Peônia: um ambiente web para apoiar processos de desenvolvimento com utilização de padrões de software e requisitos de teste no projeto de aplicações

Alessandra Chan

Peônia: um ambiente web para apoiar processos de desenvolvimento com utilização de padrões de software e requisitos de teste no projeto de aplicações

Alessandra Chan

Orientadora: ***Profa. Dra. Rosana T. V. Braga***

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

USP - São Carlos
Fevereiro/2008

Dedico este trabalho à minha família: meu pai, minha mãe e meu irmão.

Agradecimentos

Agradeço, primeiramente, a Deus que me deu a vida e está presente em cada passo que eu dou.

À Prof. Dra. Rosana Terezinha Vaccare Braga pelo apoio, paciência e confiança durante a orientação desta dissertação, mas principalmente por ter me ouvido sempre.

Aos Profs. Drs. José Carlos Maldonado e Maria Istela Cagnin por terem me incentivado e ajudado, desde os tempos de iniciação científica, na minha carreira acadêmica.

Aos meus pais que sempre me apoiaram, em todas as ocasiões e de todas as formas possíveis.

Ao meu irmão pela companhia e amizade, além dos momentos de descontração, fazendo-me rir.

Aos meus amigos do Labes, pelo companheirismo e diversão proporcionada pela tênue relação de amor e ódio entre todos os seus membros, sem exceção.

Aos meus amigos da Comp2K, pelos grandes momentos vividos durante a graduação e os inclassificáveis churrascos.

A todos os meus amigos de São Paulo e São Carlos, que sempre ouviram meus dramas com uma palavra de conforto e incentivo. Não citarei nomes para não ser injusta com ninguém. Cada um sabe o papel importante que teve para a conclusão deste projeto.

Aos colegas de trabalho, por terem me ajudado, mesmo que indiretamente, a enriquecer esta pesquisa.

Ao CNPq pelo apoio financeiro.

Resumo

A expansão e a popularização da *World Wide Web* têm incentivado o desenvolvimento de aplicações *Web*. É crescente a exigência por aplicações *Web* cada vez mais complexas, cujo desenvolvimento deve ser feito com qualidade e rapidez. Para orientar o ciclo de vida dessas aplicações, diversos métodos de desenvolvimento *Web* têm sido criados, além de ferramentas de apoio a sua utilização. O emprego de padrões de software no desenvolvimento de aplicações pode aumentar a produtividade e a qualidade. Com o objetivo de minimizar erros e facilitar a utilização, ferramentas de apoio à utilização de padrões têm sido desenvolvidas. No entanto, há uma carência por ambientes e ferramentas que apóiem o emprego de padrões durante as etapas do desenvolvimento de aplicações. Algumas das principais atividades para a garantia da qualidade de software são as de VV&T (Verificação, Validação e Teste). Algumas iniciativas de associação de teste a padrões têm sido estudadas com o intuito de minimizar o tempo despendido em VV&T. Neste trabalho é apresentado o ambiente Peônia com o intuito de apoiar o emprego de padrões de software durante as etapas de um processo de desenvolvimento. Para isso, os padrões de software podem ser previamente associados a essas etapas para que possam ser sugeridos durante a execução de um projeto. Além disso, o ambiente Peônia oferece a possibilidade de associar requisitos de teste a padrões de software, para auxiliar nas atividades de VV&T. Também é proposto um Método Para Desenvolvimento Utilizando Padrões de Software, formalizando a técnica empregada no ambiente Peônia durante a execução de projetos, onde é incentivada a utilização de padrões de software na criação de artefatos e execução de fases e atividades. O método proposto estimula usuários a empregar padrões de software durante as etapas do processo de desenvolvimento, independentemente de utilizar, ou não, o ambiente Peônia.

Abstract

The expansion and popularization of the World Wide Web have encouraged the development of Web applications. There is an increasing demand for more complex Web applications, whose development must be done with quality and urgency. Several Web development methods have been created to guide the life-cycle of these applications, as well as tools supporting their use. The use of patterns in the development of software applications can enhance productivity and quality. In order to minimize errors and facilitate use, tools supporting the use of patterns have been developed. But there is a lack of environments and tools supporting the use of patterns during all stages of applications development. Some of the main activities to ensuring the quality of software are VV&T (Verification, Validation and Testing). Some initiatives associating test to patterns have been studied in order to minimize the time spent in VV&T. This work presents the Peony environment in order to support the use of software patterns during the development process stages. Therefore, the software patterns can be previously associated with these stages so they can be suggested during project execution. Moreover, the Peony environment offers the possibility to associate test requirements to software patterns, helping VV&T activities. It is also proposed a Method for Development Using Software Patterns, which formalises the technique employed in Peony environment during projects execution, encouraging the use of software patterns in the artifact creation, as well as during phases and activities execution. The proposed method encourages users to employ software patterns during the development process stages, regardless using, or not, the Peony environment.

Uma pessoa inteligente aprende com seus erros, uma pessoa sábia aprende com os erros dos outros.
Autor Desconhecido

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas	
ACM	*Association for Computing Machinery*
AJAX	*Asynchronous JavaScript and XML*
API	*Application Programming Interface*
CASE	*Computer Aided Software Engineering*
CGI	*Common Gateway Interface*
CGI/PERL	*Common Gateway Interface*/PERL
CORBA	*Common Object Request Broker Architecture*
CRUD	*Create, Retrieve, Update e Destroy*
DAO	*Data Access Objects*
DHTML	*Dynamic HyperText Markup Language*
DOM	*Document Object Model*
DS	*Distinguishing Sequences*
ECO	*Ecosystem of Agile Software Development*
EJB	*Enterprise JavaBeans*
EL	*Expression Language*
FRED	*FRamework EDitor*
GoF	*Gang of Four*
GoV	*Gang of Five*
GREN	Gestão de REcursos de Negócio
GRN	Gestão de Recursos de Negócio
GUI	*Graphical User Interface*
HDM	*Hypermedia Design Model*
HQL	*Hibernate Query Language*
HMBS/M	*Hypertext Model Based on Statecharts/Method*
HMBS/M Estendido	*Hypertext Model Based on Statecharts/Method* Estendido
HPR	*Hypermedia Design Patterns Repository*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
ID	*Identificador*
IBM	*International Business Machines*
J2EE	*Java 2 Platform, Enterprise Edition*
J2ME	*Java Platform, Micro Edition*
Java/ORB	Java/*Object Request Broker*
JDBC	*Java Database Connectivity*
JMS	*Java Message Service*
JNDI	*Java Naming and Directory Interface*
JTA	*Java Transaction API*
LGPL	*GNU Lesser General Public License*
MEF	Máquina de Estados Finitos
MobileUI	*Mobile User Interface*
MVC	*Model View Controller*
MySQL	*My Structured Query Language*

Lista de Siglas	
OCL	*Object Constraint Language*
ODE	Ontology-based software Development Environment
OO	Orientado a Objetos
OO-H	Object-Oriented Hypermedia
OOHDM	*Object-Oriented Hypermedia Design Method*
OOWS	*Object-Oriented Web Solution*
OP	*Objectory Process*
ORB	*Object Request Broker*
PDA	*Personal Digital Assistant*
POSA	*Pattern-Oriented Software Architecture*
PSEE	*Process-Centered Software Engineering Environment*
RMM	*Relationship Management Methodology*
RUP	*Rational Unified Process*
SGBD	*Sistema de Gerenciamento de Banco de Dados*
SIGWEB	*Special Interest Group on Hypertext, Hypermedia and Web*
SQL	*Structured Query Language*
SWM	*Simple Web Method*
UHDM	*UML-based Hypermedia Design Method*
UIO	*Unique-Input-Output*
UML	*Unified Modeling Language*
UP	*Unified Process*
URL	*Universal Resource Locator*
UWE	*UML-based Web Engineering*
UX	*Experiência do Usuário*
VV&T	Verificação, Validação e Teste
W2000	*UML Extension for Modeling Web Application*
WAE	*Web Application Extension*
Web	*World Wide Web*
WebAPSEE	*Web Process-Centered Software Engineering Environments*
WebML	*Web Modeling Language*
Wp	*Partial W-Method*
XAML	*Extensible Application Markup Language*
XHTML	*Extensible HyperText Markup Language*
XMI	*Extensible Markup Language Metadata Interchange*
XML	*Extensible Markup Language*
XSL	*Extensible Stylesheet Language*
XSLT	*Extensible Stylesheet Language Transformations*
XUL	*XML User Interface Language*
ZUML	*ZK User Interface Markup Language*

Capítulo

1

Introdução

1.1 Contexto

A necessidade de aumentar a produtividade e a qualidade de software incentivou, a partir da década de 90, a reutilização de soluções bem sucedidas para problemas de desenvolvimento de software. Padrões de software são utilizados para descrever tais soluções (Gamma et al., 1995).

Quando especialistas se confrontam com um problema particular é raro que elaborem uma nova solução, completamente diferente das já existentes, para solucioná-lo. Freqüentemente recordam dificuldades similares e reusam soluções antigas, pensando em pares "problema/solução" (Buschmann, F.; et al., 1997).

Uma grande quantidade de padrões para desenvolvimento de software está disponível na literatura atual, assim como ferramentas que auxiliam os engenheiros de software a empregarem esses padrões na criação de suas aplicações (Marinho et al., 2003). No entanto, poucos trabalhos são encontrados sobre associação de testes a esses padrões, como por exemplo, os desenvolvidos por Cagnin et al. (2005, 2004).

Algumas das principais atividades para a garantia de qualidade são as de VV&T (Verificação, Validação e Teste), minimizando a ocorrência de erros e riscos associados (Rocha et al., 2001). Essa atividade constitui uma das mais onerosas de um projeto, podendo ocupar até cinco vezes mais do que a soma das outras atividades de engenharia de software (Pressman, 2002), e, no mundo globalizado e competitivo atual, cresce a

importância do desenvolvimento de software de alta qualidade, com preços acessíveis e em tempo reduzido.

Dentro do contexto de globalização, podemos destacar a expansão e a popularização da *Web* (*World Wide Web*). A *Web* pode ser definida como o universo de informação acessível pela rede mundial. Constitui um espaço abstrato que permite a comunicação entre pessoas e máquinas, preenchido por páginas de textos, imagens, animações, sons, vídeos e mundos tridimensionais interligados (Berners-Lee, 1996).

Para evitar problemas no desenvolvimento, implantação, operação e manutenção de aplicações *Web*, é necessária a existência de um processo de desenvolvimento disciplinado (Domingues, 2005). A Engenharia *Web* propõe métodos, técnicas e ferramentas para a construção de aplicações *Web* com qualidade (Brito, 2003). Alguns dos métodos de desenvolvimento *Web* encontrados atualmente na literatura são o HDM (*Hypermedia Design Model*) (Garzotto et al., 1991, 1993), o RMM (*Relationship Management Methodology*) (Isakowitz et al., 1995), OOHDM (*Object-Oriented Hypermedia Design Method*) (Rossi, 1996; Schwabe, 2005), HMBS/M (*Hypertext Model Based on Statecharts/Method*) (Carvalho, 1998; Carvalho et al., 1999), W2000 (*UML Extension for Modeling Web Application*) (Baresi et al., 2001), UWE (*UML-based Web Engineering*) (Koch, 2000), WebML (*Web Modeling Language*) (Ceri et al., 2000) e o WAE (*Web Application Extension*) (Conallen, 2002).

Ferramentas automatizadas possuem um papel importante no emprego correto desses métodos e de processos, apoiando o projeto, desenvolvimento, análise e avaliação de aplicações (Brito, 2003). Além da importância quanto à utilização correta de métodos e processos, ferramentas ou mecanismos que automatizem o reúso são necessários para que o emprego de padrões de software seja realizado de maneira eficiente (Santos, 2004).

Diversas tecnologias podem ser encontradas atualmente para a criação de aplicações *Web*, sendo que as mais recentes utilizam as vantagens fornecidas pelo paradigma OO (Orientado a Objetos). Para o desenvolvimento *Web*, a linguagem de programação Java tem sido amplamente utilizada. Segundo Santos (2004), a linguagem Java aliada as tecnologias J2EE (*Java 2 Platform, Enterprise Edition*) (Sun Microsystems, Inc., 2006) são adequadas para a criação de componentes de sistemas que possuem como requisitos a manutenibilidade, escalabilidade e suporte a concorrência. De acordo com as definições da Sun *Microsystems*, em um dos primeiros documentos de apresentação da linguagem, Java é uma linguagem de programação orientada a objetos, simples, robusta, interpretada, portável, distribuída, de arquitetura neutra, segura, de alto desempenho, *multithreaded* e dinâmica (Sun Microsystems, Inc., 1999).

1.2 Motivação

Padrões constituem uma técnica eficaz de reúso, mas são complexos, requerendo um alto custo para a introdução em projetos. Por isso, é necessário que os projetistas tenham um conhecimento prévio de diversos padrões para que utilizem os mais adequados para a solução de um problema (Sommerville, 2003). Além disso, existe uma grande quantidade de padrões na literatura atual, sendo que fazem uso de normas distintas de nomenclatura e definição, dificultando a busca pelo padrão adequado (Pressman, 2002).

Assim como o processo de engenharia da *Web*, o teste de aplicações *Web* também requer métodos específicos que complementem a validação das suas características específicas. Atualmente existem estudos de técnicas de teste específicas para aplicações *Web*, por exemplo, os citados em Ricca e Tonella (2001); Xu et al. (2005). No entanto, como mencionado anteriormente, poucos estudos são encontrados sobre a associação de testes a padrões como maneira de facilitar a validação de aplicações que os utilizem e para comprovar a qualidade do emprego desses padrões (Cagnin et al., 2005, 2004).

Foram encontrados na literatura atual, repositórios de padrões (Bolchini et al., 2002; Garzotto et al., 1999; Marinho et al., 2003) e ferramentas que auxiliam no emprego de padrões, possibilitando inclusive a geração automática de código (Braga, 2003; Braga e Masiero, 2000; Hakala et al., 2001b). No entanto, não foram encontrados ambientes que fornecessem requisitos de testes para facilitar a validação de sistemas desenvolvidos utilizando padrões.

Também foram encontradas na literatura ferramentas que apóiam a utilização de processos (Bertollo et al., 2006; Falbo et al., 2004; Lima et al., 2006) e métodos de desenvolvimento *Web* (Brito, 2003; Knapp et al., 2003, 2004a, 2005, 2004b; Schwabe e Pontes, 1998; Schwabe et al., 1999; Turine et al., 1998, 1999; Web Models, 2006) e de automatização de teste (Chaim, 1991; Delamaro, 1993; Horgan e Mathur, 1992; Price e Zorzo, 1990; Vincenzi et al., 2003), contudo, essas ferramentas não proporcionam auxílio suficiente ao emprego de padrões de software.

Assim, um ambiente que auxilie no emprego de processos e métodos de desenvolvimento e de padrões de software, além de ajudar a atividade de teste de aplicações, pode ser útil para minimizar o tempo despendido na criação dessas aplicações. Além disso, o estudo, ainda escasso, de associação de requisitos de testes a padrões de software pode auxiliar a validar a eficiência do emprego de padrões.

1.3 Objetivos

O principal objetivo deste trabalho é fornecer um método e um ambiente que incentivem o emprego de padrões de software durante a execução de um processo de desenvolvimento. Esse ambiente possibilita que os usuários acrescentem processos de desenvolvimento, padrões de software e requisitos de testes e permite que sejam associados, para que durante a execução de projetos eles sejam sugeridos, estimulando a sua utilização.

Esse ambiente, denominado Peônia, foi desenvolvido como uma aplicação *Web* para explorar, principalmente, a vantagem de permitir que usuários compartilhem as informações contidas no ambiente, além de não restringir a utilização a um local específico e facilitar a divulgação.

O ambiente Peônia foi desenvolvido utilizando as vantagens da linguagem de programação Java, orientado pelo processo proposto por Conallen (2002) e o método de desenvolvimento WAE (*Web Application Extension*) (Conallen, 2002), além de padrões de software existentes na literatura.

Por meio do método proposto e, principalmente, do ambiente Peônia, espera-se que usuários tenham uma maior facilidade para conhecer, aprender e utilizar padrões de software na criação de suas aplicações, empregando-os durante todo o processo de desenvolvimento e tendo diretrizes para validá-lo durante as atividades de VV&T.

1.4 Organização do Documento

Neste capítulo foram apresentados o contexto, a motivação e os objetivos deste trabalho. O restante do documento está organizado da seguinte maneira: no Capítulo 2 há uma breve descrição sobre os conceitos trabalhados neste projeto; no Capítulo 3 são mencionadas ferramentas de apoio ao desenvolvimento existentes na literatura; no Capítulo 4 são comparados alguns *frameworks* que apóiam o desenvolvimento de aplicações e citados os que foram empregados na implementação do ambiente Peônia; no Capítulo 5 são descritas algumas considerações sobre o emprego e teste de padrões e é proposto um *Método Para Desenvolvimento Utilizando Padrões de Software*; no Capítulo 6 são listadas as funcionalidades do ambiente Peônia; no Capítulo 7 é resumido como foi realizado o projeto, inclusive a implementação e teste, do ambiente Peônia; e no Capítulo 8 são realizadas as considerações finais sobre este projeto de mestrado.

Capítulo

2

Conceitos Básicos

2.1 Considerações Iniciais

Com o crescimento acelerado da *World Wide Web*, a rede tornou-se um importante meio de comunicação (Jia e Liu, 2002). As organizações têm utilizado cada vez mais a *Web* para disponibilizar, divulgar e negociar seus produtos e serviços. Além disso, com a globalização da economia, é crescente o processo de migração de sistemas corporativos para plataformas baseadas na *Web* (Brito, 2003).

Assim, é cada vez maior a cobrança pela qualidade de aplicações *Web* e o aumento da dificuldade de projeto, construção e manutenção. Por causa da complexidade e dificuldade, cresce também a chance de ocorrerem falhas no desenvolvimento. Portanto, é crescente a demanda por processos, métodos e ferramentas que forneçam apoio disciplinado a engenheiros de software no desenvolvimento de aplicações.

Diversos esforços podem ser encontrados na literatura com o intuito de resolver ou reduzir os problemas no desenvolvimento de aplicações. Padrões de software são criados para auxiliar projetistas a reutilizar soluções bem sucedidas, baseando-se em experiências anteriores (Gamma et al., 1995), colaborando na difusão de boas práticas de desenvolvimento. No entanto, não existe um consenso quanto ao formato de documentação de padrões (Braga et al., 2001), apesar de alguns componentes serem essenciais. Assim, é grande a dificuldade para encontrar os padrões adequados para serem aplicados em um projeto.

Testes são utilizados como uma das atividades de Garantia de Qualidade de Software (Maldonado, 1991). É impossível encontrar todos os erros de um programa (Myers, 2004). No entanto, técnicas e critérios devem ser combinados para a elaboração de uma abordagem adequada para o teste de um software. No caso de aplicações *Web*, algumas de suas características, como a interatividade, fazem com que a utilização de técnicas de teste tradicionais não sejam suficientes.

Na Seção, 2.2 informações sobre a definição e classificação de aplicações *Web* são apresentadas; na Seção 2.3, processos e métodos de desenvolvimento são definidos e exemplos de métodos que apóiam a criação de aplicações *Web* são listados; na Seção 2.4, padrões de software são definidos, os elementos essenciais de sua estrutura são detalhados e as classificações possíveis são mencionadas; na Seção 2.5, é explicada a importância das atividades de VV&T e são citadas as técnicas de teste de software; e na Seção 2.6 são descritas as considerações finais a respeito dos assuntos apresentados neste capítulo.

2.2 Aplicações Web

Aplicações *Web* são programas de software dinâmicos, interativos e que, geralmente, envolvem manipulação de banco de dados (Jia e Liu, 2002). Outras características que podem ser destacadas são distribuição, hipermídia e multi-plataforma. As aplicações *Web* diferem dos softwares tradicionais por serem executadas em ambientes heterogêneos e autônomos (Xu et al., 2005).

No início, as aplicações baseadas na *Web* eram relativamente simples e empregadas somente para leitura, ou seja, para a transmissão de informações (Ginige e Murugesan, 2001b). Atualmente, a complexidade de aplicações desse tipo tem crescido devido as mudanças nas tecnologias e as dificuldades do negócio (Brito, 2003), podendo exigir a troca e o gerenciamento de uma grande quantidade de dados (Domingues, 2005).

Com o crescimento da complexidade e da importância na vida da população, há uma maior exigência pelo controle da qualidade de aplicações *Web* (Ricca e Tonella, 2001), além do aumento da dificuldade de desenvolvimento, manutenção, gerenciamento e teste desse tipo de aplicação. Assim, é grande a demanda pelo desenvolvimento de métodos e ferramentas para garantir a qualidade de aplicações *Web* (Jia e Liu, 2002; Ricca e Tonella, 2001).

Além de estudos sobre métodos e ferramentas, diversas tecnologias têm surgido para o desenvolvimento dessas aplicações, sendo que a maioria utiliza as vantagens fornecidas pelo paradigma OO (Cho et al., 1997). Aplicações *Web* que utilizam tecnologias convencionais possuem restrições em termos de desempenho, funcionalidade e usabilidade, dentre outros (Cho et al., 1997). Como exemplos de tecnologias *Web* convencionais podem ser citadas

(Cho et al., 1997): HTML (*HyperText Markup Language*) (World Wide Web Consortium, 2005a) e CGI/PERL (*Common Gateway Interface/PERL*) (World Wide Web Consortium, 1999).

Algumas das tecnologias recentes que utilizam as vantagens da OO no desenvolvimento de aplicações *Web* são: Java (Sun Microsystems, Inc., 1999), CORBA (*Common Object Request Broker Architecture*) (OMG's CORBA, 1997), Java/ORB (*Java/Object Request Broker*) (Sun Microsystems and OMG, 2002) e Java/ORB/Agent (Cho et al., 1997).

As aplicações *Web* desenvolvidas com essas tecnologias recentes também possuem suas desvantagens, mas são menores que as criadas utilizando as tecnologias convencionais. Portanto, Java, CORBA e tecnologia de agentes serão o futuro das aplicações *Web*, juntamente com as arquiteturas OO (Cho et al., 1997).

Detalhes a respeito de classificações de aplicações *Web*, encontradas na literatura, são apresentados na Seção 2.2.1. Na Seção 2.2.2 são resumidos os conceitos de arquitetura e explicada a importância do projeto arquitetural.

2.2.1 Classificações de Aplicações Web

Apesar de serem encontradas na literatura diversas classificações para as Aplicações *Web*, não existe uma considerada como padrão (Domingues, 2005).

Para Pressman (2002), uma aplicação *Web* inclui desde páginas que exibem informações, até aplicações que fornecem serviços completos de processamento de transações.

Segundo Conallen (2002) *apud* Domingues (2005), existem os *sites Web* e as aplicações *Web*. Enquanto os *sites Web* apenas exibem informações, as aplicações *Web* são sistemas de informação tradicionais, mas executados na *Web*.

Essa classificação pode ser considerada simples e para complementá-la há a classificação de Araújo (2001) que divide as aplicações *Web* em três tipos:

- Páginas de informação: são as páginas HTML (World Wide Web Consortium, 2005a) ou DHTML (*Dynamic HyperText Markup Language*) (Alain Zarli and Yacine Rezgui, 2000; David Gardner, 2000), podendo conter componentes de animação, sons, vídeos e outros;

- Aplicações *Web* centradas em banco de dados: trabalham com banco de dados, mas não possuem fluxo de processo ou transações complexas;

- Aplicações *Web* que apóiam a realização de transação de negócios: constituem aplicações que realizam alguma computação que auxilie a realização de um negócio. Pode ser uma aplicação em grande escala, dividida em muitos servidores.

Existe uma outra maneira de classificar as aplicações *Web*, considerando os atributos de complexidade e orientação. De acordo com os atributos de complexidade, uma aplicação pode ser do tipo estática ou dinâmica, e, segundo a sua orientação, ela pode ser orientada a documento ou a aplicações (Powell et al., 1998).

Além das classificações apresentadas, outras podem ser encontradas na literatura, contendo pouca variação entre os conceitos. Destaca-se que a comunidade científica oferece pouca atenção a páginas estáticas, que apenas exibem informação, sendo realizados mais estudos nas aplicações *Web* que realizam computação, ou seja, softwares tradicionais, acessados pela *Web* (Domingues, 2005).

2.2.2 Arquiteturas de Aplicações Web

"*A arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema que abrangem os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles*" (Pressman, 2002).

A arquitetura influencia todo o processo de desenvolvimento de um software, fornecendo as regras que devem ser seguidas na sua construção e a maneira de pensar sobre o problema (Conallen, 2002).

Com a elaboração do projeto arquitetural, são desenvolvidas representações que auxiliam o engenheiro de software a realizar a análise da viabilidade desse projeto, para satisfazer os requisitos exigidos; considerar alterações arquiteturais antes de iniciar a fase de desenvolvimento, quando são menos custosas; e reduzir os riscos da construção do software.

O desenvolvimento de aplicações *Web* complexas, como as que integram dados distribuídos e que permitem a utilização de diferentes perfis de usuários, é considerada uma tarefa difícil. É necessário entender o domínio da aplicação e encontrar uma arquitetura de software flexível e adequada para as características inerentes de aplicações *Web*, como a evolução continuada e imediata (Pressman, 2002).

As aplicações *Web* se enquadram na arquitetura de sistemas distribuídos (Domingues, 2005), seguindo o padrão de sistemas cliente/servidor, mas com algumas modificações. Com a difusão da *Web*, sistemas podem ser entregues e desenvolvidos em quase todos os lugares, contudo, existem algumas restrições. A interação de arquiteturas *Web* é feita de acordo com o paradigma de estímulo/resposta, sem o controle direto do cliente no servidor. O cliente deve iniciar a interação com o sistema e, caso esse não seja o comportamento esperado, outros elementos devem ser acrescentados à arquitetura, gerando sistemas mais complexos e, possivelmente, mais sensíveis (Conallen, 2002).

Segundo Bonura et al. (2002), uma arquitetura *Web* pode ser particionada em três camadas:

- *Apresentação* (*front-end*): constitui a aplicação que renderiza a informação para o usuário;
- *Lógica de Negócio* (*business logic*): é a camada responsável por criar às informações exibidas na *Apresentação*, reunindo os dados enviados pela camada de *Dados da Aplicação*;
- *Dados da Aplicação* (*back-end*): composta por fonte de dados com ferramentas para buscar informações e retorná-las.

Em uma arquitetura estática, um *browser* é um exemplo da camada de *Apresentação*, o servidor *Web* pertence a camada de *Lógica de Negócio* e o sistema de arquivos é um exemplo da camada de *Dados da Aplicação* (Bonura et al., 2002).

Baseado nos princípios do método OOHDM (Rossi, 1996; Schwabe, 2005), que divide a modelagem do projeto de acordo com os interesses, separando em modelos conceitual, navegacional e de interface, foram desenvolvidas arquiteturas, como por exemplo, o MVC (*Model View Controller*) (Krasner e Pope, 1988).

