://java.sun.com/products/*servlet*. Data de Acesso: 15 de Fevereiro de 2009.

Sun Microsystems (2009d). *Java Persistence API*. Disponível em: http://java.sun.com/developer/technicalArticles/J2EE/jpa. Data de Acesso: 15 de Fevereiro de 2009.

Sun Microsystems (2009e). *JavaServer Faces Technology*. Disponível em: http://java.sun.com/javaee/javaserverfaces. Data de Acesso: 15 de Fevereiro de 2009.

Sun Microsystems (2009f). *Javadoc Technology*. Disponível em: http://java.sun.com/javase/6/docs/technotes/guides/javadoc/index.html. Data de Acesso: 22 de Fevereiro de 2009.

Sun Microsystems (2009g). *Welcome to NetBeans*. Disponível em: http://www.netbeans.org. Data de Acesso: 22 de Fevereiro de 2009.

The Apache Software Foundation (2009a). *Apache Tomcat*. Disponível em: http://tomcat.apache.org. Data de acesso: 15 de Fevereiro de 2009.

The Apache Software Foundation (2009b). *Apache Struts*. Disponível em: http://struts.apache.org. Data de acesso: 15 de Fevereiro de 2009.

The Apache Software Foundation (2009c). *Apache Ant*. Disponível em: http://ant.apache.org. Data de acesso: 22 de Fevereiro de 2009.

Thibault, S. e Consel, C. (1997). *A Framework for Application Generator Design*. ACM.

W3C (2009). *W3Schooll*. Disponível em: http://www.w3school.com. Data de Acesso: 16 de fevereiro de 2009.

Weiss, D. M. e Lai, C. T. R. (1999). *Software Product Line Engineering: A Family-Based Software Development Process*. Addison-Wesley.

Williams, L., Maximilien, E. M. e Vouk, M. (2003). *Test-Driven Development as a Defect-Reduction Practice*. In: 14th International Symposium on Software Reliability Engineering (ISSRE'03). IEEE Software, p. 34-48.

Wu, J. H., Hsia, T. C., Chang, I. C. e Tsai, S. J. (2003). *Application Generator: A Framework and Methodology for IS Construction*. In: 36th Hawaii International Conference on System Sciences. IEEE Software.

Yoder, J. W., Johnson, R. E., Wilson, Q. D., e Douglas, M. (1998). *Connecting Business Objects to Relational Databases*. Fifth Conference on Patterns Languages of Programs (PLoP '98).