A arquitetura MVC é utilizada para melhorar o desenvolvimento de aplicações J2EE (Sun Microsystems, Inc., 2006), pois separa a interface do usuário, dos dados da aplicação e das funcionalidades (Jacyntho et al., 2002). Esse modelo de arquitetura fornece uma interface entre a visualização e o controlador. Os objetos de visualização são responsáveis por armazenar as informações do formato de apresentação do usuário, encaminhando-as para o controlador. O controlador é responsável por processar as entradas do usuário, transformando-as em solicitações para funcionalidades específicas da aplicação. Essa divisão reflete as necessidades de separar a apresentação dos dados da aplicação, pois diferentes maneiras de visualizações podem ser utilizadas pelos usuários, como por exemplo, *browsers* e clientes WAP (*Wireless Application Protocol*) (Leavitt, 2000).

As arquiteturas *Web* estão em constante desenvolvimento, sofrendo a aplicação de mais e mais tecnologias. A utilização de padrões e mecanismos comuns tem ganhado espaço. Como resultado, a tendência é o desenvolvimento de sistemas com arquiteturas consideravelmente complexas (Conallen, 2002).

2.3 Processos e Métodos de Desenvolvimento

Software é um produto complexo, difícil de desenvolver e testar. Freqüentemente, um software pode apresentar comportamentos inesperados e indesejados, podendo causar sérios problemas e perdas. Assim, pesquisadores têm se esforçado para aumentar a qualidade do software. Uma das hipóteses é que há uma relação direta entre a qualidade do processo e a qualidade do software desenvolvido (Fuggetta, 2000).

A qualidade de um software depende das pessoas, organização e procedimento utilizados para criá-lo e entregá-lo. Um processo de desenvolvimento de software é o conjunto de políticas, estruturas organizacionais, tecnologias, procedimentos, atividades e artefatos necessários para conceber, desenvolver, implantar e manter um produto de software (Fuggetta, 2000).

No entanto, não é trivial decidir o que deve ser incluído em um processo de desenvolvimento, pois devem ser consideradas as características da equipe de desenvolvimento e do projeto, o nível de conhecimento em Engenharia de Software, os propósitos da organização, o orçamento disponível, entre outros fatores. Assim, cada organização deve definir os seus processos e melhorá-los constantemente, de acordo com a experiência adquirida durante os projetos (Bertollo et al., 2006). Rocha et al. (2001) consideram três etapas na definição de uma abordagem flexível de processos: definição de processos padrão para a organização, especialização dos processos padrão e instanciação para projetos específicos.

Processos de desenvolvimento de software podem explorar diferentes áreas como: tecnologias de desenvolvimento de software, ou seja, ferramentas, infra-estrutura e ambientes que apóiem a utilização de processos; métodos e técnicas de desenvolvimento de software, com diretrizes sobre como utilizar tecnologias ou efetuar as atividades de desenvolvimento de software; comportamento organizacional, estudando a ciência da organização e das pessoas que a compõe; e economia e *marketing*, analisando as reais necessidades do mercado para que as etapas do processo considerem o contexto onde o software será utilizado e vendido (Fuggetta, 2000).

As pesquisas encontradas na literatura se concentram principalmente nas áreas de métodos e técnicas para modelar e apoiar a execução de processos de desenvolvimento de software. Ambientes que apóiam a criação e utilização de modelos de processos de software são chamados de PSEE (*Process-Centered Software Engineering Environment*) (Fuggetta, 2000).

Processos de desenvolvimento, como os propostos por Pressman (2005) e por Ginige e Murugesan (2001b), não definem um método para a sua utilização (Bianchini, 2007). Segundo o dicionário (Houaiss, 2006), método é o *"procedimento, técnica ou meio de se fazer alguma coisa"*. Um método ensina como desenvolver um software, utilizando como base um conjunto de princípios básicos da Engenharia de Software que abrangem princípios de cada área da tecnologia, incluindo atividades de modelagem e outras técnicas descritivas (Pressman, 2005).

Para o emprego correto de processos e métodos, ferramentas e ambientes podem fornecer apoio automatizado ou semi-automatizado. Diversos estudos podem ser encontrados na literatura atual e alguns deles são citados no Capítulo 3.

Também merecem destaque os estudos voltados para *Web* (*World Wide Web*), pois as suas aplicações têm se tornado muito importantes no mundo de negócios globalizado atual. Assim, também é crescente a preocupação por processos e métodos para desenvolvimento *Web*. No caso de aplicações *Web*, para gerenciar corretamente o desenvolvimento e manutenção, é necessário utilizar técnicas e princípios tradicionais de Engenharia de Software combinados com tratamento dos aspectos específicos da *Web* (Bianchini, 2007; Brambilla et al., 2002).

Os principais motivos que levam a falhas no desenvolvimento de aplicações *Web* não estão relacionados a tecnologia utilizada, mas à ausência de previsão, visão restrita dos objetivos, erros no processo de projeto e de desenvolvimento e gerenciamento pobre dos esforços necessários (Ginige e Murugesan, 2001a).

Para obter aplicações *Web* confiáveis e com bom desempenho, desenvolvedores necessitam de um método seguro, um processo disciplinado e capaz de ser repetido, melhores ferramentas de desenvolvimento e um conjunto de boas diretrizes. Essas necessidades são preenchidas pelo campo da engenharia *Web* (Ginige e Murugesan, 2001b).

A engenharia *Web* visa gerenciar a diversidade e a complexidade do desenvolvimento de aplicações *Web*, evitando possíveis falhas (Ginige e Murugesan, 2001a). Utiliza princípios científicos, gerenciais e de engenharia, além de abordagens sistemáticas, para o projeto, a construção e a manutenção de aplicações *Web* de alta qualidade. Por meio da engenharia *Web*, há um maior controle sobre o desenvolvimento, minimizando riscos e aumentando a manutenibilidade e qualidade das aplicações *Web* (Ginige e Murugesan, 2001b).

Segundo Murugesan et al. (1999) *apud* Domingues (2005), a engenharia *Web* absorve os mais difundidos princípios e práticas da engenharia de software, além de considerar elementos específicos de aplicações *Web*.

Com relaçao aos métodos de desenvolvimento para *Web*, podem ser encontrados atualmente vários estudos como: HDM (*Hypermedia Design Model*) (Garzotto et al., 1991, 1993), RMM (*Relationship Management Methodology*) (Isakowitz et al., 1995), OOHDM (*Object-Oriented Hypermedia Design Method*) (Rossi, 1996), HMBS/M (*Hypertext Model Based on Statecharts/Method*) (Carvalho, 1998), W2000 (*UML Extension for Modeling Web Application*) (Baresi et al., 2001), UWE (*UML-based Web Engineering*) (Koch, 2000), WebML (*Web Modeling Language*) (Ceri et al., 2000), WAE (*Web Application Extension*) (Conallen, 2002), OO-H (*Object-Oriented Hypermedia*) (Gómez et al., 2001), OOWS (*Object-Oriented Web Solution*) (Fons et al., 2003), SWM (*Simple Web Method*) (Griffiths et al., 2002) e ECO (*Ecosystem of Agile Software Development*) (Figueiredo, 2005).

No contexto do trabalho de Bianchini (2007), esses métodos foram avaliados e um arcabouço foi criado para auxiliar projetistas a identificarem qual o mais apropriado no

desenvolvimento de aplicações para a *Web*. Seguindo os critérios do arcabouço, o processo e método de desenvolvimento propostos por Conallen (2002) foram considerados os mais adequados para auxiliar na construção do portal do estudo de caso realizado por Bianchini (2007).

Assim como portais, o ambiente Peônia, desenvolvido neste projeto de mestrado, utiliza muita interação com o usuário e banco de dados. Por causa dessas características, o processo e método de desenvolvimento propostos por Conallen (2002), explicados na Seção 2.3.1, podem ser empregados no desenvolvimento do ambiente Peônia.

2.3.1 Processo de Desenvolvimento Proposto Por Conallen e o WAE (Web Application Extension)

O WAE (*Web Application Extension*) (Conallen, 2002) provê uma extensão da notação UML baseada em estereótipos para a modelagem de elementos específicos de aplicações *Web*. São fornecidos estereótipos que permitem representar diferentes tipos de páginas, elementos de interface e relacionamentos entre componentes (Bianchini, 2007). Diferente da maioria dos métodos de projeto hipermídia, no WAE não são tratados separadamente os aspectos mais específicos das aplicações *Web*, ou seja, conteúdo, navegação e apresentação (Conallen, 2002) *apud* (Domingues, 2005).

Conallen (2002) descreve um processo de desenvolvimento empregando o WAE para o projeto de aplicações *Web*. Esse processo de desenvolvimento é orientado a caso de uso, centralizado na arquitetura, iterativo e incremental baseado no RUP (*Rational Unified Process*) (Kruchten, 2000) e ICONIX *Unified Process* (Rosenberg e Scott, 1999). A intenção não é utilizar o processo diretamente, mas adaptado à empresa ou projeto onde será empregado, levando em consideração a composição da empresa e da organização; a natureza da aplicação; o nível de habilidade da equipe de desenvolvimento; e as prioridades relativas do conjunto de recursos, tempo de produção, total de defeitos aceitáveis entre outros.

O processo proposto Conallen (2002) começa com uma *Análise do Negócio e Problema Percebido*, com o *Desenvolvimento do Modelo de Domínio*. Sabendo do estado do negócio e domínio, é necessário *Analisar o Problema Compreendido* enfocando na parte que realmente será trabalhada e *Desenvolver o Documento de Visão* com o escopo e a finalidade de todo o projeto de software, além de fornecer os recursos principais e critérios de aceitação para o sistema.

Em seguida, é *Desenvolvido o Plano do Projeto* onde as atividades, esforço necessário, principais eventos e documentos apropriados são definidos. É no plano de projeto que um esboço do plano de iteração é apresentado, ou seja, quais os artefatos que devem ser

produzidos e o critério para avaliar o sucesso de cada um. A *Gerência de Versões dos Artefatos* deve ser realizada durante a execução de todas as atividades (Conallen, 2002).

Na Figura 2.1, é apresentado o diagrama de atividades que representa o processo de desenvolvimento de software proposto por Conallen (2002). A atividade apresentada como "Reiterar" na Figura 2.1 representa a execução das atividades de desenvolvimento do caso de uso "Iteração de Software", listadas na Figura 2.2.

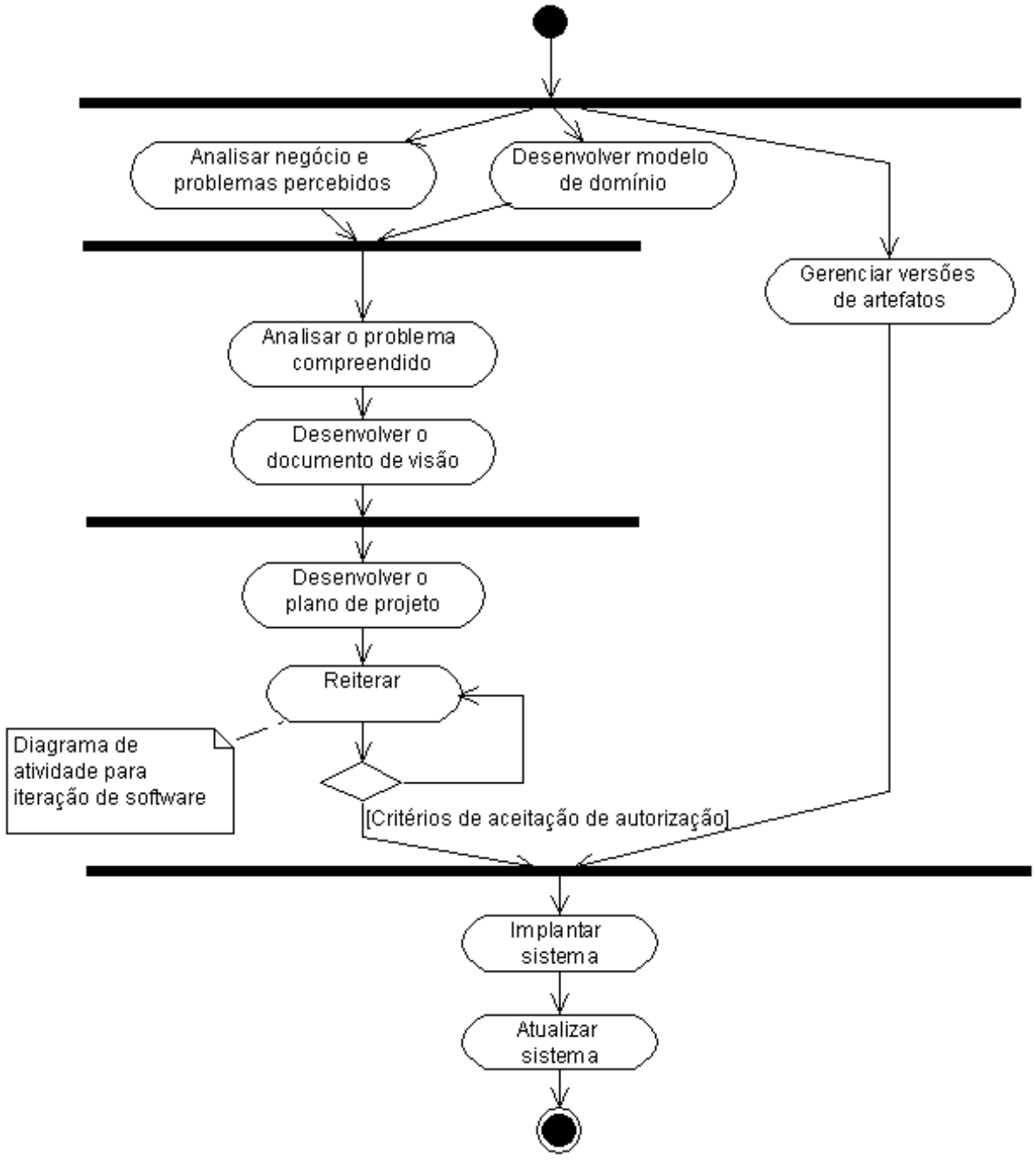

Figura 2.1: Diagrama de atividades do caso de uso *Desenvolver Software* (Conallen, 2002).

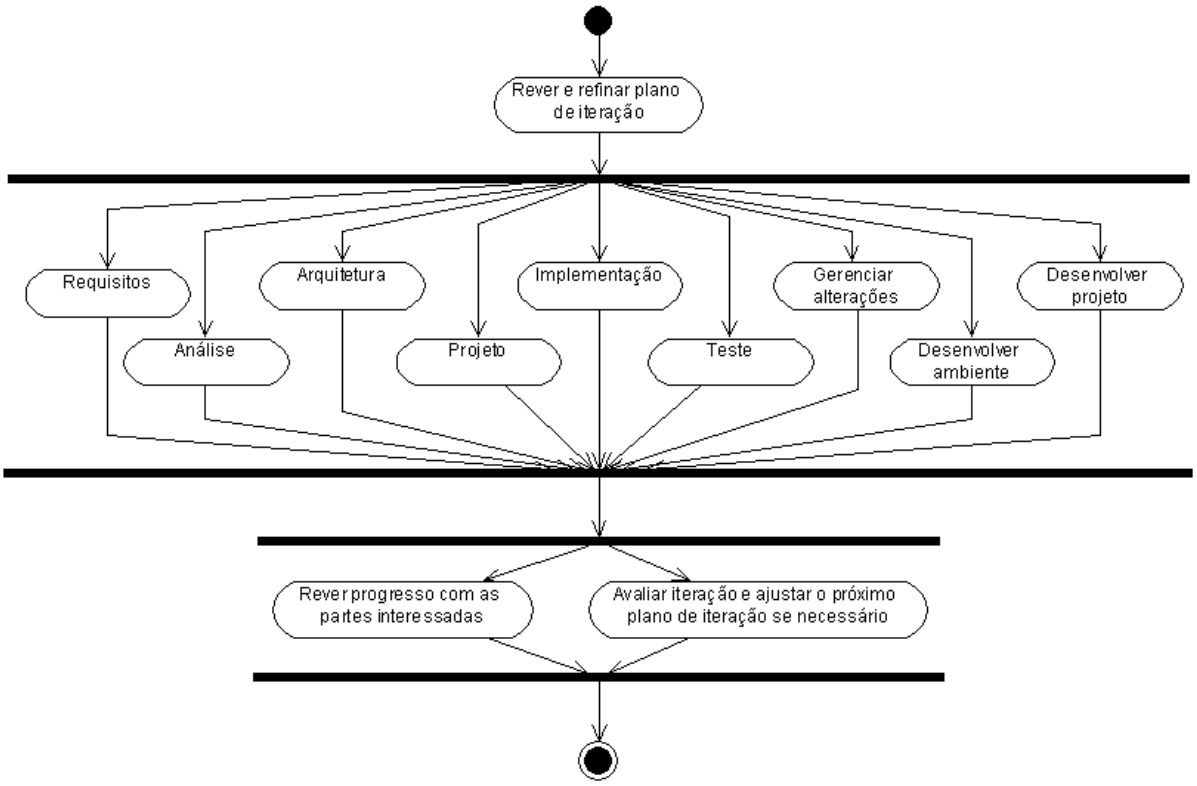

Figura 2.2: Diagrama de atividades do caso de uso *Iteração de Software* (Conallen, 2002).

Com relação à modelagem, o processo proposto por Conallen (2002) recomenda a geração dos modelos de domínio, casos de uso, análise/projeto, implementação, processo, segurança e experiência de usuário ou interface com o usuário.

Os artefatos do processo proposto por Conallen (2002) são divididos em oito etapas ou conjuntos: de gerenciamento de projeto, domínio, requisitos, análise, projeto, implementação, teste e implantação.

Durante o *Conjunto de Gerenciamento de Projeto* são desenvolvidos os planos de projeto, de iteração e de gerência de alteração. Nos dois primeiros planos citados é definido o cronograma do desenvolvimento, incluindo a seleção de pessoal, as responsabilidades e principais eventos (Conallen, 2002).

No *Conjunto de Domínio* são criados o modelo de domínio e o glossário. No modelo de domínio contém a essência do negócio e do domínio do sistema. No *Conjunto de Requisitos* são criados o documento de visão, documento de diretrizes da experiência do usuário e o modelo de casos de uso (Conallen, 2002).

Durante o *Conjunto de Análise* é gerado o documento de arquitetura de software e realizada a divisão do projeto em subsistemas, permitindo o desenvolvimento assíncrono. Também ocorre a análise dos casos de uso para a criação dos modelos conceitual e de experiência do usuário (modelo UX) e são definidos as classes de análise, os seus relaci-

onamentos estruturais e as suas propriedades, sendo utilizados diagramas de seqüência, atividade e estado (Conallen, 2002).

No *Conjunto de Projeto* os artefatos criados na análise são adicionados a arquitetura para tornar o modelo de análise realizável em software. Nessa etapa as classes são definidas em nível de projeto. No *Conjunto de Implementação* os artefatos do projeto em conjunto com ferramentas de desenvolvimento são utilizados para a implementação dos componentes do software (Conallen, 2002).

O *Conjunto de Teste* concentra-se na avaliação dos artefatos executáveis do sistema seguindo um plano de teste, composto por procedimentos e *scripts* de teste, e que produzirão os resultados a serem analisados. Por fim, o *Conjunto de Implantação* é composto por instruções de instalação e os arquivos do sistema produzido (Conallen, 2002).

Assim, criando, evoluindo e transformando os conjuntos de artefatos recomendados pelo processo descrito por Conallen (2002) chega-se ao produto final desejado.

2.4 Padrões de Software

O conceito de padrões foi introduzido por Christopher Alexander na área da arquitetura no final da década de 70. Alexander publicou os livros *"A Pattern Language"* (Alexander, 1977) e *"A Timeless Way of Building"* (Alexander, 1979) introduzindo as primeiras definições de *Padrões* e de *Linguagem de Padrões*, além de descrever o seu método de documentação de padrões (Braga et al., 2001).

Ao se depararem com um problema, especialistas costumam recordar dificuldades similares, reutilizando soluções antigas, pensando em pares "problema/solução" (Buschmann, F.; et al., 1997).

Um padrão é um conjunto nomeado de informações instrutivas que capturam a estrutura essencial e o raciocínio de uma família de soluções bem sucedidas e comprovadas de um problema recorrente que ocorre dentro de um contexto e de um sistema de forças (Appleton, 2000).

Padrões de software documentam um problema recorrente, a solução para esse problema e o contexto em que deve ser aplicada essa solução (Santos, 2004). Além de identificar a solução para um problema, padrões devem explicar porque a solução é necessária (Appleton, 2000).

Muitas soluções, algoritmos, boas práticas, regras ou heurísticas têm sido nomeadas de padrões, no entanto, mesmo que tivessem todos os componentes requisitados por um padrão, não poderiam ser assim considerados enquanto não fosse comprovado que são um fenômeno recorrente. Geralmente, para comprovar essa recorrência, é utilizada a *Regra de Três* (*Rule of Three*), que exige pelo menos a aplicação em três sistemas. Se não for

comprovada a recorrência, a solução pode ser denominada *Proto-Padrão* (*Proto-Pattern*) (Appleton, 2000).

A utilização de padrões tem por objetivos: criar uma literatura que auxilie no emprego de boas práticas no desenvolvimento de sistemas (Santos, 2004), prover um vocabulário comum, criar abstrações num nível superior ao de classes, garantir uniformidade na estrutura do software (Gall et al., 1996) *apud* (Braga et al., 2001) e atuar como bloco construtivos para a construção de projetos mais complexos (Gamma et al., 1995) *apud* (Braga et al., 2001).

Padrões de software podem trazer benefícios, tanto às áreas ligadas diretamente ao projeto e a implementação, quanto às áreas de outras disciplinas que fornecem suporte ao desenvolvimento de sistemas (Santos, 2004). No entanto, também podem trazer desvantangens, como por exemplo, a perda de eficiência devido à adição de classes ou de novas camadas de aplicação e a diminuição da legibilidade e da manutenibilidade por causa do aumento da complexidade do código com a adição de classes, mensagens, linhas de código e níveis hierárquicos de classes.

Padrões não são conceitos isolados, possuindo relacionamentos como, por exemplo, ser combinações, variações, conterem ou estarem contidos em outros padrões. Assim, padrões e seus variantes possuem soluções para problemas similares, podendo ser agrupados segundo algum critério que facilite a busca e utilização. Esse agrupamento pode ser realizado por meio de coleções, catálogos, sistemas ou linguagens de padrões (Braga et al., 2001).

Apesar de alguns componentes serem essenciais, não existe um consenso quanto ao formato de documentação de padrões (Braga et al., 2001), dificultando a localização e utilização. Assim, na Seção 2.4.1, são listados os componentes essenciais para descrever um padrão, além de outros componentes que devem ser claramente descritos para auxiliar a compreensão dos leitores.

Para facilitar a localização, padrões de software podem ser divididos em categorias. Na Seção 2.4.2 são mencionadas algumas classificações encontradas dessa tecnologia de reúso.

2.4.1 Componentes de Padrões

Descrições de padrões de software costumam ser apresentadas divididas em seções, ou componentes, formando o *template* ou formato de um padrão (Santos, 2004). Padrões têm sido documentados em diferentes formatos (Braga et al., 2001), no entanto, algumas informações são consideradas essenciais para diferenciar um padrão de uma descrição qualquer de um par problema/solução, permitindo a sua busca e utilização correta (Meszaros e Doble, 1996).

Segundo o padrão *"Presença de Elemento Obrigatório"* (*Mandatory Element Present*), da linguagem de padrões de Meszaros e Doble (1996), os seguintes elementos são obrigatórios em um padrão:

- *Nome do Padrão*: Um nome que identifique o par problema/solução, permitindo que seja referenciado.
- *Problema*: Descrição do problema que deve ser solucionado pelo padrão.
- *Solução*: A solução proposta para o problema. Pode conter mais que uma solução para um mesmo problema, sendo que a melhor a ser utilizada depende do contexto do problema e das forças.
- *Contexto*: As circunstâncias, ou seja, as pré-condições, em que o problema deve ser resolvido, definindo algumas restrições para o emprego da solução. Costuma ser definido em termos de uma situação e pode ocorrer de ser especificado um padrão que deve ter sido empregado anteriormente. É o contexto que determina a importância das forças.
- *Força*: São considerações, positivas ou negativas, que devem ser analisadas antes de tomar a decisão de utilizar uma solução. A importância da força depende do contexto do problema.

Os elementos obrigatórios podem ser apresentados em ordens diferentes, ou com nomes diferentes. Esses elementos indicam o caminho para a escolha correta da solução, sendo a informação mínima para comunicar efetivamente a idéia do padrão (Meszaros e Doble, 1996).

Segundo Appleton (2000), além dos componentes mencionados anteriormente, as seguintes informações devem ser claramente identificáveis em um padrão:

- *Exemplo*: Mostra empregos do padrão que ilustrem o contexto inicial, a maneira como o padrão foi empregado, a transformação no contexto após a utilização do padrão e o contexto final. Os exemplos facilitam a compreensão da utilização e aplicabilidade de um padrão.
- *Contexto Resultante*: Define o estado do sistema após a utilização do padrão, incluindo as conseqüências e outros problemas que podem ser gerados com o novo contexto. O contexto resultante descreve as pós-condições e efeitos colaterais do padrão e pode também ser denominado como *Resolução de Forças*, pois descreve quais forças foram solucionadas, quais não foram resolvidas e quais padrões podem ser aplicados.

- *Raciocínio* (*Rationale*): Descreve uma explicação justificando os passos e regras do padrão, além de definir o padrão como um todo, informando como e porque ele soluciona as forças. O Raciocínio explica como realmente o padrão trabalha, porque ele funciona e porque ele é bom.

- *Padrões Relacionados*: Informam os relacionamentos dinâmicos e estáticos entre os padrões de uma mesma linguagem de padrões ou entre sistemas. Geralmente, padrões relacionados compartilham das mesmas forças e podem ter um contexto inicial ou resultante compatível com o contexto de um outro padrão. Assim, existem três tipos de relacionamentos: os Padrões Predecessores, que a aplicação conduz a esse padrão; Padrões Sucessores, que são aplicados após esse padrão; e Padrões Alternativos, que descrevem a solução para um problema, mas com diferentes forças e dependências.

- *Usos Conhecidos*: Descreve utilizações conhecidas dos padrões em sistemas existentes. Os usos conhecidos auxiliam a provar que o padrão realmente é uma solução comprovada para um problema recorrente.

Normalmente, bons padrões também possuem um *Resumo* fornecendo uma visão breve e geral do problema que ele resolve. Dessa forma, o público alvo identifica com maior facilidade o quê procura e o quê espera do padrão (Appleton, 2000).

2.4.2 Classificações de Padrões

Padrões de software possuem diversos níveis de abstração. Para facilitar a sua recuperação e utilização, padrões podem ser divididos em categorias (Braga et al., 2001) de acordo com critérios de classificação. De acordo com Santos (2004), o termo *"classificacao"* é utilizado para definir o processo de classificar ou distribuir por classes, e o termo *"categoria"* é empregado para referenciar uma classe particular de uma classificação. O conjunto de categorias define um critério de classificação.

Classificação de padrões também agiliza o seu aprendizado, além de direcionar esforços na descoberta de novos padrões (Gamma et al., 1995). Alguns critérios de classificação são (Santos, 2004):

- Classificação Segundo o Estágio de Desenvolvimento de Software;

- Classificação Segundo o Domínio da Aplicação;

- Classificação Segundo a Camada de Aplicação do Padrão;

- Classificação GoF (*Gang of Four*) (Gamma et al., 1995);

- Classificação POSA (*Pattern-Oriented Software Architecture*) (Buschmann et al., 1996).

Classificações não são rigorosas e alguns padrões podem pertencer a mais de uma categoria (Braga et al., 2001). O *Almanaque de Padrões 2000* (Rising, 2000) indexa os padrões por categorias, podendo ser classificados de acordo com o estágio de desenvolvimento, o domínio da aplicação ou as categorias utilizadas pela GoF.

No contexto deste trabalho de mestrado, duas classificações podem ser destacadas: a Classificação Segundo o Estágio de Desenvolvimento de Software, detalhada na Seção 2.4.2.1, e a Classificação Segundo o Domínio da Aplicação, explicada na Seção 2.4.2.2.

2.4.2.1 Classificação Segundo o Estágio de Desenvolvimento de Software

A Engenharia de Software divide o desenvolvimento de software em cinco estágios (Corriveau et al., 1997; Jacobson, 1992) *apud* (Andrade, 2001): captura de requisitos, análise, projeto, implementação e teste. Após cada estágio, uma visão mais detalhada do sistema é produzida.

Assim, segundo Andrade (2001), padrões de software podem ser classificados de acordo com o estágio de desenvolvimento em que são aplicados: *Padrões de Requisitos*, *Padrões de Análise*, *Padrões de Projeto* e *Padrões de Implementação*, ou *Idiomas*. De acordo com Santos (2004), pode ser considerada uma quinta categoria, a de *Padrões de Testes*.

Na Figura 2.3 são ilustradas as categorias da classificação por estágios de desenvolvimento de software. Destaca-se que durante o estágio de projeto, além dos Padrões de Projeto, existem os *Padrões Arquiteturais* e *Meta-Padrões* (Andrade, 2001; Santos, 2004).

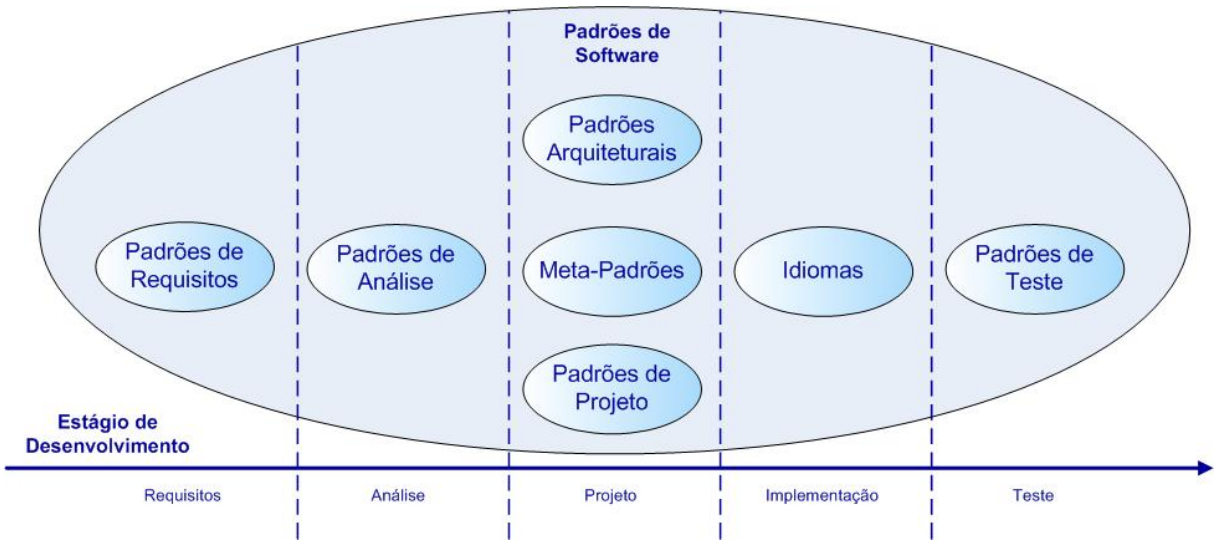

Figura 2.3: Classificação Segundo o Estágio de Desenvolvimento de Software. (Andrade, 2001; Santos, 2004)

Os *Padrões de Requisitos* documentam em um alto nível de abstração as necessidades dos usuários e o comportamento geral do sistema (Andrade, 2001).

Padrões de Análise descrevem o conhecimento de um domínio. Essa categoria de padrão tem por objetivo desenvolver, antes da etapa de projeto, um modelo de análise que descreva a estrutura conceitual do processo de negócio de um software. Assim, a maior preocupação dos Padrões de Análise é o modelo conceitual e a maneira como esse modelo afeta a flexibilidade e reusabilidade do sistema (Andrade, 2001).

Os *Padrões de Projeto* são um método para refinar componentes de sistemas de softwares ou os relacionamentos entre eles. Esses padrões são descrições de classes e de objetos comunicantes e customizáveis que solucionam um problema de projeto em um contexto particular. Uma classe representa como será a implementação de um objeto, definindo dados internos e operações que ele pode desempenhar. Objeto contém dados e procedimentos que operam sobre dados (Andrade, 2001).

Os *Padrões Arquiteturais* são relacionados a organização estrutural de uma aplicação, incluindo o projeto detalhado de componentes e a colaboração e a comunicação entre eles. Geralmente, Padrões Arquiteturais são utilizados nas fases iniciais do projeto (Andrade, 2001).

Meta-Padrões são conjuntos de Padrões de Projeto que auxiliam a construção de *frameworks* bem projetados e independentes do domínio (Pree, 1995) *apud* (Andrade, 2001). Um *framework* constitui um bloco semi-construído e pronto para ser utilizado (Andrade, 2001).

Padrões de Implementação ou *Idiomas* são guias sobre como Padrões de Projeto devem ser implementados em uma determinada linguagem (Andrade, 2001).

Segundo Santos (2004), Padrões de Projeto e de Implementação foram os primeiros padrões de software a serem publicados. Depois surgiram os Padrões de Análise, de Requisitos e de Testes.

2.4.2.2 Classificação Segundo o Domínio da Aplicação

Desenvolvedores com pouca experiência em padrões costumam buscá-los pela natureza da aplicação. Assim, a classificação por domínio de aplicação é um critério muito utilizado e é comum encontrar na literatura categorias de padrões como (Santos, 2004): *Web*, *GUI* (*Graphical User Interface*), *XML*, *Segurança de Sistemas*, *Redes*, *Banco de Dados* e *Sistemas Distribuídos*.

Alguns *sites* disponibilizam padrões de acordo com a área específica de pesquisa (Santos, 2004), por exemplo, o *SecurityPatterns.org*[1], que possui padrões de diversos autores

[1] `http://www.securitypatterns.org`

sobre segurança de sistemas, o *Amsterdam Patterns Collection*[2], que possui uma coleção de padrões de projeto para *Web*, *GUI* e *MobileUI* (*Mobile User Interface*) e o HPR (*Hypermedia Design Patterns Repository*) (Bolchini et al., 2002), que é um repositório para padrões hipermídia, melhor detalhado na Seção 3.4.2.

Também existem as categorias que fornecem apoio ao desenvolvimento de sistemas, por exemplo, *Gerenciamento de Configuração*, *Organizacional*, *Processo*, *Modelagem de Sistemas* e *Treinamento* (Ambler, 2006; Coplien, 1995; Rising, 2000) *apud* (Santos, 2004).

2.5 Teste de Software

Um software deve ser previsível e consistente sem oferecer surpresa aos seus usuários. (Myers, 2004). Mesmo que o processo de desenvolvimento de software utilize uma série de técnicas, métodos e ferramentas, erros no produto ainda podem ocorrer. Assim, um conjunto de atividades, denominadas de Garantia de Qualidade de Software, são introduzidas durante todo o processo de desenvolvimento de software, destacando-se as atividades de VV&T, que visam minimizar riscos e erros associados. O teste é a atividade mais utilizada e constitui um dos elementos para fornecer evidências da confiabilidade do software (Maldonado, 1991).

Teste de Software é o processo, ou conjunto de processos, utilizado para garantir que um código computacional realiza o que é esperado e não efetua o que não é do seu propósito. Teste é o processo de executar um programa com o objetivo de encontrar erros (Myers, 2004).

O software deve ser testado para que sejam encontrados e removidos os erros. Para localizar a maior quantidade possível de erros, testes devem ser conduzidos de maneira sistemática e casos de testes devem ser projetados utilizando técnicas disciplinadas (Pressman, 2002).

Segundo Myers (2004), é impraticável e geralmente impossível encontrar todos os erros de um programa. Assim, uma estratégia deve ser estabelecida antes de iniciar os testes (Myers, 2004), para que seja coberta adequadamente a lógica do programa e para garantir que as condições do projeto tenham sido cumpridas (Pressman, 2002).

Os critérios de testes são estabelecidos, em geral, de acordo com as seguintes técnicas: *Teste Funcional*, *Teste Estrutural*, *Teste Baseado em Erros* e *Teste Baseado em Máquinas de Estados Finitos*.

Dessas técnicas destaca-se o *Teste Funcional*, cujos critérios são descritos na Seção 2.5.1. É a partir de critérios de teste funcional que Cagnin et al. (2005) propõem uma

[2] `http://www.welie.com`

estratégia para validar a solução de padrões de software, cujos detalhes são apresentados na Seção 2.5.2.

2.5.1 Teste Funcional

O *Teste Funcional*, ou *Teste Caixa-Preta*, considera o programa como uma caixa-preta, preocupando-se com o aspecto fundamental do sistema, sem preocupar-se muito com o comportamento interno e a estrutura lógica do software (Myers, 2004). Os testes são conduzidos na interface do software, focalizando seus requisitos funcionais.

Testes funcionais são utilizados para comprovar que as funções do software estão operacionais, que a entrada é aceita e a saída produzida é correta e que as informações externas são mantidas. Assim, o teste funcional tenta encontrar funções incorretas ou omitidas; erros de interface; erros de acesso à base de dados ou de estrutura de dados; erros de comportamento ou de desempenho; erros de iniciação e término (Pressman, 2002).

Essa técnica deriva os casos de teste da especificação do software (Myers, 2004), assim, é essencial que essa especificação esteja bem elaborada (Maldonado et al., 2004). Os critérios de teste funcional mais conhecidos são (Myers, 2004; Pressman, 2002):

- *Particionamento de Equivalência*: divide o domínio de entrada de um programa em classes de equivalência, que representam um conjunto de estados válidos ou inválidos. Assume-se que o teste de um valor de uma classe de equivalência é representativo ao teste de qualquer outro valor dessa mesma classe, ou seja, se um caso de teste de uma classe de equivalência apresentar um erro, todos os outros casos de teste dessa classe também apresentarão erros e, se não detectar erro, todos os outros também não detectarão. Casos de testes são selecionados de tal forma que seja exercitada a maior quantidade de atributos de uma classe. Esses casos de teste devem ser gerados tanto para classes válidas quanto inválidas.

- *Análise do Valor Limite*: seleciona casos de teste que exercitam os valores limites do domínio de entrada. É um critério que complementa o Particionamento de Equivalência, pois ao invés de escolher qualquer valor da classe de equivalência, realiza-se a seleção de elementos da fronteira classe, onde se costuma encontrar mais erros. São derivados casos de testes tanto das condições de entrada quanto do espaço resultante.

- *Grafo de Causa-Efeito*: realiza a combinação das condições de entradas, o que não é explorado pelos dois critérios citados anteriormente. A partir da especificação, é realizado um levantamento das causas e efeitos. As causas são as condições de entrada e os efeitos são as condições de saída ou transformações do sistema. Depois é construído um grafo booleano ligando as causas e os efeitos. Esse grafo é convertido para uma tabela de decisão, de onde são gerados os casos de teste.

Outro critério da técnica funcional é o *Teste Funcional Sistemático* que propõe a combinação dos critérios Particionamento de Equivalência e Análise do Valor Limite (Linkman et al., 2003).

Como as especificações de software são, em geral, descritivas e informais, os critérios funcionais possuem problemas com a criação dos requisitos de teste, que se tornam, também, imprecisos e informais. Assim, há uma dificuldade para realizar a automatização da aplicação desses critérios, no entanto, são aplicáveis em todas as fases do teste de software, pois necessitam apenas de dados de entrada, da função a ser testada e da saída produzida.

2.5.2 Teste Para Padrões de Software

Os padrões de teste, encontrados na literatura, provêem diretrizes e procedimentos que auxiliam a avaliação da qualidade do produto, mas não capturam a solução dos especialistas, nem os aspectos específicos da validação de uma aplicação. Uma boa solução deve ter a sua validação capturada do padrão, permitindo o reúso de informações de VV&T juntamente com o reúso da solução (Cagnin et al., 2005).

Tabela 2.1: Passos da estratégia para alocar recursos de teste a padrões de software. (Cagnin et al., 2005)

- **Dado um Padrão de Software**
  - **Defina** tipos de requisitos
  - **Selecione** critérios de teste existentes
  - **Para** cada tipo de requisito definido e comum a maioria dos elementos do padrão
    - **Crie** requisitos de teste baseados no critério de teste selecionado
  - **Para** cada elemento do padrão
    - **Para** cada tipo de requisito de teste definido e específico ao elemento do padrão
      - **Crie** requisitos de teste baseados no critério de teste selecionado
- **Classifique e documente** cada requisito de teste criado
- **Derive** casos de teste
- **Mapeie** casos de teste comuns
- **Torne** os recursos de teste disponíveis

Assim, Cagnin et al. (2005) propõe uma estratégia de alocar recursos de testes a padrões de software, permitindo que seja realizado o reúso da solução e dos recursos de

testes correspondentes, para testar a aplicação. Essa alocação é motivada pela importância de facilitar as atividades de VV&T em softwares baseados em padrões. Na Tabela 2.1 são descritos os passos da estratégia, proposta por Cagnin et al. (2005), para alocar padrões de software a recursos de teste. Mais detalhes sobre a abordagem proposta por Cagnin et al. (2005) são dadas na Seção 5.3.

2.6 Considerações Finais

Neste capítulo foram apresentados conceitos a respeito de aplicações *Web* e são listadas algumas classificações encontradas na literatura, além de resumida a definição de arquitetura e comentada a importância de um projeto arquitetural.

No contexto *Web*, pouca atenção é dada a páginas estáticas, sendo a maior parte dos estudos voltados a aplicações que envolvam algum tipo de computação de dados (Domingues, 2005). Portanto, com relação à classificação, assim como no trabalho de doutorado de Domingues (2005), neste trabalho é adotada a definição que descreve uma aplicação *Web* como uma aplicação que apóia a realização de transações de negócio.

Como a quantidade e a complexidade de aplicações *Web* têm aumentado, o desenvolvimento *Web* tem se tornado uma tarefa difícil. A elaboração do projeto arquitetural pode auxiliar engenheiros de software a analisar a viabilidade e avaliar a satisfação quanto aos requisitos de uma aplicação. Com o crescimento das exigências por aplicações *Web* complexas e de qualidade, têm aumentado a necessidade de processos, métodos e ferramentas que apóiem o desenvolvimento desse tipo de aplicações.

Também foram apresentadas as definições de engenharia *Web* e de processos e métodos de desenvolvimento e a motivação de empregá-los. Além disso, foram listados alguns dos métodos de desenvolvimento *Web* encontrados na literatura atual. Como o ambiente Peônia é uma aplicação *Web*, a utilização de métodos de desenvolvimento que tratem das características específicas desse tipo de aplicação foi necessária.

A comparação entre métodos de desenvolvimento não é trivial, pois os objetivos de cada método podem ser relativamente diferentes. Domingues (2005) cita três estudos comparativos utilizando como base de comparação os atributos gerais, as etapas do processo de desenvolvimento e os aspectos de informação e funcional das aplicações *Web*. No Apêndice A são apresentadas duas tabelas comparativas considerando as diferenças de atributos gerais e etapas do processo de desenvolvimento entre os métodos *Web* citados neste capítulo.

Os métodos apresentados neste capítulo foram avaliados no contexto do trabalho de mestrado de Bianchini (2007) e foi criado um arcabouço para auxiliar usuários na escolha do mais apropriado para guiar a criação de aplicações.

Neste capítulo também foram descritos padrões de software, as vantagens e as desvantagens de empregar padrões e mencionadas as maneiras como eles podem ser agrupados. Foram listadas algumas classificações utilizadas para melhorar a compreensão de padrões e apresentados os seus componentes essenciais, necessários para facilitar o entendimento dos usuários.

Como existe uma grande quantidade de padrões na literatura, é preciso tomar cuidado com a sua qualidade. Uma das maneiras de selecionar padrões é avaliar se contém os componentes essenciais e se foi utilizado em pelo menos três aplicações. Se o padrão possuir uma seção especial descrevendo como proceder para validar, tanto o padrão, como as aplicações criadas a partir dele, usuários podem avaliar melhor a qualidade da sua solução.

Por fim, foram listadas algumas técnicas de teste de software, explicados os critérios da técnica de teste funcional e apresentada a abordagem de associação de requisitos de teste a padrões de software, proposta por Cagnin et al. (2005). Para que seja obtido um conjunto de casos de teste adequado para avaliar uma aplicação, técnicas e critérios devem ser combinados para montar uma boa abordagem.

Ferramentas podem ser utilizadas para automatizar a atividade de teste, assim como para apoiar outras atividades do ciclo de vida de um software. *Frameworks* podem ser empregados para apoiar a implementação de aplicações. Dentro do contexto deste trabalho, no próximo capítulo são apresentadas ferramentas de apoio ao desenvolvimento.

Capítulo

3

Ferramentas de Apoio ao Desenvolvimento

3.1 Considerações Iniciais

À medida que a demanda e complexidade de softwares crescem, ambientes e ferramentas de engenharia de software tornam-se importantes facilitadores (Harrison et al., 2000).

Como mencionado no Capítulo 2, ferramentas são utilizadas para fornecer apoio automatizado ou semi-automatizado no emprego de métodos de desenvolvimento *Web*. Além de auxiliar no emprego de métodos, ferramentas podem também ser utilizadas para auxiliar engenheiros de software a empregarem padrões de projeto na criação de suas aplicações (Marinho et al., 2003) e para apoiar as atividades de VV&T. Outro recurso são os ambientes, que auxiliam a gerência de processos de desenvolvimento de software (Harrison et al., 2000).

Ferramentas e ambientes minimizam a complexidade no desenvolvimento, evitam erros de usuários inexperientes ao utilizarem processos, métodos, padrões, ou teste, além de prevenir contra a omissão na verificação, validação e teste de aplicações. Alguns trabalhos sobre ambientes e ferramentas podem ser citados, sendo que cada um deles fornece auxílio isolado nas quatro áreas mencionadas.

Na Seção 3.2 são dados exemplos de ambientes e ferramentas que apóiam a utilização de processos e métodos de desenvolvimento. Na Seção 3.3 são listadas ferramentas de

automatização de teste. Na Seção 3.4 são apresentadas algumas ferramentas que apóiam a utilização de padrões de software. Por fim, na Seção 3.5 são apresentadas as considerações finais sobre este capítulo.

3.2 Ferramentas de Apoio a Processos e Métodos de Desenvolvimento

As primeiras ferramentas de apoio ao processo de software auxiliavam a execução de atividades de maneira isolada, não levando em consideração a integração de ferramentas e a execução coordenada de processo. Os ambientes de desenvolvimento de software surgiram para apoiar as atividades de engenharia de software de maneira integrada, durante todo o ciclo de vida. Os ambientes de desenvolvimento de software centrados em processo são construídos tomando por base a representação explicita dos processos de software, seus produtos e sua interação, integrando ferramentas de apoio ao desenvolvimento de software com ferramentas de modelagem e execução de processos (Bertollo et al., 2006) *apud* (Harrison et al., 2000).

Como exemplos de ambientes que apóiam e acompanham o emprego de processos de desenvolvimento de software podem ser citados:

- WebAPSEE (*Web Process-Centered Software Engineering Environments*) (Lima et al., 2006): ambiente que auxilia a modelagem, execução e manutenção de processos de desenvolvimento de software. Outras funcionalidades estão sendo implementadas para apoiar a simulação, reutilização e instanciação de processos.

- ODE (*Ontology-based software Development Environment*) (Bertollo et al., 2006): é um ambiente de desenvolvimento de software centrado em processo, baseado em ontologias (Bertollo et al., 2006) e utilizado para a definição de processos ODE (Falbo et al., 2004).

Ferramentas que apóiam o desenvolvimento utilizando métodos de desenvolvimento apresentam divergências em muitos aspectos, por causa dos diferentes conceitos, categorizações e processo de desenvolvimento que seguem. Com o crescimento da utilização da *Web*, diversos métodos e ferramentas que auxiliam o desenvolvimento nessa área tem surgido. Exemplos dessas ferramentas que apóiam o emprego de métodos de desenvolvimento *Web* são:

- OOHDM-Web (Schwabe e Pontes, 1998; Schwabe et al., 1999): ambiente que permite o desenvolvimento, orientado por *templates*, de aplicações *Web* projetadas utilizando OOHDM (Rossi, 1996). Esse ambiente também permite a implementação

de aplicações hipermídia no formato de *scripts* CGI, para a geração dinâmica de páginas, cujo conteúdo é preenchido com informações de base de dados e integrado a *templates* pré-definidos.

- HySCharts (*Hypermedia System based on Statecharts*) (Turine et al., 1998, 1999): ambiente que permite a criação, interpretação e execução de aplicações *Web*, seguindo HMBS. Oferece apoio a fase de modelagem navegacional de instâncias, auxiliando a especificar a estrutura organizacional e navegacional da aplicação *Web* (Brito, 2003). Permite também realizar a simulação interativa dessa estrutura desenvolvida (Domingues, 2005).

- WebSCharts (Brito, 2003): ferramenta CASE (*Computer Aided Software Engineering*) que permite a geração de aplicações *Web* empregando o método HMBS/M Estendido (Brito, 2003). Essa ferramenta pode ser classificada como um gerador de aplicação.

- ArgoUWE (Knapp et al., 2003, 2004a, 2005, 2004b): ferramenta CASE que fornece apoio a etapa de projeto do método de desenvolvimento UWE (Koch, 2000). A ArgoUWE é parte da ferramenta OpenUWE (Koch, 2004), que é um ambiente orientado a modelo para a geração de aplicações *Web* (Knapp et al., 2003).

- WebRatio (Web Models, 2006): ferramenta CASE que apóia o desenvolvimento rápido de aplicações *Web* utilizando o método WebML (Ceri et al., 2000). Permite a construção de um modelo visual para a especificação do conteúdo, navegação e lógica de negócio de uma aplicação para a *Web*. A WebRatio integra as funcionalidades de geração automática de código com o processo de análise de especificação (Web Models, 2006).

3.3 Ferramentas de Apoio ao Teste de Software

Sem uma ferramenta automatizada de teste a aplicação de um critério é uma atividade propensa a erros. Ferramentas automatizadas, além de contribuirem para a produção de software de alta qualidade, auxiliam pesquisadores e alunos a obterem conceitos básicos e experiência na comparação, seleção e estabelecimento de estratégias de teste (Maldonado et al., 2004).

Para auxiliar na automatização de teste são encontradas na literatura ferramentas como: a Proteste (Price e Zorzo, 1990), que fornece apoio ao teste estrutural de programas em Pascal; a Atac (*Automatic Test Analysis for C*) (Horgan e Mathur, 1992), que apóia a aplicação dos critérios estruturais de fluxo de controle e de dados em programas C

e C++; a JaBUTi (*Java Bytecode Understanding and Testing*) (Vincenzi et al., 2003), utilizada para o teste de programas Java; a PokeTool (*Potential Uses Criteria Tool for Program Testing*) (Chaim, 1991), que apóia a aplicação dos critérios de teste Todos-Nós, Todos-Arcos e Potenciais-Usos (Maldonado, 1991); e a Proteum (Delamaro, 1993) que apóia o teste de mutação para programas desenvolvidos na linguagem C.

3.4 Ferramentas de Apoio a Padrões de Software

Ferramentas têm sido criadas para auxiliar a utilização de padrões de software no desenvolvimento de aplicações. Essas ferramentas podem ser desde repositórios até geradores automáticos de código.

As ferramentas utilizadas como repositório de padrões tem como meta concentrar os padrões existentes de tal forma que facilitem a busca feita pelo usuário. Por sua vez, as ferramentas que permitem a geração automática de código têm por objetivo facilitar o emprego de padrões nos projetos, evitando erro de desenvolvedores inexperientes ao reutilizar a solução proposta.

Nas seções seguintes são descritas algumas ferramentas, encontradas na literatura, que apóiam a aplicação de padrões. Na Seção 3.4.1 é descrito um repositório de padrões integrado ao RUP proposto por Marinho et al. (2003), na Seção 3.4.2 é resumido o repositório para padrões de projeto hipermídia denominado HPR (*Hypermedia Design Patterns Repository*), na Seção 3.4.3 é sintetizado o FRED (*FRamework EDitor*) e na Seção 3.4.4 é apresentado o *framework* GREN (Gestão de REcursos de Negócio).

3.4.1 Repositório de Padrões Integrado ao RUP

Segundo Marinho et al. (2003), recomenda-se que a utilização de padrões seja realizada nas fases iniciais do desenvolvimento de software. Assim, para agilizar a reutilização de um conjunto mais amplo de padrões disponíveis na literatura, Marinho et al. (2003) propõem um repositório de padrões integrado ao RUP (Kruchten, 2000). Esse repositório utiliza as vantagens de um procedimento de busca em um repositório e a aplicação dos padrões recuperados nas fases iniciais de desenvolvimento utilizando o RUP.

A integração do repositório com um modelo de processo, no caso o RUP, é realizada para facilitar a busca de padrões em diferentes momentos das fases de desenvolvimento e para indicar o momento mais adequado de aplicação desses padrões (Marinho et al., 2003).

Modelo de processo constitui um conjunto de passos parcialmente ordenados para construir um produto de software com qualidade, atendendo as necessidade e exigências dos usuários, dentro do planejamento e orçamentos previstos. A escolha do RUP foi

realizada por ter as características de incremental e iterativa, além de *"possuir as notações mais apropriadas para a reutilização de padrões nas diferentes fases de desenvolvimento, além de já estar sendo largamente adotado por inúmeras empresas nacionais"* (Marinho et al., 2003).

O RUP é uma abordagem disciplinada para o desenvolvimento de software. É utilizado para realizar o controle de qualidade e o gerenciamento de riscos contínuos do projeto, atacando problemas no início do desenvolvimento. Contribui para a construção de uma arquitetura robusta, minimizando re-trabalho, aumentando a reutilização de componentes e ampliando a capacidade de manutenção de sistemas.

As funcionalidades oferecidas pelo repositório proposto seriam (Marinho et al., 2003): "*Cadastrar linguagem de padrões*", "*Cadastrar categoria de padrões*", "*Cadastrar sistema*", "*Cadastrar padrão*", "*Modificar padrão*", "*Excluir padrão*" e "*Realizar busca*". As funcionalidades referentes a manutenção de padrões, isto é, "*Cadastrar padrão*", "*Modificar padrão*" e "*Excluir padrão*", são consideradas responsabilidades do administrador do repositório. Por sua vez, a captura de padrões, ou seja, "*Realizar busca*", é a principal funcionalidade para o usuário do repositório (Marinho et al., 2003).

Com relação a busca do repositório, realiza-se por palavras-chaves nos campos selecionados no *template* do padrão. Assim, existe a possibilidade de realizar uma busca simples (Figura B.1, Apêndice B) ou uma busca avançada (Figura B.2, Apêndice B). Enquanto na busca simples é realizada uma pesquisa geral pela palavra-chave, na busca avançada existe a opção de escolha dos tipos de campos a serem pesquisados (Marinho et al., 2003).

3.4.2 HPR (Hypermedia Design Patterns Repository)

O HPR (*Hypermedia Design Patterns Repository*) (Bolchini et al., 2002) é um repositório de padrões de projeto para hipermídia e aplicações *Web*. É uma iniciativa da ACM (*Association for Computing Machinery*) SIGWEB (*Special Interest Group on Hypertext, Hypermedia and Web*) (ACM SIGWEB Officers and Executive Committee, 1998) em conjunto com a Universidade da Suíça Italiana[1] (*University of Italian Switzerland*). O HPR foi criado com o intuito de facilitar o acesso de projetistas *Web* a padrões de projeto de hipermídia (Bolchini et al., 2002; Garzotto et al., 1999).

Esse repositório permite que sejam armazenados padrões, artigos a respeito de padrões e informações sobre os autores. Os padrões são classificados de acordo com três categorias: *Interface/Layout*, *Estrutura/Navegação* e *Orientados a Contexto*.

Atualmente esse repositório conta com 28 padrões cadastrados, sendo que há algumas regras para a inclusão de padrões. Para submeter um padrão é necessário enviar as seguintes informações (Bolchini et al., 2002):

[1] `http://www.unisi.ch`

- *Descrição do Padrão*: composto por nome do padrão, nomes dos autores, data de criação, palavras-chaves, características dos padrões, problema e motivação;
- *Três exemplos de utilização*: composto por nome da aplicação, endereço, nomes dos autores, editores da aplicação e descrição do porquê a utilização pode ser considerada um exemplo do padrão. Caso menos do que três exemplos de utilização sejam enviados, o padrão é considerado um *proto-padrão*;
- *Apresentação do Autor*: composto por nome, afiliação, e-mail. Opcionalmente podem ser enviados um currículo e uma foto.

Como exemplo do formato de exibição de um padrão, na Figura 3.1 é apresentado o padrão *Excursão Guiada* (*Guided Tour*) (Garzotto et al., 1999) da categoria *Estrutura/-Navegação* (Bolchini et al., 2002). Para cada padrão o usuário visualiza as informações dos componentes do padrão (Seção 2.4.1), o nome dos autores, a data de criação e a data da última revisão do padrão. Além disso, no canto superior direito são exibidos os exemplos de utilização e de padrões relacionados.

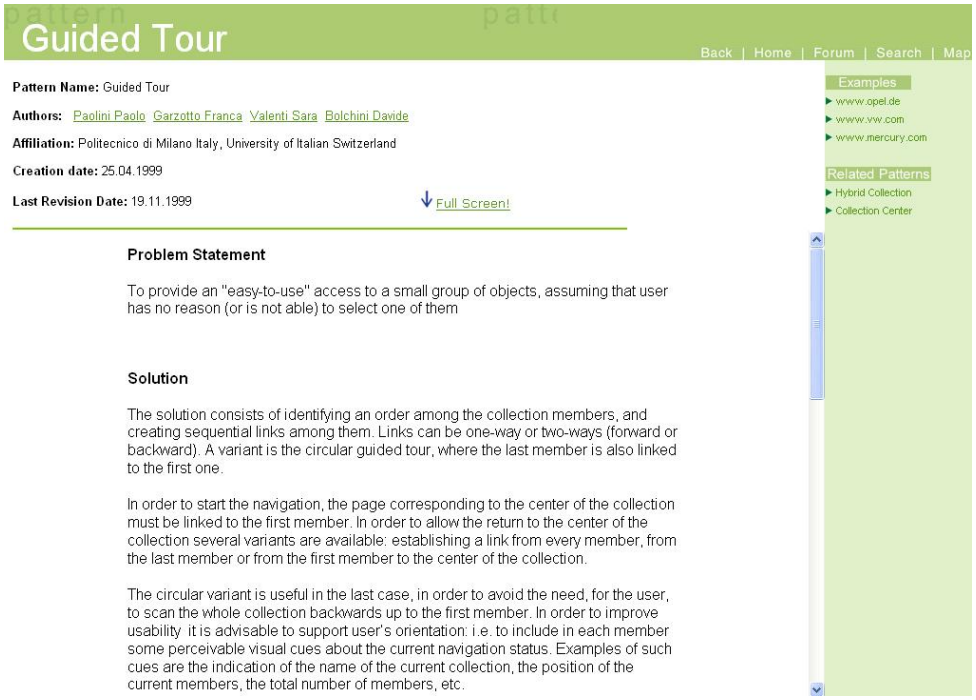

Figura 3.1: Padrão *Excursão Guiada*: exemplo do formato de exibição de um padrão no repositório HPR.

3.4.3 FRED (FRamework EDitor)

O FRED (*FRamework EDitor*) é um ambiente de desenvolvimento de prototipação para Java. É composto por um editor de padrões e um ambiente de programação incremental,

permitindo que padrões sejam instanciados e adicionados incrementalmente. O FRED é uma aplicação orientada a tarefas e pode ser utilizado para a especialização de *frameworks* (Hakala et al., 2001a).

As aplicações criadas com o FRED podem ser desenvolvidas por meio da especialização de *frameworks*. A especialização é feita por meio de uma interface que define uma seqüência de tarefas definindo como deve ser adicionado o código específico da aplicação (Hakala et al., 2001a).

O FRED fornece suporte para as tarefas de especialização, possuindo características conjuntas que outras ferramentas não contém, ou seja (Hakala et al., 2001b):

- Apoio para processos de especialização incrementais, iterativos e interativos: permite que o desenvolvimento de uma aplicação seja particionado em partes menores, com a possibilidade de alterações futuras;
- Instruções especializadas da especialização: como o desenvolvimento da aplicação pode ser incremental, a ferramenta permite colher gradualmente as informações sobre a aplicação, criando gradualmente uma documentação especializada;
- Edição de código fonte sensíveis a arquitetura: possui preocupação em respeitar a linguagem de programação e as regras de arquitetura do *framework*;
- Processos de especialização abertos: permite que atualizações possam ser realizadas em aplicações já finalizadas.

As tarefas geradas pela ferramenta estão diretamente ligadas à criação de elementos de um programa, como por exemplo, classes ou métodos. Existem três opções de utilização do FRED: geração da implementação padrão (*default*), implementação de uma funcionalidade ou exibição de uma implementação existente. O código fornecido é verificado quanto a possíveis violações das definições dos padrões ou das tarefas fornecidas (Hakala et al., 2001a).

O FRED contém um tutorial explicando passo a passo, por meio de um exemplo, como especializar um *framework*. Na Figura B.4, encontrada no Apêndice B, são ilustradas as telas apresentadas ao usuário, onde são executadas as tarefas. O FRED também fornece uma classe de teste para que os usuários possam executar e testar o funcionamento do *framework* especializado pela ferramenta. Na Figura B.5, encontrada no Apêndice B, é apresentada a tarefa que cria essa classe de teste.

3.4.4 GREN (Gestão de REcursos de Negócio)

O GREN (Gestão de REcursos de Negócio) (Braga, 2003; Braga e Masiero, 2000) é um *framework* desenvolvido com base na linguagem de padrões GRN (Gestão de Recursos de

Negócio) (Braga et al., 1999, 2000). É um *framework* caixa-cinza, podendo ser utilizado por meio de herança ou por composição, e que permite a criação de aplicações no domínio de gestão de recursos de negócio.

Para apoiar a instanciação do *framework* GREN, foi desenvolvido o GREN-*Wizard* (Braga, 2002; Braga e Masiero, 2002) que fornece apoio a aplicação da linguagem GRN e permite a geração automática do código e do banco de dados da aplicação final. A aplicação é gerada na linguagem de programação Smalltalk (Smalltalk.org, 2004) e o banco de dados é criado em MySQL (*My Structured Query Language*) (MySQL AB, 2006).

O conteúdo da interface do GREN-*Wizard* é similar as opções de escolhas encontradas na GRN, facilitando a associação. Depois de concluída a instanciação, o usuário pode alterar o código da aplicação resultante para adequar-se aos requisitos não oferecidos pelo GREN (Braga, 2002). Uma tela ilustrando a interface do GREN-*Wizard* pode ser encontrada na Figura B.3, contida no Apêndice B.

3.5 Considerações Finais

Neste capítulo foram listados alguns ambientes e ferramentas, encontrados na literatura, que apóiam o desenvolvimento de aplicações. No caso de métodos, foram descritas ferramentas específicas para a *Web*, enquanto para processos, teste e padrões de software foram descritos ambientes e ferramentas que podem ser utilizados em diferentes contextos.

Processos, métodos, padrões, ambientes e ferramentas têm o objetivo comum de apoiar os engenheiros de softwares no desenvolvimento de aplicações. Diversos *frameworks* têm sido criados e utilizados no desenvolvimento de aplicações *Web*. Assim, no próximo capítulo serão apresentados alguns *frameworks* que foram analisados quanto a utilidade para facilitar a implementação do ambiente Peônia, proposto neste projeto de mestrado.

Capítulo

4

Frameworks de Apoio ao Desenvolvimento

4.1 Considerações Iniciais

Frameworks são aplicações semi-completas com componentes estáticos e dinâmicos utilizados para a produção de aplicações específicas do usuário (Fayad e Johnson, 2000). Contém um conjunto de blocos de contrução que podem ser utilizados, estendidos ou customizados para soluções específicas de computação. Utilizando *frameworks*, desenvolvedores de software não precisam iniciar do nada ao desenvolverem uma aplicação (Taligent Inc., 1994).

Assim, para auxiliar na implementação do ambiente Peônia foi realizada uma pesquisa sobre *frameworks* que auxiliassem no gerenciamento de persistência e na estruturação da interface gráfica.

Na Seção 4.2 são apresentados *frameworks* utilizados para auxiliar o desenvolvimento da camada de persistência de aplicações e a comparação entre o Hibernate e o iBATIS. Na Seção 4.3 é fornecida uma descrição de tecnologias empregadas no desenvolvimento da camada de apresentação e explicado o *framework* ZK. Por fim, na Seção 4.4 são apresentadas as considerações finais a respeito dos *frameworks* apresentados, inclusive os escolhidos para o desenvolvimento do ambiente proposto neste projeto de mestrado.

4.2 Frameworks de Persistência

Atualmente existem diversos *frameworks* de persistência, tanto de código aberto, quanto pagos. O principal objetivo da maior parte deles é tornar transparente para os usuários as transações realizadas em um banco de dados, realizando a conexão e o mapeamento Objeto/Relacional (mapeamento O/R). Como exemplos de *frameworks* de persistência de código aberto podem ser citados Apache OpenJPA[1], Castor[2], Cayenne[3], iBATIS[4], jPersist[5], Hibernate[6], O/R Broker[7], OJB[8] (*Apache ObJectRelationalBridge*) e Torque[9] (Java-Source.Net, 2007). Dentre esses *frameworks* mencionados, destacam-se o Hibernate e o iBATIS, utilizados por uma grande quantidade de desenvolvedores e por possuírem diversos fóruns e comunidades na *Web* destinados à troca de informações.

Assim, nas seções seguintes esses dois *frameworks* são apresentados resumidamente e comparados quanto algumas de suas características. Na Seção 4.2.1 é apresentado um resumo sobre o Hibernate, na Seção 4.2.2 é descrito o iBATIS e na Seção 4.2.3 esses dois *frameworks* de persistência são comparados.

4.2.1 Hibernate

Hibernate (Red Hat Middleware, LLC, 2006) é um serviço de consulta e de persistência para mapeamento O/R. Por meio da sua utilização, classes de persistência podem ser desenvolvidas seguindo os princípios de orientação a objeto. As consultas podem ser criadas utilizando a extensão portável do SQL (*Structured Query Language*) do Hibernate, denominada HQL (*Hibernate Query Language*), SQL nativo do banco de dados utilizado, ou as API orientadas a objeto *"Criteria"*[10] e *"Example"*[11]. O objetivo do projeto é aliviar os desenvolvedores de 95% da programação ligada à persistência de dados, comparada com a programação manual utilizando SQL e API JDBC (*Java Database Connectivity*) (Sun Microsystems, Inc., 2002). A API JDBC tem por objetivo fornecer uma maneira comum de acesso à dados para a linguagem de programação Java, permitindo que informações contidas em qualquer SGBD (*Sistema de Gerenciamento de Banco de Dados*) possam ser acessadas.

[1] `http://openjpa.apache.org/`
[2] `http://www.castor.org/`
[3] `http://cayenne.apache.org/`
[4] `http://ibatis.apache.org/index.html`
[5] `http://www.jpersist.org/`
[6] `http://www.hibernate.org/`
[7] `http://orbroker.sourceforge.net/`
[8] `http://db.apache.org/ojb/`
[9] `http://db.apache.org/torque/`
[10] `http://www.hibernate.org/hib_docs/v3/api/org/hibernate/Criteria.html`
[11] `http://www.hibernate.org/hib_docs/v3/api/org/hibernate/criterion/Example.html`

O HQL é uma linguagem de consulta semelhante ao padrão SQL ANSI (International Organization for Standardization, 1992), no entanto, é orientada a objetos, abrangendo conceitos como herança, polimorfismo e associação. Permite o emprego de funções, expressões e agrupamentos sendo que a tradução correta das informações depende do dialeto utilizado pela aplicação (Red Hat Middleware, LLC, 2007b). Também é possível utilizar subconsultas dentro de consultas no Hibernate, no entanto, só funciona corretamente se a linguagem do banco de dados também fornecer esse apoio. Por ser semelhante ao SQL e empregar orientação a objetos, usuários que conhecem esses conceitos têm facilidade de aprender e utilizar o HQL.

O Hibernate *Core* para Java é o responsável por gerar o SQL, poupando o desenvolvedor de implementar manualmente a manipulação dos resultados JDBC e conversão para objetos, ao mesmo tempo que provê a portabilidade para diferentes SGBDs. Além do suporte para desenvolvimento Java, o projeto Hibernate também possui o NHibernate, que é uma porta do Hibernate *Core* para persistência de aplicações desenvolvidas em .NET[12]. O Hibernate e o NHibernate utilizam licença LGPL (*GNU Lesser General Public License*)[13], permitindo que seja utilizado em projetos de código aberto e comerciais (Red Hat Middleware, LLC, 2006).

Hibernate adapta-se ao processo de desenvolvimento empregado, podendo ser utilizado tanto em projetos iniciados do zero, quanto em trabalhos baseados em banco de dados existentes (Red Hat Middleware, LLC, 2006). Para utilizá-lo é preciso conhecer as suas interfaces de programação, cujo objetivo principal é limitar o máximo possível o acesso direto aos componentes de software, facilitando a utilização. Na prática as APIs de mapeamento O/R do Hibernate não são tão restritas e pequenas, no entanto, não é necessário entender todas as interfaces do Hibernate para utilizá-lo (Bauer e King, 2004).

Na Figura 4.1 são ilustrados os papéis das interfaces mais importantes do Hibernate nas camadas de persistência e de negócio. A camada de negócio comporta-se como um cliente da camada de persistência. As interfaces do Hibernate apresentadas na Figura 4.1 são classificadas como (Bauer e King, 2004):

- Interfaces requisitadas pela aplicação para executar operações básicas de CRUD (*Create, Retrieve, Update e Destroy*) e de busca. São o ponto principal da lógica de negócio e de controle no Hibernate. Elas incluem *"Session"*[14], *"Transaction"*[15] e *"Query"*[16].

[12] `http://msdn2.microsoft.com/pt-br/netframework/default(en-us).aspx`
[13] `http://www.gnu.org/licenses/lgpl.html`
[14] `http://www.hibernate.org/hib_docs/v3/api/org/hibernate/Session.html`
[15] `http://www.hibernate.org/hib_docs/v3/api/org/hibernate/Transaction.html`
[16] `http://www.hibernate.org/hib_docs/v3/api/org/hibernate/Query.html`

- Interfaces requisitadas pelo código de infra-estrutura da aplicação para configurar o Hibernate, sendo que a classe mais importante é a *"Configuration"* [17].

- Interfaces de retorno de requisições (*"Callback"*), responsáveis pela reação das aplicações a eventos ocorridos dentro do Hibernate. Incluem as classes de interface *"Interceptor"* [18], *"Lifecycle"* [19] e *"Validatable"* [20].

- Interfaces que permitem estender as funcionalidades de mapeamento do Hibernate. Incluem as classes *"UserType"* [21], *"CompositeUserType"* [22] e *"IdentifierGenerator"* [23].

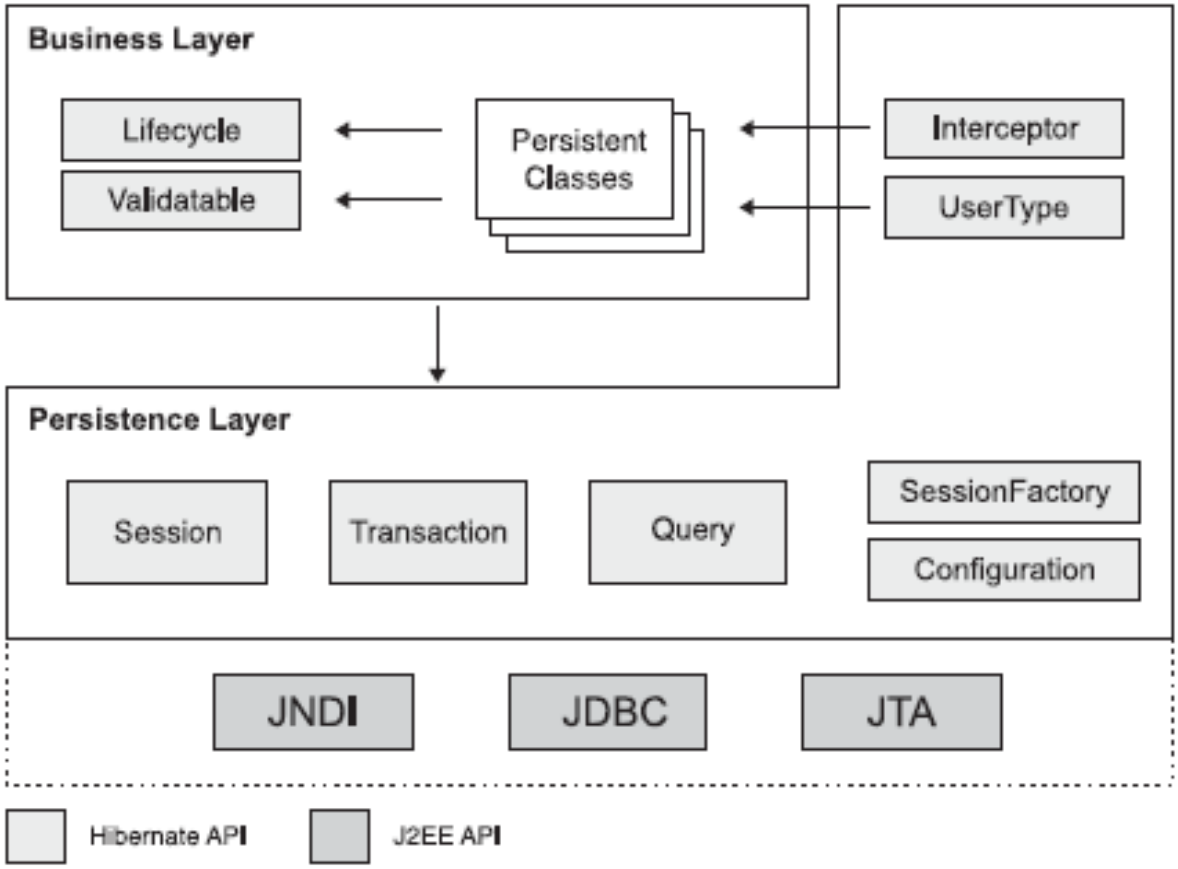

Figura 4.1: APIs do Hibernate em uma arquitetura de camadas. (Bauer e King, 2004)

[17] http://www.hibernate.org/hib_docs/v3/api/org/hibernate/cfg/Configuration.html
[18] http://www.hibernate.org/hib_docs/v3/api/org/hibernate/Interceptor.html
[19] http://www.hibernate.org/hib_docs/v3/api/org/hibernate/classic/Lifecycle.html
[20] http://www.hibernate.org/hib_docs/v3/api/org/hibernate/classic/Validatable.html
[21] http://www.hibernate.org/hib_docs/v3/api/org/hibernate/usertype/UserType.html
[22] http://www.hibernate.org/hib_docs/v3/api/org/hibernate/usertype/CompositeUserType.html
[23] http://www.hibernate.org/hib_docs/v3/api/org/hibernate/id/IdentifierGenerator.html

Hibernate utiliza APIs Java, como o JDBC, JTA[24] (*Java Transaction API*) e JNDI[25] (*Java Naming and Directory Interface*). O JDBC provê abstração das funcionalidades comuns de banco de dados relacionais, permitindo que a maioria dos bancos de dados com um *"driver"* [26] JDBC possam usufruir do Hibernate. Por sua vez, o JTA e JNDI permitem que o Hibernate seja integrado com Servidores de Aplicação J2EE (Bauer e King, 2004).

Hibernate pode ser configurado para executar em quase todas as aplicações e ambientes de desenvolvimento Java. Em geral, é utilizado em aplicações cliente/servidor de duas ou três camadas, sendo que o Hibernate fica implantado apenas no servidor. Hibernate pode ser configurado para dois tipos de ambientes (Bauer e King, 2004; Red Hat Middleware, LLC, 2007b):

- **Gerenciados**: o container utilizado é responsável por gerenciar as transações. Como exemplos de container podem ser citados o JBoss[27], BEA WebLogic[28] e IBM WebSphere[29];
- **Não Gerenciados**: o Hibernate é responsável pelo *"pool"* de conexões. O desenvolvedor é que deve implementar os limites das transações, iniciando, confirmando e desfazendo as transações.

Aplicações que utilizam Hibernate contêm classes de persistência que são mapeadas como tabelas do banco de dados, sendo necessário apenas informar como essas classes devem ser persistidas, ou seja, a correspondência entre as propriedades de uma classe e as colunas de uma tabela. Em geral, isso é feito por meio de um documento de mapeamento XML, mas pode ser realizada também por meio de anotações Java. (Bauer e King, 2004). Um exemplo de documento de mapeamento XML do Hibernate é ilustrado na Figura C.1 contida no Apêndice C.

O Hibernate também permite que consultas escritas em SQL nativo e execução de funções e procedimentos armazenados no banco de dados sejam definidas separadamente nos documento de mapeamento XML (Red Hat Middleware, LLC, 2006), como ilustrado na Figura C.2 contida no Apêndice C.

[24] `http://java.sun.com/products/jta/`
[25] `http://java.sun.com/products/jndi/`
[26] Administrador de dispositivo
[27] `http://www.jboss.com/products/index`
[28] `http://www.bea.com/`
[29] `http://www-306.ibm.com/software/websphere/`

4.2.2 iBATIS

O projeto iBATIS (iBATIS Team, 2007) começou em 2001 e tinha inicialmente o objetivo de desenvolver soluções de software para criptografia. O nome iBATIS foi inspirado nesse objetivo inicial, sendo composto pelas palavras *"**i**nternet"* e *"a**batis**"*. A palavra de origem francesa *"abatis"* é utilizada para denominar um obstáculo defensivo composto de troncos de árvores sobrepostos diante do inimigo.

No entanto, o projeto começou a direcionar suas atenções para a *Web*, lançando softwares e *frameworks* para este setor. Um dos resultados do projeto é a Camada de Banco de Dados iBATIS composta pelos *frameworks* SQL *Maps* e DAO (*Data Access Objects*). Assim, atualmente o projeto iBATIS concentra-se na manipulação desses dois *frameworks* da camada de persistência também conhecidos como *iBATIS Data Mapper* e *iBATIS Data Access Objects*.

O *framework iBATIS Data Mapper* (iBATIS Team, 2007) tem por objetivo facilitar a utilização de banco de dados em aplicações Java e .NET. Por meio de um descritor XML, relaciona objetos com procedimentos armazenados no banco de dados (*stored procedure*) ou com instruções SQL. O projeto também fornece o RBatis que é uma porta do iBATIS para o desenvolvimento utilizando o *Ruby on Rails Framework*[30]. O código e a documentação do iBATIS e do RBatis estão sob a licença *"Apache License 2.0"* [31].

Utilizando o *iBATIS Data Mapper*, a quantidade de código para acessar um banco de dados relacional é menor do que a normalmente requerida. A filosofia defendida pela equipe do iBATIS é que o *Data Mapper* forneça 80% de funcionalidades JDBC utilizando 20% de código (iBATIS Team, 2006). Uma das vantagens do iBATIS é a sua simplicidade, pois é necessário conhecer apenas *JavaBeans*, XML e SQL para utilizá-lo (iBATIS Team, 2007). Segundo Sun Microsystems, Inc. (1997), *JavaBeans* são componentes de software reutilizáveis que podem ser manipulados visualmente e customizados para um propósito, além de serem facilmente desenvolvidos e combinados para criar aplicações sofisticadas.

O *framework* utiliza descritores XML para mapear *JavaBeans*, implementações da interface *"Map"* [32], tipos primitivos encapsulados, por exemplo, *"String"* [33] e *"Integer"* [34], e os próprios documentos XML para declarações SQL. Na Figura 4.2 é apresentado um diagrama do fluxo de execução do *iBATIS Data Mapper*, dividido em três partes principais:

[30] `http://www.rubyonrails.org/`
[31] `http://www.apache.org/licenses/LICENSE-2.0`
[32] `http://java.sun.com/j2se/1.4.2/docs/api/java/util/Map.html`
[33] `http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html`
[34] `http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Integer.html`

1. **Prover um objeto do tipo parâmetro.** Esse "objeto parâmetro" é utilizado para a inserção de valores de entrada em declarações de atualização, ou como cláusulas em consultas.

2. **Executar a declaração de mapeamento.** O *framework Data Mapper* cria uma instância de *"PreparedStatement"*[35], utiliza os dados contidos no "objeto parâmetro" para configurar os parâmetros necessários, executa a declaração e constrói um objeto com o resultado retornado pelo iBATIS em um objeto *"ResultSet"*[36].

3. **Retornar resultado.** Com relação ao retorno, se for uma atualização, o número de colunas alteradas é retornado. Caso seja uma consulta, tanto um único objeto, quanto uma coleção de objetos podem ser retornados. Com relação ao tipo retornado, assim como os parâmetros de entrada, podem ser retornados objetos *JavaBeans*, *"Maps"*, tipos primitivos encapsulados ou XML.

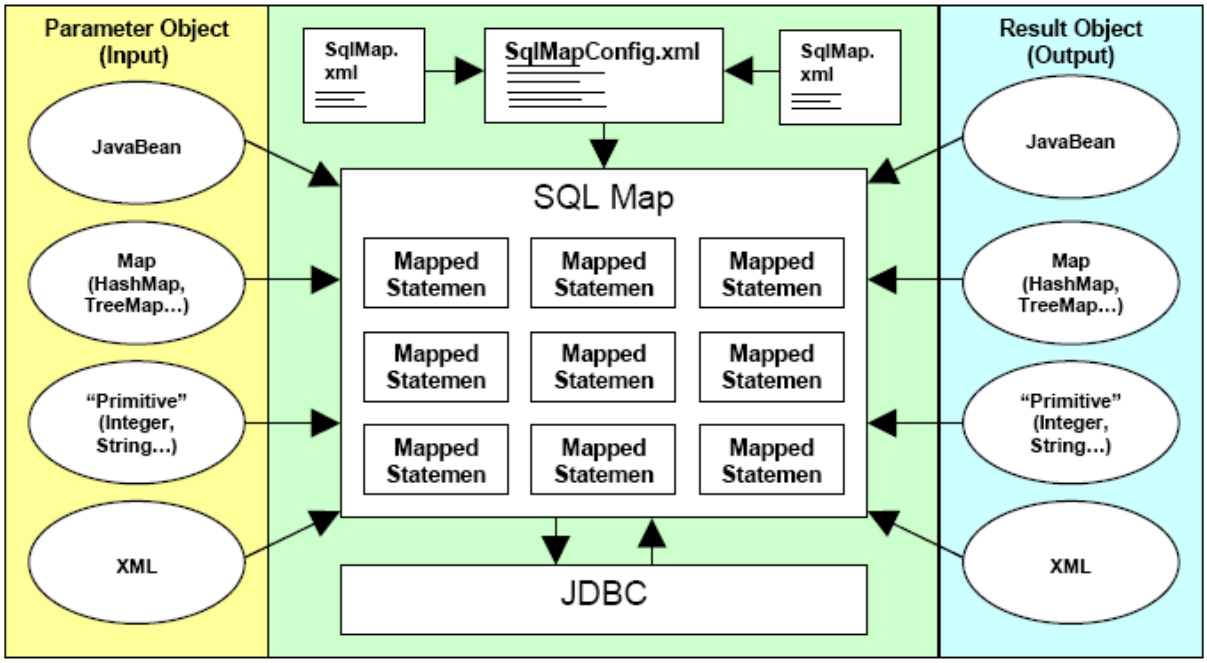

Figura 4.2: *iBATIS Data Mapper*: Fluxo de execução.

No iBATIS, as transações realizadas com o banco de dados são mapeadas por meio de consultas escritas em SQL nativo, reduzindo esforços de implementação de desenvolvedores acostumados com a linguagem do SGBD utilizado. As consultas podem ser inseridas nos arquivos XML de mapeamento O/R, facilitando alterações nas transações, quando necessárias. No Apêndice C, na Figura C.3, é apresentado um exemplo de descritor XML utilizado pelo iBATIS para realizar o mapeamento O/R.

[35] `http://java.sun.com/j2se/1.4.2/docs/api/java/sql/PreparedStatement.html`
[36] `http://java.sun.com/j2se/1.4.2/docs/api/java/sql/ResultSet.html`

O outro componente do *framework* é o *iBATIS Data Access Objects*, uma camada de abstração que encobre os detalhes da solução de persistência e provê uma API (*Application Programming Interface*) comum para ser utilizada na aplicação do usuário. Utilizando DAO é possível criar componentes simples que provêem acesso ao banco de dados sem que a aplicação desenvolvida conheça os detalhes específicos da implementação, além de permitir que a aplicação seja dinamicamente configurada para utilizar diferentes mecanismos de persistência (iBATIS Team, 2007).

Para os usuários de Java, o *framework Data Access Objects* está empacotado na Camada do Banco de Dados do iBATIS, juntamente com *framework SQL Maps*. Apesar de estarem empacotados juntos, eles podem ser utilizados independentemente. Para os usuários .NET, os dois *frameworks* são obtidos separadamente (iBATIS Team, 2007).

4.2.3 Comparação Entre Hibernate e iBATIS

Com relação ao desenvolvimento de aplicações, tanto o Hibernate quanto o iBATIS oferecem apoio a implementação da camada de persistência de aplicações Java ou .NET, tornando transparente para os usuários o mapeamento O/R. Como o interesse deste trabalho de mestrado está voltado para o desenvolvimento de um ambiente *Web* utilizando a linguagem de programação Java, apenas o suporte à implementação de aplicações Java foi averiguado. Destaca-se que a comparação entre os *frameworks* foi realizada com a intenção de avaliar qual deles deveria ser utilizado no ambiente Peônia, portanto, a comparação foi feita de maneira restrita, sem estudos profundos sobre o impacto da sua generalização.

Para comparar o Hibernate e iBATIS, as documentações de ambos os *frameworks* foram analisadas e aplicações exemplos foram desenvolvidas para realizar o levantamento das suas diferenças, vantagens e desvantagens. Para cada *frameworks* duas versões da mesma aplicação foram implementadas, sendo que uma versão utilizou documentos XML para mapeamento e a outra utilizou APIs e anotações Java, com o mapeamento realizado diretamente no código fonte.

As aplicações exemplos desenvolvidas realizam manipulação de dados em uma tabela contendo informações sobre usuários de um sistema qualquer, por exemplo, do ambiente proposto neste trabalho de mestrado. A classe do conceito *"Usuário"* é representada na Figura 4.3.

Com relação à documentação dos dois *frameworks*, ambos a fornecem, tanto *online*, como por arquivo no formato PDF, com dados sobre a arquitetura, componentes e exemplos para configurar e criar aplicações utilizando-os. Em especial, o Hibernate também oferece uma documentação completa das suas APIs (Red Hat Middleware, LLC, 2007a) no mesmo formato da especificação da API para Java (Sun Microsystems, Inc., 2003b), o que facilita a consulta, utilização e compreensão por desenvolvedores Java.

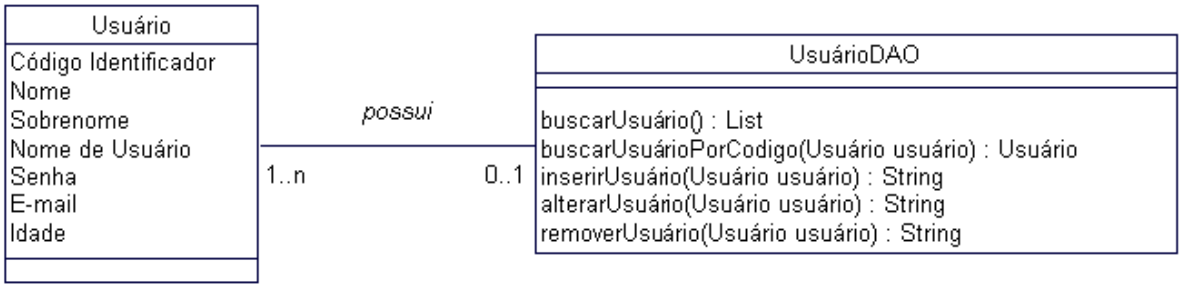

Figura 4.3: Modelo da classe *Usuário* utilizada no desenvolvimento das aplicações exemplos.

A equipe desenvolvedora do iBATIS não esclarece dúvidas sobre a utilização, mas oferecem FAQ (*Frequent Asked Questions*), listas de e-mails e *Wiki* para que usuários compartilhem suas dúvidas e recebam respostas de outros desenvolvedores. Já a equipe responsável pelo Hibernate fornece apoio comercial e treinamentos, além de também manter listas de e-mails, *Wiki* e fóruns de desenvolvedores para que dúvidas sejam esclarecidas. Segundo Tazzoli et al. (2004), um *Wiki* é o apelido para *Wiki Wiki Web*, que é um *site Web* onde usuários podem contribuir adicionando informações. Ao criar uma página em um *Wiki*, qualquer usuário pode editar o seu conteúdo colaborativamente.

Para que os usuários contribuam com o projeto, nos *sites* oficiais do iBATIS[37] e do Hibernate[38] existem instruções de como colaborar com a equipe desenvolvedora, seja enviando erros, como sugestões de novas funcionalidades.

Como mencionado no início da Seção 4.2, esses dois *frameworks* foram selecionados para uma análise mais detalhada por serem utilizados por uma quantidade maior de usuários. Além disso, são os que mais possuem fóruns e comunidades de desenvolvedores que trocam experiências sobre os *frameworks*. No entanto, é importante destacar que a quantidade de usuários do Hibernate, analisando a quantidade de dúvidas, fóruns, comunidades e *sites* ligados ao assunto, é maior do que a do iBATIS.

Durante o desenvolvimento das aplicações exemplos surgiram erros e dúvidas sobre os *frameworks*. Como tanto o Hibernate, quanto o iBatis, incentivam a utilização de listas de discussões, fóruns e comunidades para esclarecer as dúvidas dos desenvolvedores e solucionar os problemas encontrados, foram realizadas buscas nas listas de discussões propostas pelo Hibernate e iBATIS e na *Web*. Para todos os erros e dúvidas que surgiram no desenvolvimento das aplicações exemplos empregando o Hibernate foram encontradas respostas, inclusive com diferentes maneiras de solucionar o mesmo erro. No entanto, o mesmo não aconteceu com o iBATIS, pois em pelo menos três casos foram encontra-

[37] `http://ibatis.apache.org/index.html`
[38] `http://www.hibernate.org/`

das mais dúvidas semelhantes, ou comentários não confiáveis, do que respostas para o problema.

Com relação à concorrência, segundo Red Hat Middleware, LLC (2007b), o Hibernate garante o tratamento da concorrência em caso de conflito de dados, além de fornecer recursos para que o usuário realize o seu próprio tratamento. No entanto, a mesma garantia e recursos de controle de concorrência não foram encontrados na documentação do iBATIS (iBATIS Team, 2007). Nas buscas pelo assunto realizadas nos fóruns e comunidades da *Web* desenvolvedores recomendam que seja implementado pelo usuário o controle de concorrência, utilizando recursos fornecidos pelo SQL do SGBD empregado.

Tanto o Hibernate, quanto o iBATIS, permitem que a conexão ao banco de dados e mapeamento O/R sejam realizados por meio de documentos XML, ou por anotações Java. A configuração e os passos necessários para conexão e mapeamento são semelhantes, com diferença nos comandos utilizados. Assim, ao conhecer um dos *frameworks*, o esforço no aprendizado e a facilidade para utilizar o outro tende a ser reduzido. No entanto, é necessário considerar que o iBATIS é mais simples de aprender, pois concentra o desenvolvimento na utilização de SQL, enquanto o Hibernate possui muitas APIs e o HQL, exigindo em geral um tempo maior de aprendizado.

Com o mapeamento O/R que os *frameworks* realizam, a quantidade de código diminui, se comparado ao produzido com a implementação manual de SQL em conjunto com APIs JDBC. No entanto, o iBATIS explora as vantagens do emprego da linguagem SQL, enquanto o Hibernate utiliza as facilidades da orientação a objetos. O iBATIS permite que consultas complexas sejam criadas com a mesma sintaxe que a utilizada em um SGBD específico, além de permitir mapeá-las separadamente em arquivos XML facilitando a manutenção de consultas que mudam com freqüência. Por sua vez, o Hibernate favorece os usuários que possuem menos conhecimentos na linguagem SQL nativa do banco e mais experiência em orientação a objetos, pois fornece APIs para acessar e gerenciar dados armazenados em bancos sem que seja necessário criar a consulta em SQL, além de permitir também que seja utilizado o HQL, realizando a comparação direta entre as classes da aplicação (Red Hat Middleware, LLC, 2007b).

Para que o mapeamento O/R do iBATIS funcione é necessário que as tabelas estejam criadas no banco de dados utilizado. Já o Hibernate possui um parâmetro de configuração indicando que as tabelas devem ser geradas automaticamente de acordo com o mapeamento fornecido, ou indicando que as tabelas já existem e apenas devem ser atualizadas se existirem diferenças entre o mapeamento e as tabelas. É importante comentar que as atualizações só funcionam caso sejam adicionadas colunas, ou seja, em caso de alteração ou remoção de colunas no mapeamento, as mesmas permanecem idênticas no banco de dados.

Por fim, destaca-se novamente que o iBATIS e Hibernate são *frameworks* adotados crescentemente pela comunidade de desenvolvedores Java por facilitarem a manipulação de transações com o banco de dados, além de tornarem o código mais limpo e utilizarem técnicas para otimizar as transações. Ambos continuam sofrendo alterações, corrigindo problemas e assimilando funcionalidades, melhorando a persistência de dados ao qual se propõem a realizar de maneira eficiente e simples.

4.3 Frameworks de Apresentação

Como mencionado na Seção 2.2, as aplicações *Web* evoluíram de páginas estáticas, utilizadas apenas para exibição de informações textuais, para páginas dinâmicas e complexas, com integração com banco de dados e outros sistemas, além de exigirem alto desempenho e disponibilidade contínua. Teconologias *Web*, como o HTTP (*Hypertext Transfer Protocol*) (Lafon, 2005) e HTML, têm origem no modelo baseado em páginas e comunicação *"stateless"*, ou seja, com comunicação direta e sem armazenar informações sobre estados anteriores. No entanto, com a evolução exigida pelos negócios e a utilização da *Web* como principal plataforma para o desenvolvimento de aplicações, nota-se que esse modelo baseado em páginas encontra dificuldades para representar visualmente a complexidade das aplicações *Web* atuais (Potix Corporation, 2007a).

Utilizando o modelo baseado em páginas, as aplicações executadas no servidor devem lidar com quase tudo: a análise da requisição, montagem da resposta a ser enviada, roteamento dos processos que levam um usuário de uma página a outra e gerenciamento de erros cometidos pelos usuários. Diversos *frameworks* têm surgido para auxiliar o desenvolvimento, como por exemplo, ZK (Potix Corporation, 2007b), Struts[39], Tapestry[40] e JSF[41]. No entanto, devido à distância existente entre os modelos baseados em páginas e as aplicações modernas, o aprendizado e utilização desses *framewoks* nem sempre é uma tarefa agradável, intuitiva e simples (Potix Corporation, 2007a).

Aplicações *Web* evoluíram de páginas estáticas em HTML para páginas em DHTML, em seguida para Applets e Flash e por último para tecnologias AJAX (*Asynchronous JavaScript and XML*). AJAX (Crane et al., 2006) é um tipo novo de DHTML que utiliza o navegador padrão (*"browser"*) e *JavaScript*[42] (Mozilla Foundation, 2007), fornecendo para aplicações *Web* o mesmo nível de interatividade e resposta que aplicações *"desktop"*.

[39] `http://struts.apache.org/`
[40] `http://tapestry.apache.org/`
[41] `http://java.sun.com/javaee/javaserverfaces/`
[42] JavaScript é uma linguagem de *script* pequena, leve, orientada a objetos e multi-plataforma.

Diferentemente de Applets[43] (Sun Microsystems, Inc., 2008a) e Flash[44] (Adobe Systems Incorporated, 2008), não necessitam de *"plugins"* (Potix Corporation, 2007a).

Assim como DHTML, AJAX utiliza o *JavaScript* para aguardar eventos disparados por atividades executadas pelos usuários, manipulando visualmente as informações e criando dinamicamente páginas no navegador. Também realiza comunicação assíncrona com o servidor, sem haver necessidade de sair da página exibida ou de renderizar novamente a página inteira exibida ao usuário. AJAX disponibiliza para as aplicações *Web* os componentes comuns em aplicações *"desktop"*, além de permitir que todo o conteúdo seja atualizado dinamicamente de acordo com o controle realizado pela aplicação (Potix Corporation, 2007a).

Provendo a interatividade que usuários exigem, AJAX aumenta a complexidade e habilidades exigidas para o desenvolvimento das aplicações *Web*. Desenvolvedores necessitam manipular o DOM[45] (*Document Object Model*) no navegador e lidar com a comunicação realizada com o servidor, utilizando possivelmente API *JavaScript* incompatíveis e com defeitos. Além disso, para melhorar a interatividade, desenvolvedores têm que replicar subconjuntos de dados e da lógica de negócio da aplicação para o navegador, aumentando o custo de manutenção e a sincronização entre os dados. Portanto, aplicações AJAX *"ad hoc"* não são diferentes de aplicações *Web* tradicionais, a não ser pela maneira como processam requisições. Desenvolvedores ainda são os responsáveis por tratar os problemas encontrados no modelo baseado em páginas *"stateless"* (Potix Corporation, 2007a).

Na próxima seção, o *framework* ZK, utilizado na implementação da interface visual do ambiente Peônia, é apresentado resumidamente.

4.3.1 ZK

O ZK é um *framework* dirigido a eventos, baseado em componentes e com o objetivo de fornecer uma rica interface com o usuário em aplicações *Web*. O ZK contém um *motor* dirigido a eventos baseado em AJAX, um rico conjunto de componentes XUL (*XML User Interface Language*) (Ben Goodger and Ian Hickson and David Hyatt and Chris Waterson, 2001) e XHTML (*Extensible HyperText Markup Language*) (World Wide Web Consortium, 2002) e uma linguagem de marcação denominada ZUML (*ZK User Interface Markup Language*). O objetivo do ZK é trazer o modelo empregado em aplicações *"desktop"* para a *Web* (Potix Corporation, 2007a).

Empregando o ZK, as aplicações são compostas por componentes XHTML e XUL, manipuladas de acordo com os eventos disparados pelas atividades dos usuários, assim

[43] Programas na linguagem de programação Java que podem ser incluídos em páginas HTML.

[44] Aplicativo com conteúdo interativo criado utilizando o Adobe Flash para plataformas digitais, *Web* ou móveis.

[45] `http://www.w3.org/DOM/`

como acontece em aplicações *"desktop"*. Ao contrário de outros *frameworks*, AJAX é utilizado como uma tecnologia embutida. A sincronização do conteúdo dos componentes e o *"pipelining"* de evento, ou seja, o gerenciamento das execuções em paralelo dos eventos, é realizado automaticamente pelo *motor* do ZK (Potix Corporation, 2007a).

Os usuários da aplicação *Web* desenvolvida têm a mesma interatividade e resposta que uma aplicação *"desktop"* ao mesmo tempo que o desenvolvimento é realizado com a mesma simplicidade que aplicações *"desktop"* (Potix Corporation, 2007a).

Com relação à linguagem de marcação ZUML, assim como XHTML, habilitam os desenvolvedores a projetar interfaces de usuários sem programação. Com o espaço de nome (*"namespace"*) XML, ZUML integra perfeitamente diferentes conjunto de *"tags"* na mesma página, sendo que atualmente, apóia dois conjuntos de *"tags"*, XUL e HTML (Potix Corporation, 2007a). ZUML pode ser utilizada em conjunto com código Java e expressões EL (*Expression Language*) (Sun Microsystems, Inc., 2007).

ZK é projetado para ser compacto e é destinado apenas à camada de apresentação, no entanto, não é recomendado para aplicações que executam a maioria das tarefas no cliente, como jogos de ação em 3D. O *framework* não lida com a persistência ou comunicação inter-servidor, não exige que sejam utilizadas tecnologias de *"back-end"* e funciona com tecnologias de *"middleware"* como JDBC, Hibernate, Java Mail, EJB (*Enterprise JavaBeans*) (Sun Microsystems, Inc., 2008b) ou JMS (*Java Message Service*) (Sun Microsystems, Inc., 2008d) (Potix Corporation, 2007a).

Atualmente, suporta ZUL e XHTML e, futuramente, irá suportar também XAML (*Extensible Application Markup Language*) (Microsoft Corporation, 2008), XQuery (World Wide Web Consortium, 2007b) e outros. Com o ZK para dispositivos móveis (*"mobile"*), as aplicações desenvolvidas podem ser visualizadas em qualquer dispositivo de suporte a J2ME (*Java Platform, Micro Edition*) (Sun Microsystems, Inc., 2008c), como PDA (*Personal Digital Assistant*) (Tech-FAQ, 2007), celulares e consoles de games, sem necessidade de adaptações (Potix Corporation, 2007a).

Destaca-se que o ZK não obriga a utilização de padrões de projeto, como o MVC, mas não fornece barreiras, deixando essa escolha para os desenvolvedores (Potix Corporation, 2007a).

4.4 Considerações Finais

Neste capítulo foram listados alguns *frameworks*, encontrados na literatura, que apóiam o desenvolvimento *Web*, em especial os que auxiliam o desenvolvimento de aplicações Java.

Com relação à persistência, os *frameworks* Hibernate e iBATIS foram analisados e suas vantagens e desvantagens descritas. Neste projeto optou-se por utilizar o Hibernate

na manipulação de transações e realização da persistência do ambiente proposto. Apesar do iBATIS ser mais fácil de aprender, as opções oferecidas pelo Hibernate são maiores e, como a documentação do Hibernate é rica em detalhes, não foi tomado como um problema o esforço maior no aprendizado. Além disso, o Hibernate foi escolhido em razão da maior quantidade de usuários no mercado atual, facilidade na localização de respostas para erros e dúvidas que poderiam surgir, garantia do tratamento de concorrência, geração automática das tabelas mapeadas, possibidade de escolha entre o emprego do HQL e de SQL nativo da base de dados, ao caráter orientado a objetos do HQL e pela organização da documentação fornecida.

O *framework* ZK foi escolhido para ser utilizado na implementação da interface visual do ambiente Peônia principalmente por permitir que elementos sejam adicionados dinamicamente a uma página criada sem precisar renderizá-la novamente. É flexível, pois pode ser utilizado com diferentes tecnologias, como *JavaScript*, HTML e XML. O conjunto de componentes que oferece é fácil de ser empregado por desenvolvedores que tenham conhecimento de HTML e XML, além de produzir uma boa interface de interação com o usuário.

Nos Capítulos 6 e 7 o ambiente Peônia, desenvolvido com os *frameworks* selecionados e descritos neste capítulo, é detalhado quanto às suas funcionalidades e explicado o projeto executado.

Capítulo

5

Padrões e Teste de Software no Desenvolvimento de Aplicações

5.1 Considerações Iniciais

Como mencionado na Seção 2.4, a utilização de padrões de software pode trazer benefícios, como a difusão de boas práticas no desenvolvimento de sistemas e o aumento da qualidade do software com o emprego de soluções bem sucedidas e comprovadas. Processos de desenvolvimento e atividades de VV&T também são conceitos de Engenharia de Software utilizados para aumentar a qualidade de software.

Apesar da ampla divulgação e utilização, a maior parte dos estudos é voltada para identificação de novos padrões de software. Pouca atenção é dada para o estudo de maneiras eficientes de empregar (Henninger e Corrêa, 2007) e testar aplicações desenvolvidas utilizando padrões (Cagnin, 2005). Assim, este capítulo apresenta a proposta de uma abordagem para auxiliar o usuário a localizar e utilizar com maior facilidade os padrões de software existentes na literatura, além de empregar os conceitos da abordagem ARTe nas atividades de VV&T.

Na Seção 5.2, são apresentadas considerações a respeito do emprego de padrões de software. Na Seção 5.3, é resumida a abordagem ARTe, cujo objetivo é reduzir o tempo despendido nas atividades de VV&T associando requisitos de teste a padrões de software. Na Seção 5.4, é apresentada a proposta de um Método Para Desenvolvimento Utilizando

Padrões de Software, criado a partir do levantamento do fluxo de execução de projetos criados no ambiente Peônia e com o objetivo de auxiliar usuários na aplicação de padrões de software durante um projeto. Por fim, na Seção 5.5, são apresentadas as considerações finais sobre este capítulo.

5.2 Emprego de Padrões de Software em Projetos

Como mencionado na Seção 2.4, de acordo com a maneira como são utilizados, padrões de software podem trazer vantagens ou desvantagens. Além disso, como há uma grande quantidade de padrões de software na literatura, é necessário atentar para sua qualidade. Uma das maneiras de selecionar bons padrões é verificar se possuem os elementos obrigatórios e se foram utilizados em pelo menos três aplicações. Outra forma do usuário selecionar e verificar a qualidade de um padrão é adicionar uma seção especial descrevendo como proceder para validar não somente o padrão como também as aplicações criadas a partir dele, conforme sugerido por Cagnin et al. (2005).

Padrões de software têm sido divulgados pela comunidade e adotados por desenvolvedores, no entanto, apesar de ser um problema antigo, pouca atenção tem sido dada para a aplicação de linguagem de padrões em projetos sistemáticos. Atualmente, a maioria dos trabalhos relacionados ao assunto é referente à identificação e descrição de padrões de software, assim como coleções e linguagens de padrões (Henninger e Corrêa, 2007).

Para usuários inexperientes é difícil identificar durante o desenvolvimento de software o momento certo para utilizar um padrão. Além disso, o crescimento da quantidade disponível de padrões de software dificulta a memorização e emprego das soluções. Como citado na Seção 3.4, ferramentas auxiliam engenheiros de software de diversas maneiras, como por exemplo, para apoiar a implementação, modelagem de diagramas, controle de versão, consulta a padrões e teste de software. Apesar das variadas formas de auxílio, não foram encontrados na literatura ambientes e ferramentas que apoiassem a utilização de padrões durante as diversas etapas de um processo de desenvolvimento, além de apoiar a associação de diretrizes de teste para facilitar a verificação, validação e teste das soluções reusadas.

Como citado na Seção 2.3, a qualidade do processo de desenvolvimento de software afeta a qualidade do software produzido. Processos de desenvolvimento devem ser escolhidos e adaptados de acordo com as necessidades do projeto e da empresa (Bertollo et al., 2006). Assim como processos de desenvolvimento, padrões de software são utilizados com o objetivo de melhorar a qualidade de um software e esses dois conceitos podem ser empregados em conjunto para atingir tal objetivo. De acordo com as suas características

e classificação, padrões de software podem ser distribuídos e associados às etapas de um processo de desenvolvimento facilitando sua identificação e sua utilização em projetos.

5.3 Verificação, Validação e Teste de Padrões de Software

As atividades de VV&T são algumas das mais onerosas no desenvolvimento de software (Rocha et al., 2001). A associação de requisitos de teste a padrões de software pode auxiliar engenheiros de software na verificação das soluções propostas e reduzir o tempo despendido na atividade de VV&T (Cagnin et al., 2005).

Cagnin et al. (2005) propõem uma abordagem, denominada ARTe (Abordagem de Reúso de Teste), que agrega recursos de teste a padrões de software facilitando as atividades de VV&T. A abordagem ARTe captura aspectos de validação embutidas em padrões de análise que modelam sistemas de informação de arquiteturas em três camadas: interface com o usuário, regras de negócio e armazenamento em banco de dados relacional (Cagnin, 2005).

Na Figura 5.1 são ilustradas as duas etapas que compõem a abordagem ARTe. A Etapa 1 define uma estratégia de apoio à definição de recursos de teste para cada padrão de uma linguagem de padrões de análise, resultando em uma linguagem de padrões com recursos de teste associados. Os recursos são compostos por requisitos de teste elaborados considerando os critérios de teste funcional Particionamento de Equivalência e Análise do Valor Limite e por casos de teste criados a partir desses requisitos de teste. A Etapa 2 contém diretrizes que apóiam o reúso das classes de equivalência, dos requisitos e casos de teste criados na primeira etapa, resultando em um conjunto de casos de teste para o sistema desenvolvido (Cagnin, 2005).

Com relação aos critérios de teste que devem ser utilizados, a escolha depende de cada tipo de requisito de teste, identificados no Passo 1 da Etapa 1, e dos aspectos a serem considerados no teste. Quando a especificação do produto é considerada para derivar os requisitos de teste, critérios de teste funcional devem ser empregados, por exemplo, quando são avaliados padrões de análise. Se a implementação do produto for considerada, critérios de teste estrutural devem ser utilizados, como por exemplo, para criar requisitos de teste para padrões de projeto ou de implementação (Cagnin, 2005).

Como pode ser observado nos passos da Etapa 1, não há restrição quanto ao critério de teste a ser empregado. No entanto, a abordagem ARTe propõe a utilização dos critérios de teste funcional Particionamento de Equivalência e Análise do Valor Limite por ter sido aplicada no contexto de linguagens de padrões de análise.

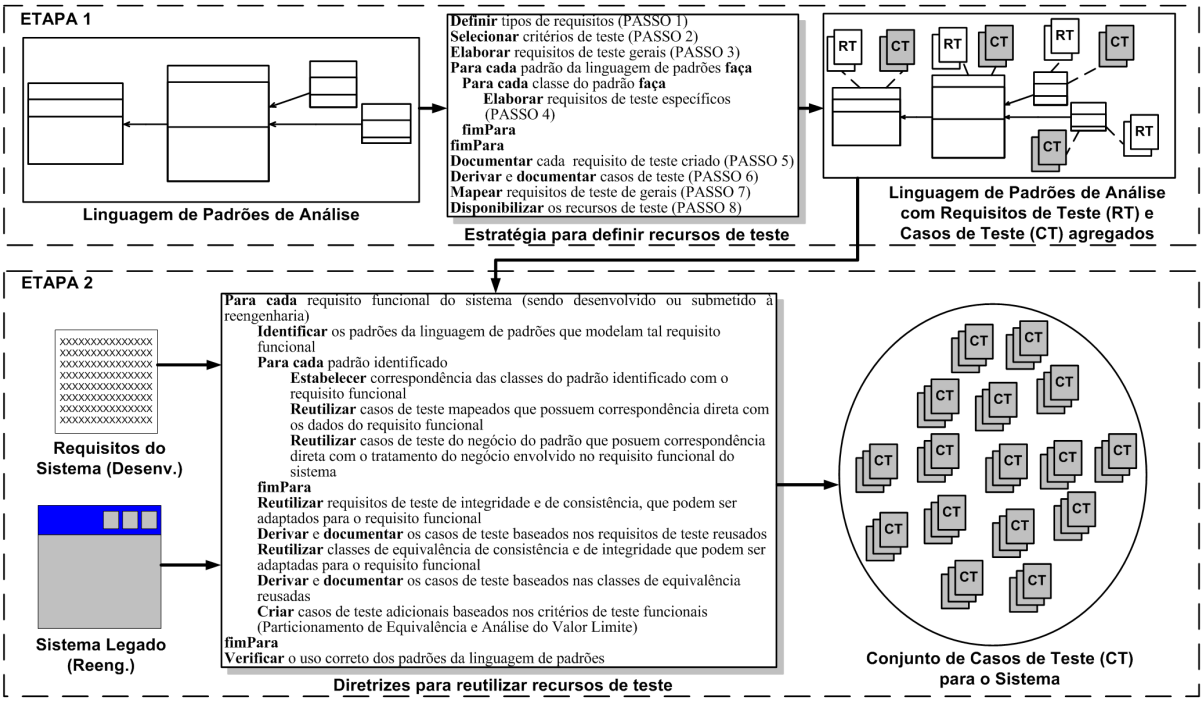

Figura 5.1: Visão Geral da Abordagem ARTe. (Cagnin, 2005)

Na Etapa 1 da abordagem ARTe podem ser ressaltados os Passos 5 e 6, pois destacam a importância da documentação dos requisitos e casos de teste gerados, e o Passo 8, que propõe a criação de uma seção no padrão para disponibilizar informações de VV&T.

Com a abordagem ARTe é possível minimizar esforços e custos despendidos nas atividades de VV&T, no desenvolvimento e reengenharia de software baseados em linguagem de padrões de análise. Além disso, também colabora com a prática ágil do teste antes da implementação, fornecendo recursos de teste à especificação do software por meio dos padrões de software (Cagnin, 2005).

5.4 Método Para Desenvolvimento Utilizando Padrões de Software

Com o objetivo de auxiliar desenvolvedores no emprego de padrões de software e requisitos de teste, foi desenvolvido o ambiente Peônia, descrito nos Capítulos 6 e 7. Durante o desenvolvimento e teste do ambiente Peônia foi possível observar a repetição do fluxo de execução no desenvolvimento de projetos apoiado por padrões de software, motivando a elaboração do Método Para Desenvolvimento Utilizando Padrões de Software. Por meio do emprego desse método, espera-se que os usuários utilizem com maior facilidade o ambiente Peônia e apliquem mais padrões de software em projetos.

Apesar de ter sido concebido para facilitar a utilização do ambiente Peônia, o método proposto concentra-se em dar diretrizes para o emprego de padrões de software durante todas as etapas de um desenvolvimento, independentemente de ser apoiado ou não pelo ambiente. No entanto, é importante ressaltar que o método proposto foi elaborado em cima de projetos criados utilizando o ambiente Peônia e, por causa das restrições de cronograma, não foi possível realizar medições controladas sobre sua eficiência e eficácia. Assim, em trabalhos futuros, planeja-se realizar essas medições e validações por meio de estudos de casos controlados experimentalmente para poder confirmar as vantagens de se utilizar o método no desenvolvimento apoiado por padrões de software, independentemente da utilização do ambiente Peônia.

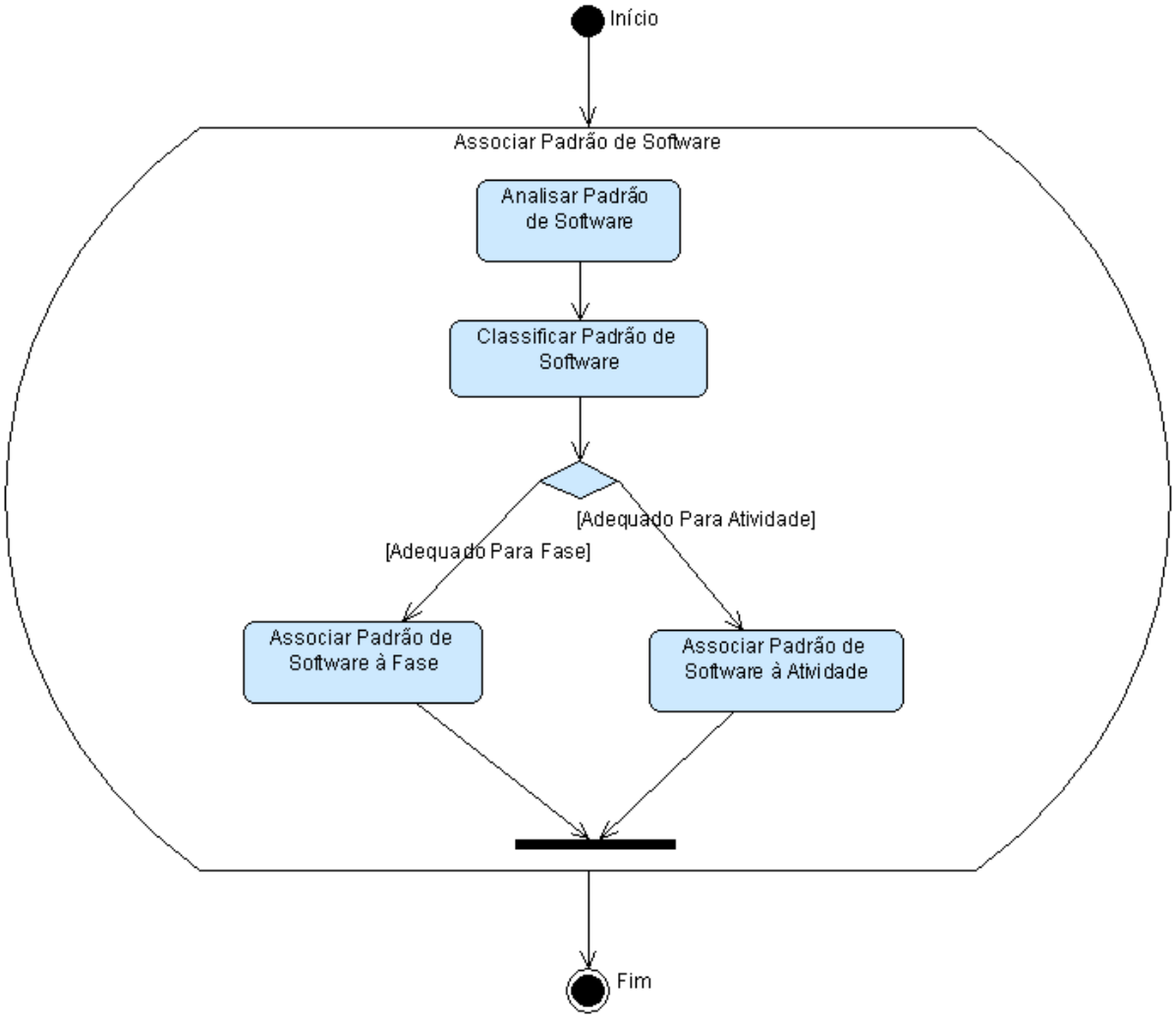

Figura 5.2: Associação de Fases e Atividades a Padrões de Software.

Inspirado na classificação segundo o estágio de desenvolvimento de software, mencionado na Seção 2.4.2.1, o Método Para Desenvolvimento Utilizando Padrões de Software propõe que padrões de software sejam associados às etapas de um processo de desenvol-

vimento para facilitar a utilização durante a execução de um projeto e estimular usuários a empregá-los. As atividades envolvidas nessa associação são exibidas na Figura 5.2.

Em geral, usuários buscam padrões de software apenas quando se deparam com um problema e não encontram facilmente uma solução. Ao associá-los às fases e atividades de um processo de desenvolvimento o usuário é induzido a averiguar, antes mesmo de encontrar um problema, a necessidade de utilizar esses padrões de software nas fases e em cada uma de suas atividades e, conseqüentemente, nos artefatos que são produzidos. Dessa maneira, o usuário é incentivado desde o início do projeto a utilizar as soluções propostas pelos padrões, melhorando a qualidade do software a ser produzido. Além disso, ao visualizar os padrões de software associados, mesmo que não os utilize, quando encontrarem um problema, a solução provavelmente será localizada com maior facilidade.

Seguindo o mesmo princípio, associando requisitos de teste a padrões de software o usuário pode testar a solução empregada com maior facilidade, principalmente porque muitas vezes o usuário não é o autor do padrão, desconhecendo a maneira mais apropriada de verificar, validar ou testar uma solução. Essa associação auxilia engenheiros de software a averiguar as soluções propostas e pode reduzir o tempo despendido nas atividades de VV&T (Cagnin et al., 2005). Outros detalhes sobre o teste de padrões de software podem ser encontrados na Seção 5.3.

Os elementos fundamentais do RUP (Kruchten, 2000) são utilizados como a base da estrutura dos processos de desenvolvimento empregada na definição das atividades executadas pelo método proposto. O RUP é um processo de desenvolvimento que provê uma abordagem disciplinada para determinar tarefas e responsabilidades em um projeto. Além disso, fornece um arcabouço de processo extensível e adaptável às necessidades dos usuários (Cagnin, 2005). Exemplos de processos baseados no RUP são o PARFAIT (Cagnin, 2005) e o processo proposto por Conallen (2002) utilizando o WAE. Neste trabalho, apenas os elementos fundamentais do RUP são empregados, pois a intenção é fornecer a base essencial para a execução de um processo de desenvolvimento e, ao mesmo tempo, dar a liberdade para o usuário criar a estrutura do processo a ser empregado.

De acordo com o RUP, um processo de desenvolvimento é composto por *fases* que, por sua vez, são compostas de *atividades*. A execução de uma atividade é atribuída a uma pessoa que desempenha um *papel* no desenvolvimento do produto. Uma *atividade* descreve os procedimentos de como o trabalho deve ser realizado e pode receber artefatos de entrada e produzir artefatos de saída. Um *artefato* é um produto de trabalho gerado com a execução de uma atividade, por exemplo, modelos, elementos de modelo, código-fonte e documentos (Conallen, 2002).

Na Figura 5.3, é exibida uma visão geral do método proposto. Basicamente, são ilustradas as atividades que envolvem a preparação e a execução de um projeto, res-

pectivamente nas cores rosa e lilás. Antes de iniciar um projeto, é necessário avaliá-lo (*Atividade: Avaliar Projeto*), por exemplo, realizando a análise do domínio e cronograma. A partir das informações levantadas, um processo de desenvolvimento deve ser escolhido (*Atividade: Escolher Processo de Desenvolvimento*) para organizar, estruturar e controlar a execução do projeto. O processo de desenvolvimento escolhido pode conter elementos desnecessários ou não compatíveis com o escopo do projeto, tornando necessário selecionar o que deve ser excluído do processo e o que deve ser adicionado para que o projeto seja executado com sucesso (*Atividade: Adaptar Processo de Desenvolvimento ao Contexto do Projeto*).

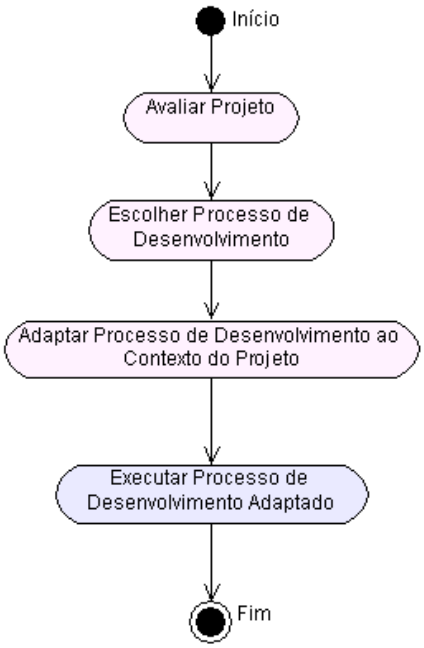

Figura 5.3: Visão Geral do Método Para Desenvolvimento Utilizando Padrões de Software.

Após escolher e adaptar um processo de desenvolvimento, o projeto é iniciado (*Atividade: Executar Processo de Desenvolvimento Adaptado*) por meio da realização de cada uma das suas fases. A execução de um processo de desenvolvimento é ilustrado na Figura 5.4. Antes de cada fase, verifica-se a existência de fases que ainda não tenham sido cumpridas para que sejam averiguadas quanto à necessidade de executá-las (*Ação: Verificar Existência de Fase Não Executada*).

Dois tipos de fases podem existir dentro de um processo: obrigatórias e opcionais. Caso existam fases obrigatórias não cumpridas, elas devem ser executadas (*Atividade: Executar Fase*), pois impedem que um processo seja finalizado com sucesso e produzindo a aplicação com a qualidade desejada. Se fases opcionais existirem, deve-se avaliar a necessidade

de cumpri-las (*Ação: Avaliar Necessidade de Execução*), pois não são essenciais para a conclusão do projeto.

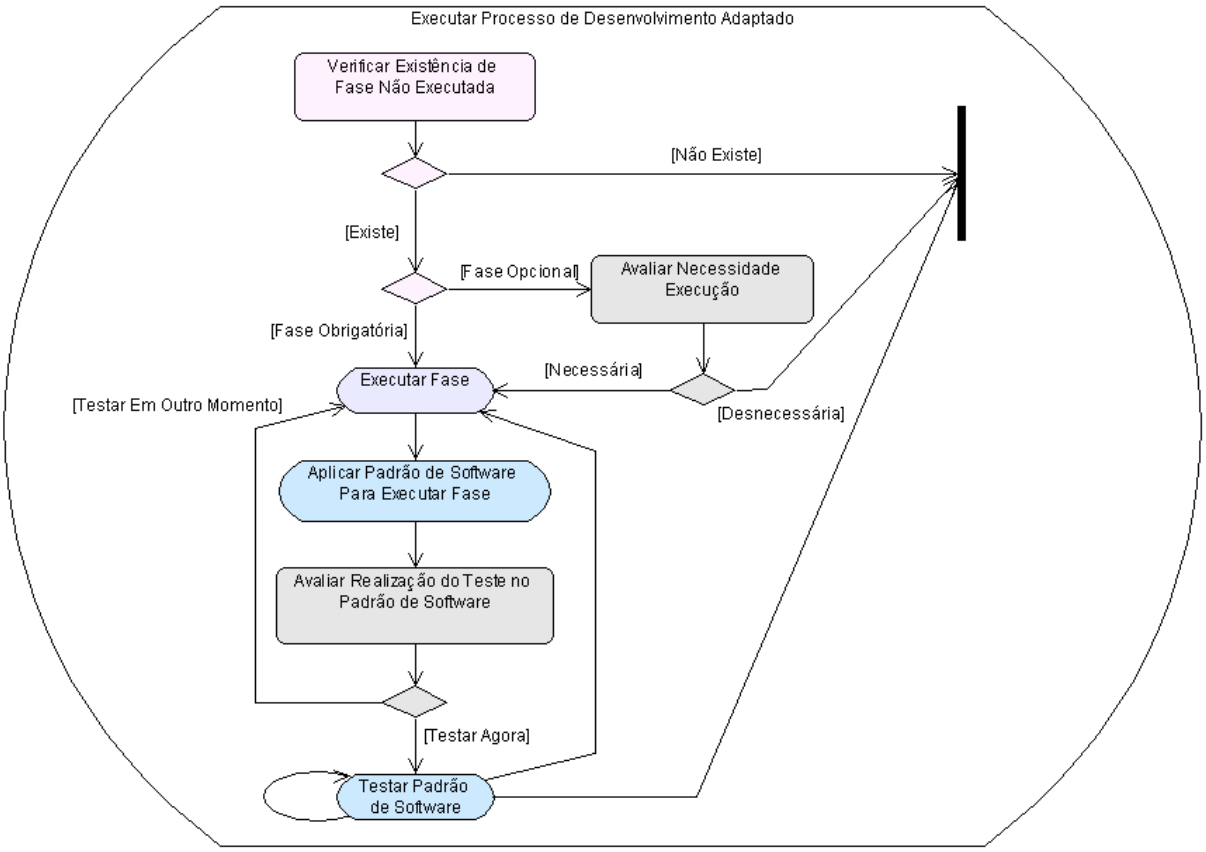

Figura 5.4: Diagrama de Atividades de *Executar Processo de Desenvolvimento Adaptado.*

Como pode ser observado na Figura 5.5, uma fase é cumprida por meio da execução de todas as atividades obrigatórias que a compõem. Portanto, enquanto uma fase é executada buscam-se as atividades que ainda não foram executadas (*Ação: Verificar Existência de Atividade Não Executada*). Caso sejam obrigatórias, elas devem ser cumpridas (*Atividade: Executar Atividade*). Se forem atividades opcionais, devem ser averiguados os impactos de executá-las (*Ação: Avaliar Necessidade de Execução*).

Por fim, uma atividade é executada quando os artefatos que devem ser produzidos são criados com sucesso, como pode ser observado na Figura 5.6. Enquanto uma atividade é executada, é realizada a análise dos artefatos ainda não criados (*Ação: Verificar Existência de Artefatos Não Criados*). Caso sejam obrigatórios, eles devem ser criados (*Atividade: Criar Artefato*) e, se forem opcionais, deve-se averiguar a necessidade de criá-los (*Ação: Avaliar Necessidade de Criação*). Após a conclusão de todas as fases e atividades obrigatórias e a produção dos artefatos obrigatórios, o processo pode ser finalizado, pois o desenvolvimento da aplicação é considerado como concluído.

Por meio da produção dos artefatos, o projeto evolui incrementalmente até chegar à aplicação desejada. Esses artefatos podem ser construídos utilizando padrões de software instanciados, permitindo a alteração e adaptação ao contexto do projeto. Padrões de software também podem ser utilizados para apoiar a execução de fases e atividades. Durante a execução de processo de desenvolvimento, fase e atividade, respectivamente, as Figuras 5.4, 5.5 e 5.6, citadas anteriormente, as caixinhas azuis representam ações e atividades ligadas ao emprego de padrões de software em projetos.

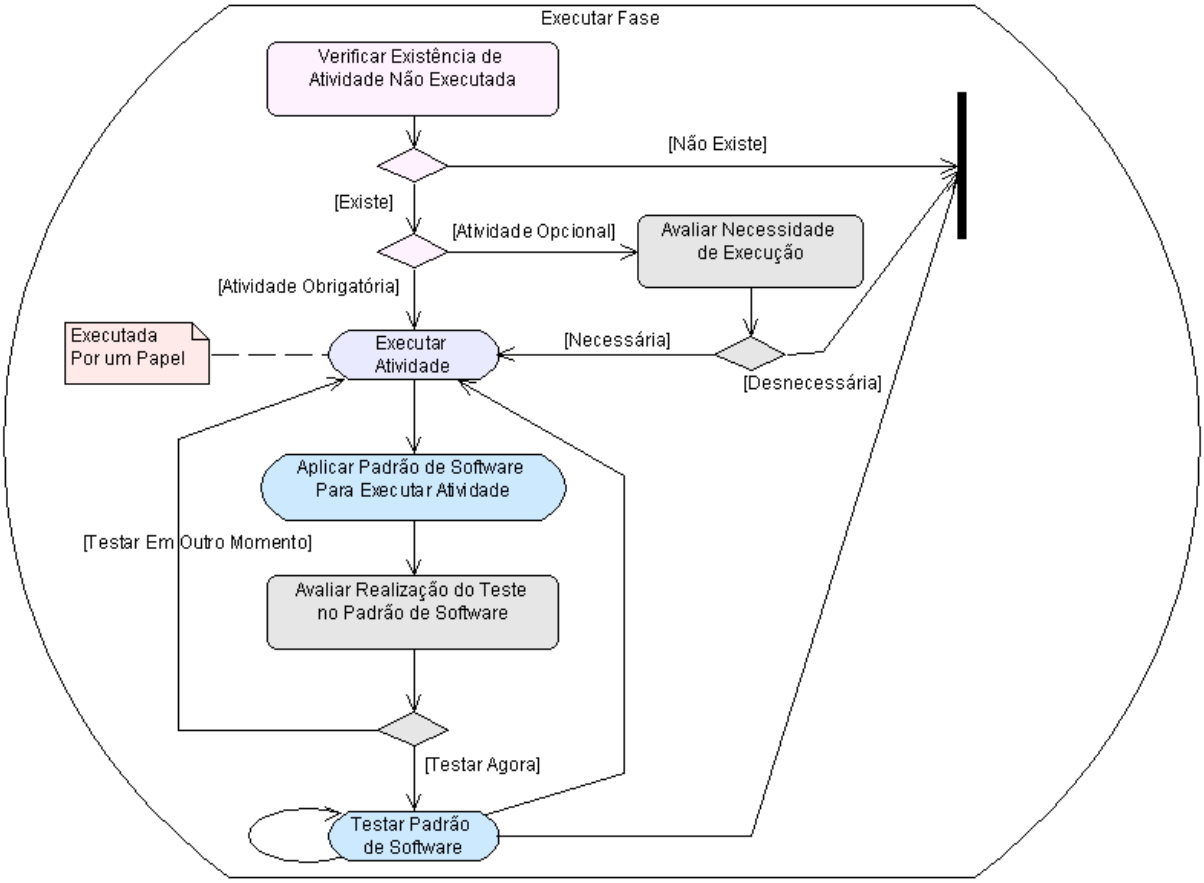

Figura 5.5: Diagrama de Atividades de *Executar Fase*.

Na Figura 5.7 é apresentada a aplicação de padrão durante a execução de uma fase (*Atividade: Aplicar Padrão de Software Para Executar Fase*) e na Figura 5.8 são ilustradas as atividades referentes ao uso de um padrão durante o desenvolvimento de uma atividade ou artefato, pois ambas possuem o mesmo fluxo de execução (*Atividades: Aplicar Padrão de Software Para Executar Atividade* e *Aplicar Padrão de Software Para Criar Artefato*). A diferença entre as duas figuras está na verificação de padrão associado a uma atividade (*Ação: Verificar Existência de Padrão de Software Associado à Atividade*), que não é efetuada ao empregar padrões na execução de uma fase (*Atividade: Aplicar Padrão de Software Para Executar Fase*). Isso ocorre porque, apesar de uma fase ser composta de

atividades, padrões de software associados a uma atividade podem ser específicos demais para a fase. Assim, os padrões associados a uma atividade não são sempre aplicáveis à fase, sendo necessário avaliá-los separadamente. Caso sejam aplicáveis a fase e estejam associados apenas à atividade, no método proposto, é considerado como uso de um padrão não associado à fase.

O emprego de padrões de software é opcional e depende da avaliação do usuário (*Ação: Avaliar Intenção de Aplicar Padrão de Software*). Caso opte por utilizar padrões de software no seu projeto, o usuário deve analisar e escolher a melhor solução para o seu problema. Como mencionado anteriormente neste capítulo, é difícil lembrar e escolher entre as diversas opções existentes atualmente na literatura. O método aqui proposto sugere que padrões de software sejam associados previamente às fases e atividades do processo de desenvolvimento executado, facilitando a visualização dos padrões mais adequados em cada etapa de um projeto.

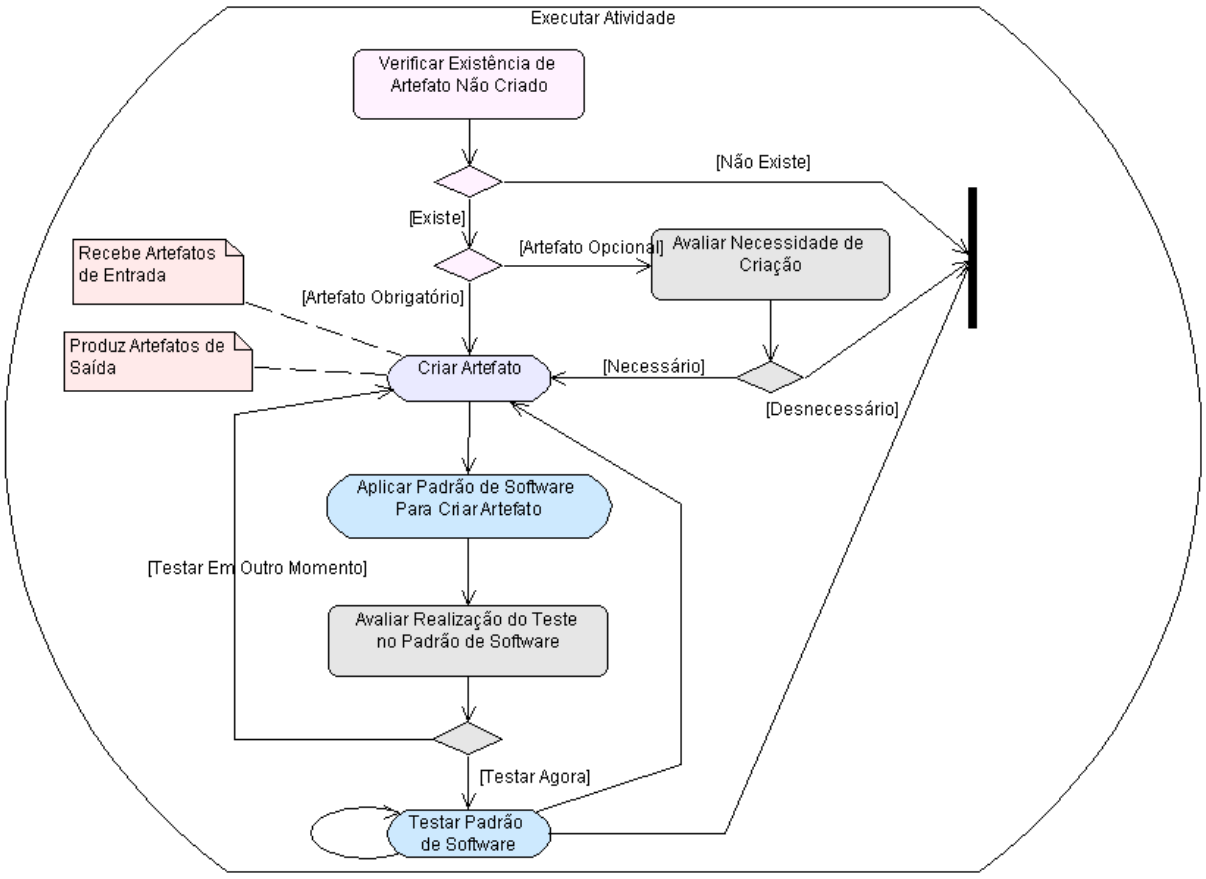

Figura 5.6: Diagrama de Atividades de *Executar Atividade*.

Portanto, caso existam padrões associados à fase (*Ação: Verificar Existência de Padrão de Software Associado à Fase*), durante a realização de uma fase (*Atividade: Executar Fase*) o usuário pode utilizá-los para auxiliar o desenvolvimento do projeto. Padrões de Software associados a uma fase também estão indiretamente associados a uma atividade.

Um artefato não possui padrões associados diretamente, pois a sua criação é realizada dentro de uma atividade, o que implica que todos os padrões associados a uma fase ou atividade também são sugestões para auxiliar a criação de artefatos. Assim, o usuário tem a opção de procurar por padrões associados a fases e atividades (*Ações: Verificar Existência de Padrão de Software Associado à Fase* e *Verificar Existência de Padrão de Software Associado à Atividade*) para aplicar padrões de software durante realização de uma atividade (*Atividade: Executar Atividade*) e o desenvolvimento de um artefato (*Atividade: Criar Artefato*).

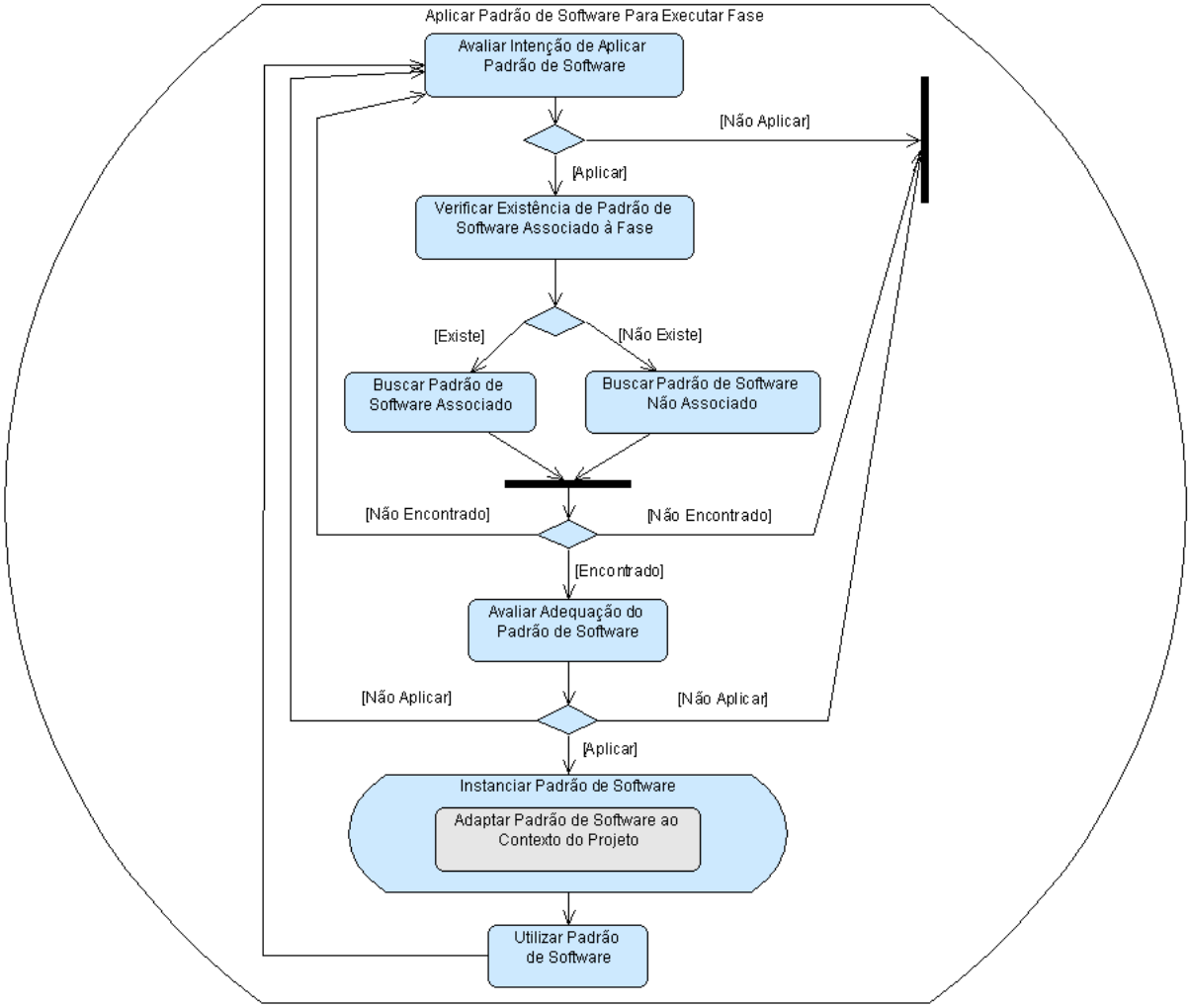

Figura 5.7: Diagrama de Atividades de *Aplicar Padrão de Software Para Executar Fase*.

Destaca-se que, apesar de propor a associação, não é descartada a possibilidade de empregar padrões de software não associados. Assim, é proposta realização de buscas pela solução mais adequada a ser empregada, tanto em padrões de software associados às etapas do processo de desenvolvimento (*Ação: Buscar Padrão de Software Associado*), quanto em outros padrões da literatura (*Ação: Buscar Padrão de Software Associado*).

Caso seja encontrado um padrão na pesquisa realizada, deve ser realizada uma avaliação para verificar se a solução proposta é adequada para o problema do usuário (*Ação: Avaliar Adequação do Padrão de Software*).

Se o usuário concluir que o padrão de software é o mais adequado a ser empregado, o padrão de software deve ser instanciado (*Atividade: Instanciar Padrão de Software*). A instanciação de um padrão de software é realizada por meio da adaptação da solução para o contexto do projeto (*Ação: Adaptar Padrão de Software ao Contexto do Projeto*). Finalmente, o padrão instanciado é utilizado na execução de fases e atividades, ou na criação de artefatos (*Ação: Utilizar Padrão de Software*).

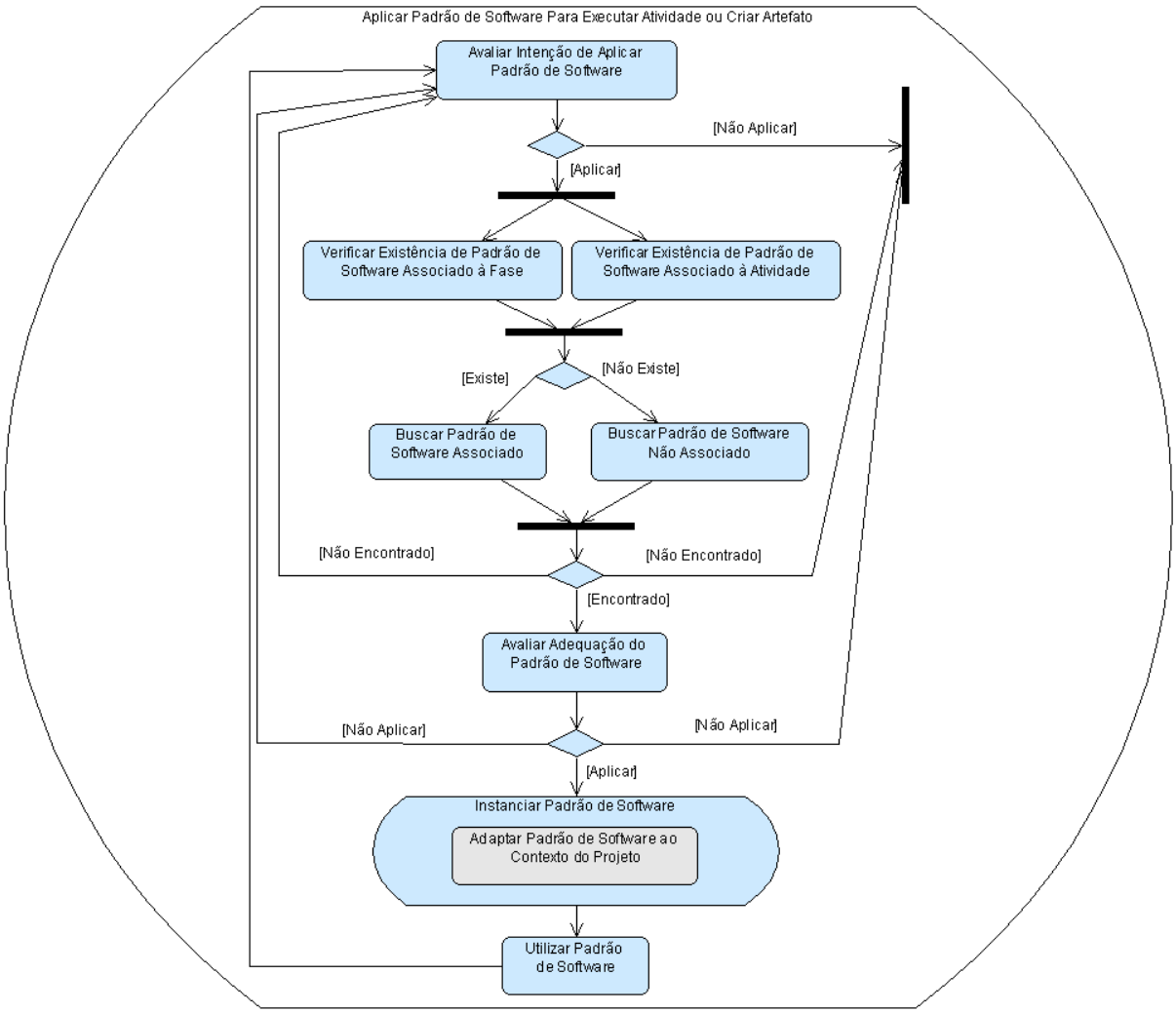

Figura 5.8: Diagrama de Atividades de *Aplicar Padrão de Software Para Executar Atividade* e *Aplicar Padrão de Software Para Criar Artefato*.

O momento e a maneira como o teste das soluções são realizados depende do projeto e é responsabilidade do usuário avaliar e definir quando isso deve acontecer (*Ação: Avaliar Realização do Teste do Padrão de Software*). Ao optar pela execução das atividades de

VV&T, o padrão de software utilizado no projeto deve ser validado (*Atividade: Testar Padrão de Software*), tarefa que envolve a seleção de requisitos de teste. Como pode ser observado na Figura 5.9, no teste é verificada a existência de requisitos de teste associados ao padrão de software (*Ação: Verificar Existência de Requisito de Teste Associado*) e, caso esses existam, são selecionados os requisitos (*Ação: Escolher Requisito de Teste Associado*) utilizados na atividade de VV&T. Caso o usuário não queira empregar os requisitos associados, ou queira complementar o teste, também podem ser escolhidos requisitos de teste não associados (*Ação: Escolher Requisito de Teste Não Associado*). Após a escolha dos requisitos de teste, devem ser gerados casos de testes (*Ação: Criar Caso de Teste*) utilizados no teste do padrão de software (*Ação: Validar Padrão de Software*).

Assim, como pode ser observado nas Figuras 5.4, 5.5 e 5.6, o Método Para Desenvolvimento Utilizando Padrões de Software facilita a localização de padrões com a associação prévia às etapas de um processo de desenvolvimento, ao mesmo tempo que lembra ao usuário a possibilidade de aplicá-los na execução das fases e atividades e na criação de artefatos.

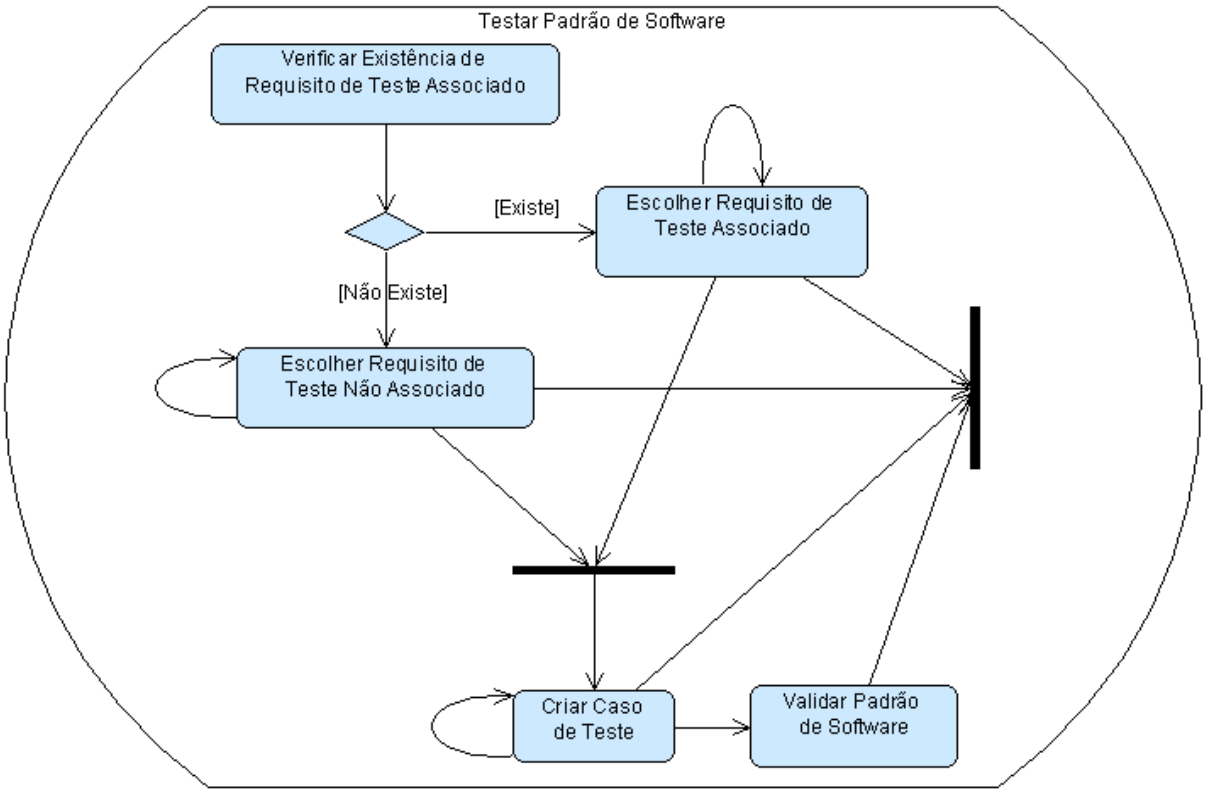

Figura 5.9: Diagrama de Atividades de *Testar Padrão de Software*.

5.5 Considerações Finais

Neste capítulo apresentou-se uma visão geral sobre o emprego e teste de padrões de software. Apesar da grande aceitação existente quanto ao emprego de padrões de software, poucos estudos são voltados para encontrar maneiras de tornar a utilização mais fácil e intuitiva para os usuários. Apesar de ambientes e ferramentas serem encontradas na literatura, há uma carência por ambientes e ferramentas que orientem a utilização de padrões durante as etapas de um processo de desenvolvimento, ou que apóiem a associação de diretrizes de teste para agilizar as atividades de VV&T.

Com a proposta de associação de padrões de software a requisitos de teste, a abordagem ARTe tem o objetivo de minimizar esforços e custos despendidos nas atividades de VV&T, no desenvolvimento e reengenharia de softwares quando utilizadas linguagem de padrões de análise. Requistos de teste associados aos padrões de software também colaboram com a prática ágil do teste antes da implementação.

Da mesma maneira que padrões de software podem ser associados a requisitos de teste, etapas de um processo de desenvolvimento também podem ser associadas a padrões de software. Assim, com a intenção de tornar mais fácil o emprego de padrões de software no desenvolvimento de projetos, também foi apresentado neste capítulo o Método Para Desenvolvimento Utilizando Padrões de Software. Esse método propõe a associação de padrões de software às fases e atividades com o objetivo de estimular a sua utilização. Ao associar padrões de software às etapas de um processo de desenvolvimento o usuário é induzido a averiguar, antes mesmo de encontrar um problema, a necessidade de utilizar esses padrões de software no projeto. O Método Para Desenvolvimento Utilizando Padrões de Software também estimula a associação de requisitos de teste a padrões de software, facilitando e agilizando as atividades de VV&T.

No próximo capítulo, o ambiente Peônia, a partir do qual foram feitas as observações para criar o Método Para Desenvolvimento Utilizando Padrões de Software, é descrito quanto a sua arquitetura, modelagem e implementação.

Capítulo

6

Ambiente Peônia - Funcionalidades

6.1 Considerações Iniciais

Como mencionado nos capítulos anteriores, para garantir a qualidade de software, processos de desenvolvimento, padrões de software e requisitos de teste são empregados em conjunto, para controlar e auxiliar projetos.

Por causa da quantidade de padrões de software existente e a complexidade de suas soluções, usuários possuem dificuldade para empregá-los. Além disso, a validação das soluções utilizadas não é uma tarefa trivial, principalmente porque na maior parte das vezes o usuário não é o autor do padrão de software.

Assim, com o objetivo de apoiar a consulta, aplicação e documentação de padrões de software durante cada uma das etapas de um processo de desenvolvimento de software, além de informar os requisitos de teste necessários para validar a solução utilizada, foi desenvolvido o ambiente Peônia, cujas funcionalidades são detalhadas neste capítulo. O ambiente Peônia, cuja página inicial é ilustrada na Figura 6.1, está disponível no endereço: `http://www.labes.icmc.usp.br:8180/alechan/`.

Peônia é o nome dado a uma flor originária da China. Assim como a flor de Peônia significa timidez, representa riqueza e é um símbolo de abundância no seu país de origem, esperamos que, apesar de atualmente muitos desenvolvedores utilizarem timidamente a tecnologia de padrões de software, com o ambiente Peônia usuários possam empregar e recuperar padrões de maneira abundante, enriquecendo os seus projetos. A flor de Peônia

hoje é encontrada em diversas partes do mundo e, da mesma maneira, desejamos que o ambiente Peônia seja amplamente divulgado para ajudar uma grande quantidade de desenvolvedores.

Assim como o Método Para Desenvolvimento Utilizando Padrões de Software, mencionado na Seção 5.4, o ambiente Peônia propõe a associação de padrões de software às etapas de um processo de desenvolvimento, estimulando a sua utilização durante todo o ciclo de vida. A divisão entre as etapas foi inspirada na classificação segundo o estágio de desenvolvimento de software, explicada na Seção 2.4.2.1.

Na Seção 6.2, são explicadas as três fases do ambiente Peônia e listadas cada uma das suas funcionalidades. Por fim, na Seção 6.3, são apresentadas as considerações finais sobre este capítulo.

Figura 6.1: Tela da página inicial do ambiente Peônia.

6.2 Descrição das Funcionalidades

O apoio oferecido pelo ambiente Peônia para o desenvolvimento de aplicações pode ser dividido em três fases: *Alimentação do Peônia*, *Desenvolvimento do Projeto* e *Gerência do Peônia*. Cada uma das fases do ambiente Peônia são compostas de atividades. As três fases e suas 168 (cento e sessenta e oito) atividades são ilustradas na Figura 6.2.

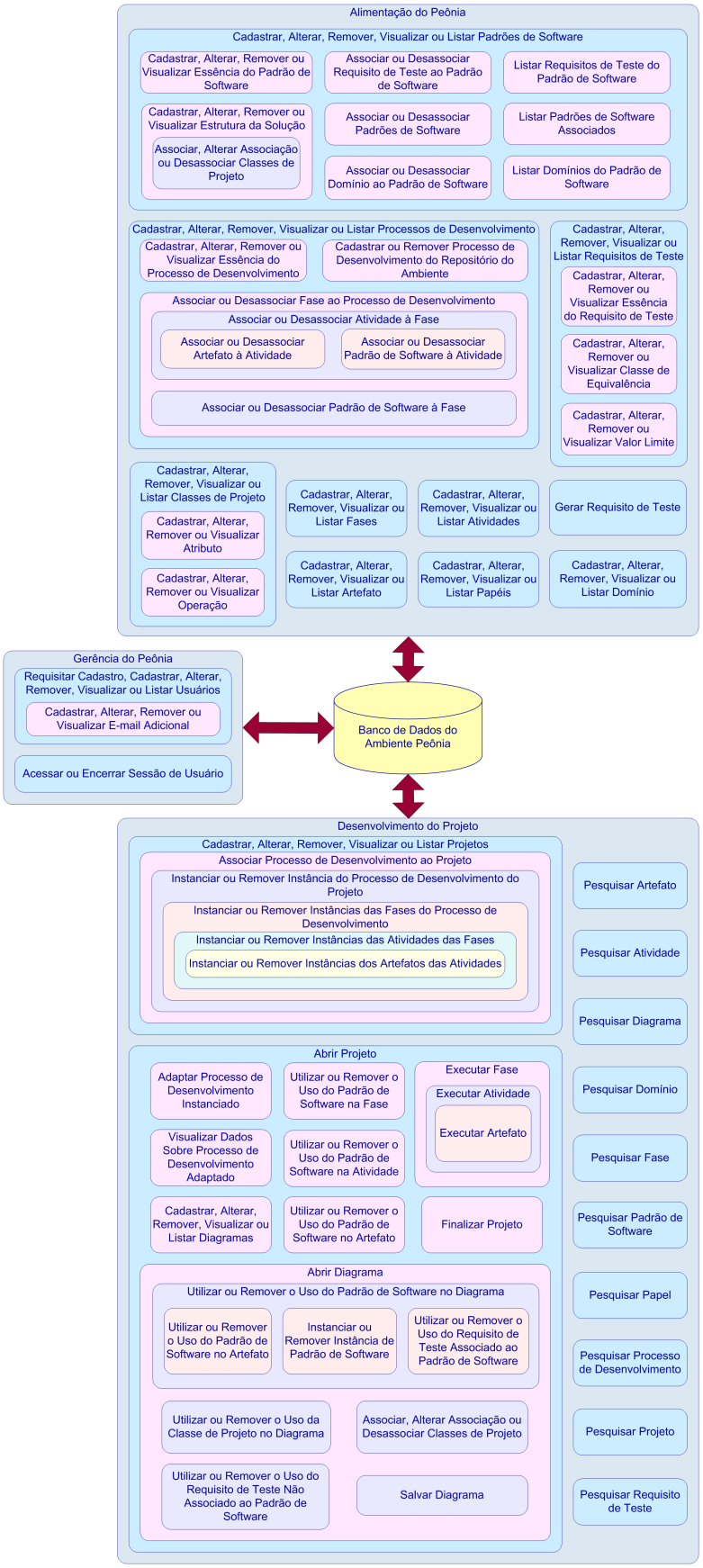

Figura 6.2: As três fases do ambiente Peônia.

A primeira fase (*Fase: Alimentação do Peônia*) é composta pelas atividades de cadastro e associação de processos de desenvolvimento, padrões de software e requisitos de teste, assim como todos os sub-conceitos que compõem esses elementos.

Na segunda fase (*Fase: Desenvolvimento do Projeto*), são realizadas as atividades de preparação e execução de um projeto, ilustradas no capítulo anterior na Figura 5.3. Durante a fase de *Desenvolvimento do Projeto*, um processo de desenvolvimento é instanciado e suas etapas e artefatos podem ser alterados quanto à obrigatoriedade para o projeto que está sendo desenvolvido.

Segundo Rocha et al. (2001) três etapas devem ser consideradas na definição de uma abordagem flexível de processos de desenvolvimento: definição de processos padrão para a organização, especialização dos processos padrão e instanciação para projetos específicos. O ambiente Peônia oferece apoio a essas três etapas.

Assim, a fase de *Alimentação do Peônia* permite que o usuário cadastre processos padrão definidos pelo usuário e a fase de *Desenvolvimento do Projeto* permite que o usuário instancie e utilize esses processos. Ambas as fases podem ser empregadas na especialização dos processos padrão, pois o processo padrão especializado pode ser cadastrado como um novo processo, ou pode ser alterado na instanciação ao ser adaptado para o contexto de um projeto.

Na terceira fase (*Fase: Gerência do Peônia*), são efetuadas operações relacionadas à manutenção de informações contidas no ambiente Peônia e ao controle e acesso de usuários. Todas as atividades são executadas nos elementos do ambiente Peônia. Os elementos cadastrados possuem campos obrigatórios e opcionais, além de algumas regras para manter a consistência das informações contidas no ambiente Peônia.

Nas seções seguintes, são apresentados detalhes sobre essas atividades, que são as funcionalidades oferecidas pelo ambiente para auxiliar usuários no desenvolvimento de aplicações. As atividades oferecidas pelas fases de *Gerência do Peônia*, *Alimentação do Peônia* e *Desenvolvimento do Projeto* são apresentadas, respectivamente, nas Seções 6.2.1, 6.2.2 e 6.2.3.

6.2.1 Fase: Gerência do Peônia

A administração de contas de usuários e controle de acesso ao ambiente Peônia são atividades relacionadas à fase de *Gerência do Peônia*. Para que o ambiente Peônia possa ser utilizado, o usuário precisa ter um cadastro e deve estar autenticado corretamente. Como pode ser observado no modelo conceitual do ambiente Peônia (Figura 7.2), existem dois tipos de usuários: o *Engenheiro de Software* e o *Administrador*.

O *Engenheiro de Software* tem permissão para executar as funcionalidades de: acesso e encerramento de sessão do usuário; cadastro, gerência, instanciação e acompanhamento da

execução de processos de desenvolvimento; criação e gerência de projetos; criação, gerência e instanciação de padrões de software; criação e gerência de requisitos de teste; criação e gerência de diagramas de classes e de todos os elementos necessários na construção desses diagramas; associação de etapas e artefatos a processos de desenvolvimento; associação de padrões de software a processo de desenvolvimento e requisitos de teste; e busca dos elementos cadastrados no ambiente.

Além de executar todas as funcionalidades disponibilizadas para um *Engenheiro de Software*, o *Administrador* possui também permissão para cadastrar, visualizar e gerenciar as contas dos usuários e é o único que pode alterar as informações de um processo cadastrado no "Repositório do Ambiente", que é descrito com mais detalhes na Seção 6.2.2.7.

Na Seção 6.2.1.1 são apresentadas as atividades relacionadas ao controle de usuários do ambiente. Na Seção 6.2.1.2 são apresentadas as atividades de acesso e encerramento de sessão do usuário.

6.2.1.1 Atividades: Requisitar Cadastro, Cadastrar, Alterar, Remover, Visualizar ou Listar Usuários

O ambiente Peônia só pode ser utilizado por usuários cadastrados. Um usuário pode ser cadastrado de duas maneiras: pelo envio de uma requisição realizada por visitantes, ou pelo cadastro direto feito por administradores do ambiente Peônia.

Na primeira maneira, visitantes do ambiente Peônia enviam seus dados pessoais informando o desejo de ter uma conta (*Atividade: Requisitar Cadastro de Usuário*). A tela de envio de dados é apresentada na Figura 6.3. Em seguida, um administrador deve autorizar o acesso, classificando-o como um usuário do tipo *Administrador* ou *Engenheiro de Software*. Após a aprovação, o usuário é considerado cadastrado e pode utilizar as funcionalidades do ambiente Peônia.

Atualmente, não há um mecanismo automatizado de autorização ou que avise o administrador quando um usuário enviou uma requisição. Assim é necessário que o usuário aguarde até que o *Administrador* autorize o cadastro.

Administradores também podem cadastrar diretamente os dados pessoais e autorizar o acesso de um usuário (*Atividade: Cadastrar Usuário*). Após o cadastro, o usuário inserido pode alterar os seus dados pessoais (*Atividade: Alterar Cadastro*) e realizar a remoção do seu cadastro (*Atividade: Remover Cadastro*). Apesar de todos os usuários terem permissão para alterar o seu cadastro, não é permitido a usuários do tipo *Engenheiro de Software* modificar a sua classificação para *Administrador*, pois trata-se de uma tarefa exclusiva de administradores.

Figura 6.3: Tela da atividade *Requisitar Cadastro de Usuário*.

Além da exclusividade para executar o cadastro direto, administradores também podem alterar (*Atividade: Alterar Usuário*) e remover (*Atividade: Remover Usuário*) usuários. A verificação dos dados de um usuário (*Atividade: Visualizar Usuário*), assim como a escolha da pessoa a ser alterada ou removida, é realizada a partir de uma lista (*Atividade: Listar Usuários*) que é apresentada assim que a aba do elemento *Usuário* é selecionada.

É por meio da atividade de alteração que é realizada a aprovação de um usuário. Essa atividade é ilustrada na Figura 6.4. A seta "A" vinho indica o campo "Tipo de Permissão", que deve ser modificado da situação *Pendente* para um dos tipos de usuário do ambiente Peônia.

O cadastro de um usuário pode ser removido pelo administrador ou pelo próprio dono do cadastro. Na Figura 6.5, é ilustrada a tela exibida ao administrador quando um usuário é selecionado para remoção e, na Figura 6.6, é apresentada a tela exibida ao usuário quando requisita a sua auto remoção.

A conta e os dados do cadastro do usuário não são excluídos do ambiente na atividade de remoção de usuário. Suas informações permanecem armazenadas e seu tipo de usuário é apenas alterado para *Removido*. Essa decisão foi tomada para evitar que padrões de software percam o histórico dos usuários que os cadastraram.

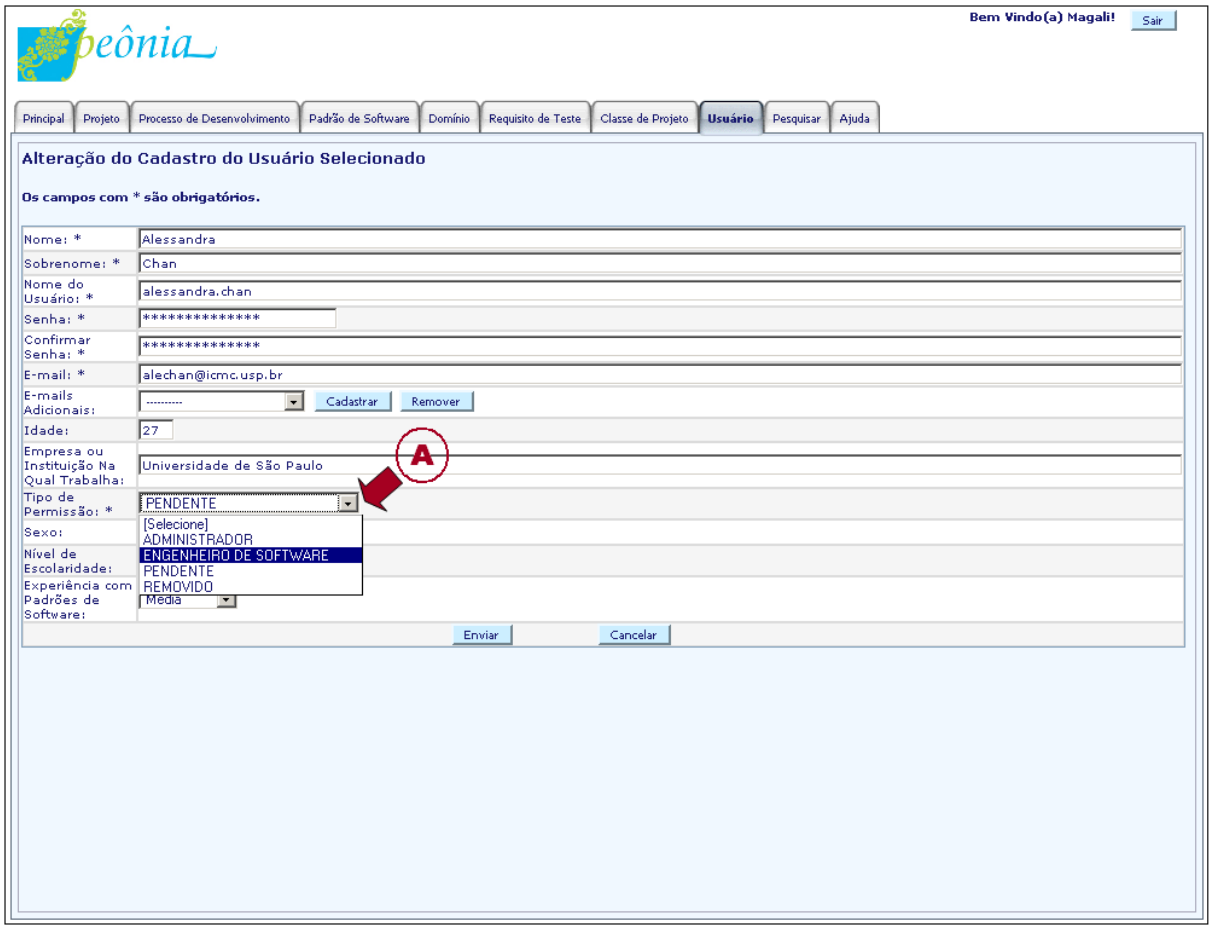

Figura 6.4: Tela da atividade *Alterar Cadastro*.

Um dos requisitos para o funcionamento correto do ambiente Peônia é a existência obrigatória de ao menos um *Administrador*, permitindo a realização de cadastros de outros usuários.

Com relação aos dados contidos em um cadastro de usuário, os dados pessoais que o usuário deve enviar se dividem em:

- Campos obrigatórios: “Nome”, “Sobrenome”, “Nome do Usuário”, “Senha”, “Confirmação de Senha” e “E-mail”;
- Campos opcionais: “E-mails Adicionais”, “Idade”, “Sexo”, “Nível de Escolaridade”, “Experiência com Padrões de Software” e “Empresa ou Instituição Na Qual Trabalha”.

O campo “Nome do Usuário” é único, sendo empregado pelo ambiente Peônia para identificar cada cadastro de usuário. Além dos campos de dados pessoais, também é armazenado o tipo de permissão, já citado anteriormente, e um código identificador numérico gerado automaticamente pelo *framework* Hibernate. Esse código é utilizado pelo

framework Hibernate para reconhecer e controlar cada entidade *Usuário* no banco de dados.

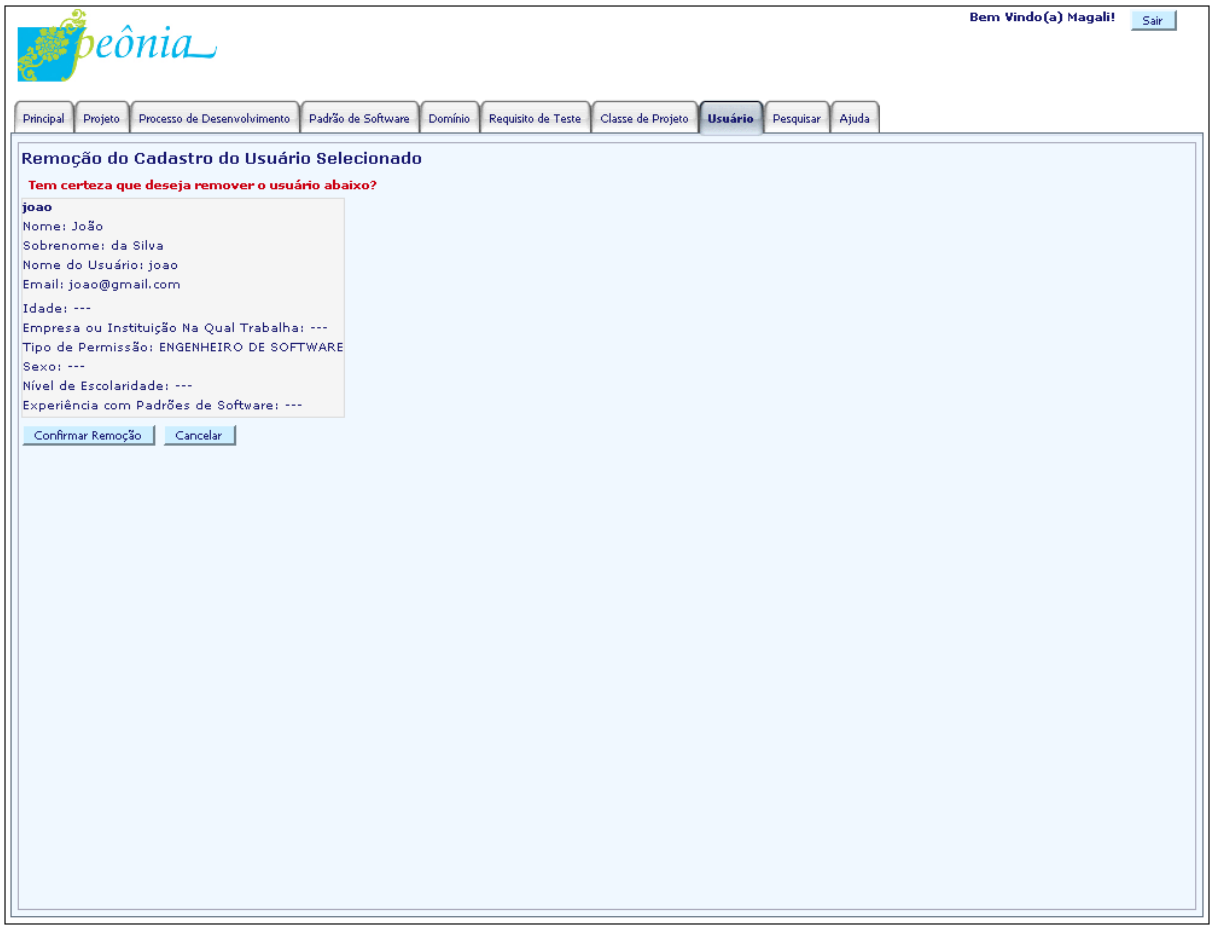

Figura 6.5: Tela exibida ao selecionar um outro usuário e requisitar a atividade *Remover Usuário*.

Com relação a gerência de e-mails adicionais (*Atividades: Cadastrar, Alterar, Remover* ou *Visualizar E-mail Adicional*), essa é realizada em um formulário separado quando requisitado pelo usuário durante o cadastro ou alteração de usuário. Não há limite para a quantidade cadastrada de e-mails adicionais.

As informações criadas pelos usuários são restritas a sua sessão, com exceção de padrões de software cadastrados e processos de desenvolvimento compartilhados no "Repositório do Ambiente". Mais informações sobre os repositórios de padrões e processos de desenvolvimento do ambiente e do usuário são apresentadas, respectivamente, nas Seções 6.2.2.5 e 6.2.2.7.

Figura 6.6: Tela exibida ao requisitar a atividade *Remover Usuário* para o usuário autenticado no ambiente.

6.2.1.2 Atividades: Acessar ou Encerrar Sessão do Usuário

Para utilizar as funcionalidades oferecidas pelo ambiente Peônia, um usuário precisa acessar a sua sessão (*Atividade: Acessar Sessão do Usuário*). Para isso, é necessário informar os campos "Nome do Usuário" e "Senha", que são dados pertencentes ao cadastro do usuário, como citado na Seção 6.7. Após o envio das informações, o ambiente Peônia realiza a autenticação dos dados, ou seja, verifica se os dados estão corretos, e, em caso afirmativo, exibe a página inicial da sessão do usuário, ilustrado na Figura 6.7. Caso alguma das informações esteja diferente das cadastradas no banco de dados, uma mensagem informando o erro é exibida para o usuário.

Após terminar de executar as funcionalidades do ambiente Peônia, o usuário deve requisitar o encerramento da sua sessão (*Atividade: Encerrar Sessão do Usuário*), evitando que danos sejam causados aos seus projetos e dados pessoais. O encerramento é executado ao clicar no botão "Sair", localizado no canto superior direito da tela, marcado na Figura 6.7 pelo círculo na cor vinho. Após a requisição da operação, o ambiente Peônia remove as informações armazenadas em memória sobre as tarefas executadas pelo usuário

autenticado e apresenta-lhe novamente a tela inicial, ilustrada anteriormente na Figura 6.1.

Figura 6.7: Tela da página inicial da sessão de um usuário do tipo *Administrador*.

6.2.2 Fase: Alimentação do Peônia

Usuários utilizam o ambiente Peônia com o objetivo de reusar padrões de software no desenvolvimento de aplicações. Requisitos de teste e processos de desenvolvimento também são reutilizados em projetos executados com o apoio do ambiente Peônia. Para que padrões de software, requisitos de teste e processos de desenvolvimento possam ser reutilizados, eles devem estar previamente cadastrados e padrões devem estar associados às etapas de um processo e aos requisitos. Essas atividades são realizadas durante a fase de *Alimentação do Peônia*, ilustradas no início da Seção 6.2 pela Figura 6.2.

Na Seção 6.2.2.1, são descritas as atividades de manipulação de domínios. Na Seção 6.2.2.2, são apresentadas as atividades de controle de requisitos de teste. Na Seção 6.2.2.3, é explicada a atividade de geração automática de requisitos de teste. Na Seção 6.2.2.5, são

apresentadas as atividades de gerência de padrões de software, inclusive as relacionadas a associação e desassociação de domínios, outros padrões de software e requisitos de teste. Na Seção 6.2.2.6, são apresentadas as atividades de administração de fases, atividades, artefatos e papéis. Na Seção 6.2.2.7, são apresentadas as atividades de manipulação de processos de desenvolvimento, tanto da essência quanto da estrutura, e atividades de associação e desassociação de padrões de software às fases e atividades de um processo de desenvolvimento.

6.2.2.1 Atividades: Cadastrar, Alterar, Remover, Visualizar ou Listar Domínios

O *Domínio* é utilizado para agrupar padrões de software, permitindo que usuários possam pesquisá-los e empregá-los de acordo com o domínio do projeto. Para poder classificar um padrão, os domínios devem estar previamente cadastrados (*Atividade: Cadastrar Domínio*).

Figura 6.8: Tela de erro exibida ao requisitar a atividade *Remover Domínio* de um domínio associado a um padrão de software.

Um domínio é identificado pelo campo obrigatório "Nome", que deve ser único por usuário. Após o cadastro, ele é adicionado à lista de domínios (*Atividade: Listar Domí-*

nios) e pode ser selecionado para visualização, alteração, ou remoção (*Atividade: Visualizar, Alterar* ou *Remover Domínio*). Um domínio só pode ser removido caso não esteja associado a padrões de software. Na Figura 6.8, é exibida a tela de erro produzida quando um domínio associado a um padrão de software é selecionado para remoção. A atividade de associação de domínio a padrões de software é detalhada na Seção 6.2.2.5.

6.2.2.2 Atividades: Cadastrar, Alterar, Remover, Visualizar ou Listar Requisitos de Teste

De acordo com a abordagem ARTe, requisitos de teste podem ser associados aos padrões de software (*Atividade: Associar Requisito de Teste ao Padrão de Software*). Para que isso ocorra no ambiente Peônia, é necessário que esses requisitos de teste estejam previamente cadastrados (*Atividade: Cadastrar Requisito de Teste*).

Inicialmente, quatro tipos de requisitos de teste podem ser cadastrados:

- Não Classificado: requisito de teste criado pelo usuário e que não se encaixa com nenhuma das classificações de critério de teste fornecidas pelo ambiente Peônia.
- Valor Limite: requisito de teste criado pelo usuário utilizando o critério Análise do Valor Limite.
- Classe de Equivalência: requisito de teste criado pelo usuário utilizando o critério Particionamento de Equivalência.
- Valor Limite e Classe de Equivalência: requisito de teste criado pelo usuário utilizando os critérios Análise do Valor Limite e Particionamento de Equivalência, ou seja, o Teste Funcional Sistemático.

A atividade de cadastro de requisito de teste é realizada na tela apresentada na Figura 6.9 e é dividida em duas partes: inserção de informações essenciais do requisito de teste e cadastro de dados sobre o critério de teste. A primeira parte inclui o preenchimento dos campos "Código Identificador" e "Descrição" (*Atividades Cadastrar Essência do Requisito de Teste*). A segunda parte inclui a seleção do critério de teste empregado e o preenchimento do formulário com os dados sobre as condições do requisito de teste.

A seta "A" vinho indica o campo obrigatório "Código Identificador" utilizado para auxiliar usuários na identificação de requisitos de teste cadastrados. Esse campo pode ser repetido na sessão do usuário e deve ser preenchido com um valor que facilite a busca ou agrupamento dos requisitos de teste, por exemplo, "RT1" ou "REQUISITO DE TESTE 1". Com relação ao agrupamento de informações, para conjuntos de requisitos de teste recomenda-se preencher o campo "Código Identificador" com valores iguais de prefixo ou

sufixo, por exemplo, “RT”, possibilitando que sejam visualizados com maior facilidade ao serem listados (*Atividade: Listar Requisitos de Teste*).

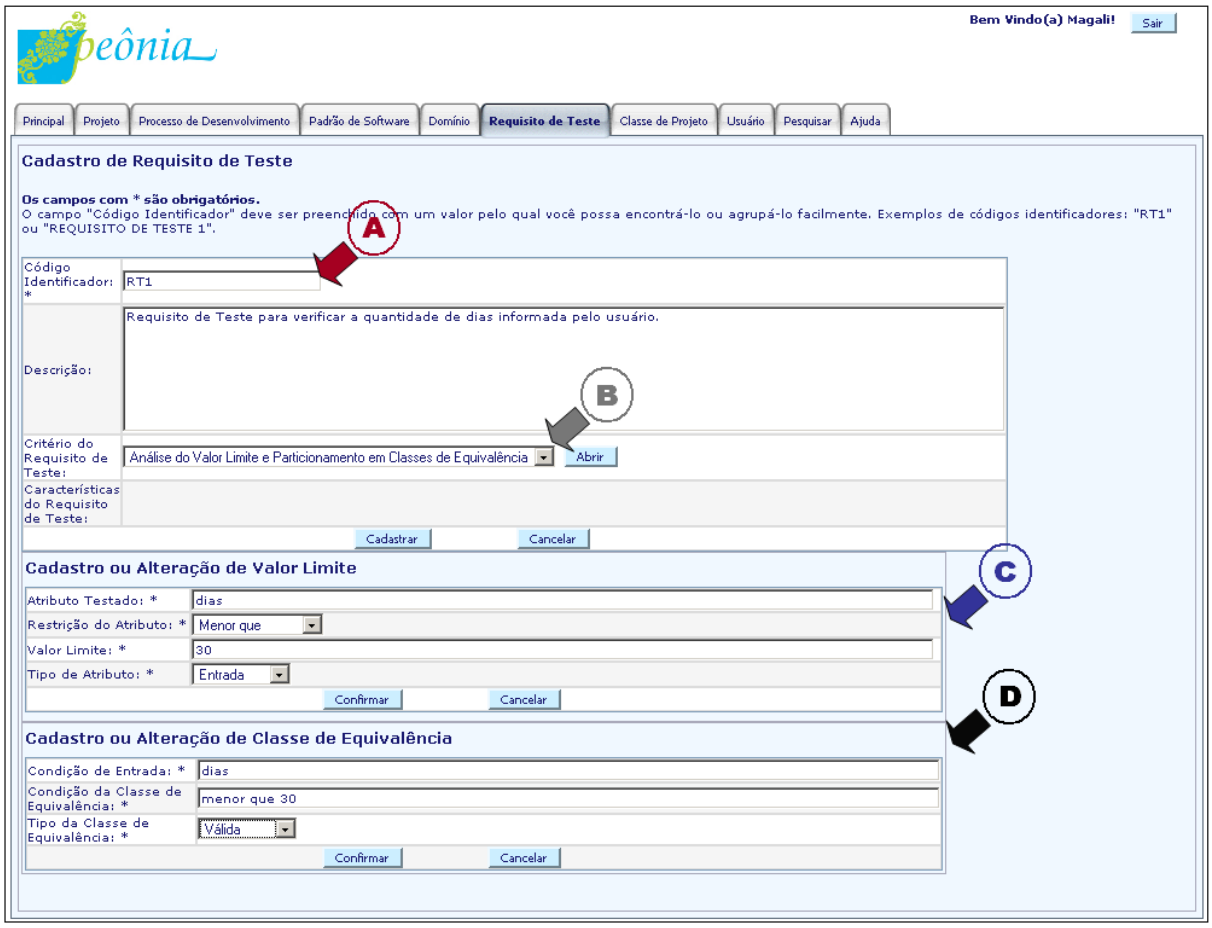

Figura 6.9: Tela exibida ao requisitar a atividade *Cadastrar Requisito de Teste*.

A seta “B” cinza na Figura 6.9 indica o campo onde o usuário classifica o requisito de teste cadastrado. De acordo com o valor selecionado, o formulário do critério de teste escolhido é apresentado.

Caso seja selecionado o critério Análise do Valor Limite, é exibido o formulário de cadastro ou alteração de valor limite (*Atividades: Cadastrar* ou *Alterar Valor Limite*), sinalizado na Figura 6.9 pela seta “C” azul marinho.

No entanto, se o critério selecionado for Particionamento de Equivalência, é exibido o formulário de cadastro ou alteração de classe de equivalência (*Atividades: Cadastrar* ou *Alterar Classe de Equivalência*), sinalizado pela seta “D” preta.

Quando é realizado o cadastro ou alteração de requisitos de teste não classificados os formulários indicados pelas setas “C” azul marinho e “D” preta não são apresentados e apenas as informações inseridas no campo “Descrição” caracterizam o requisito de teste, ou seja, não há cadastro separado das condições de teste e são executadas apenas as

atividades de gerência das informações essenciais do requisito (*Atividades: Cadastrar* ou *Alterar Essência do Requisito de Teste*).

Se for selecionada a utilização conjunta dos critérios Análise do Valor Limite e Particionamento de Equivalência, tanto o formulário indicado pela seta "C" azul marinho quanto o sinalizado pela seta "D" preta são exibidos para que sejam preenchidos com os dados requeridos.

Figura 6.10: Tela exibida ao selecionar a aba *Requisito de Teste*.

Na alteração de requisito de teste (*Atividades: Alterar Requisito de Teste* e *Alterar Essência do Requisito de Teste*), se for realizada a modificação do critério de teste cadastrado, as informações referentes ao critério anterior são removidas do ambiente Peônia. Por exemplo, se um requisito de teste utiliza o critério Análise do Valor Limite e ao alterá-lo passa a empregar o Particionamento de Equivalência, o valor limite do requisito é removido (*Atividade: Remover Valor Limite*) e uma classe de equivalência é inserida. Da mesma forma, a classe de equivalência é removida (*Atividade: Remover Classe de Equivalência*), caso seja escolhido a Análise do Valor Limite durante a alteração. Se for escolhida a não classificação do requisito de teste, tanto o valor limite quanto a classe de

equivalência são removidos. Na alteração de critério, somente não ocorre remoção se for utilizado em conjunto a Análise do Valor Limite e Particionamento de Equivalência.

Um requisito de teste só pode ser removido (*Atividade: Remover Requisito de Teste*) se não estiver associado a padrões de software ou sendo utilizado em projetos. Ao remover um requisito de teste todas as informações relacionadas também são removidas (*Atividades: Remover Essência do Requisito de Teste, Remover Valor Limite* e *Remover Classe de Equivalência*).

Na Figura 6.10, é exibida a lista de requisitos de teste cadastrados pelo usuário no ambiente Peônia (*Atividade: Listar Requisito de Teste*). Ao selecionar um elemento da lista, todas as informações ligadas ao requisito de teste são exibidas (*Atividade: Visualizar Requisito de Teste* que inclui as *Atividades: Visualizar Essência do Requisito de Teste, Visualizar Valor Limite* e *Visualizar Classe de Equivalência*), como pode ser observado pelo quadro da Figura 6.10 apontado pela seta "A" vinho.

6.2.2.3 Atividade: Gerar Requisito de Teste

Outra funcionalidade oferecida pelo ambiente Peônia é a geração de requisitos de teste do tipo classe de equivalência a partir de um valor limite (*Atividade: Gerar Requisito de Teste*). Primeiramente, é necessário selecionar um dos elementos apresentados na lista de requisitos de teste, ilustrada na Figura 6.10 da Seção 6.2.2.2, e escolher o botão "Gerar Requisito", indicado pela seta "B" cinza.

Em seguida, são exibidas informações a respeito do requisito de teste selecionado, como mostrado pela seta "A" vinho na Figura 6.11. Caso o requisito de teste já possua valor limite e classe de equivalência, ou não possua valor limite, uma mensagem é apresentada ao usuário alertando sobre a impossibilidade de gerar o requisito de teste. Caso contrário, são processadas as informações do valor limite selecionado e, de acordo com o tipo selecionado, os dados da classe de equivalência a ser gerada são apresentados para o usuário no quadro sinalizado pela seta "B" cinza. Três tipos de classes de equivalência podem ser escolhidas, como pode ser observado na seta "C" azul marinho:

- Válida: contém condições que produzem testes com resultados aceitos positivamente. A classe de equivalência é produzida com as mesmas informações contidas no valor limite, inclusive o "Código Identificador", o que facilita o reconhecimento das informações geradas.

- Inválida: contém condições que produzem testes com resultados que devem resultar em situações de alerta ou erro. A classe de equivalência é produzida com o oposto da condição contida no valor limite, mas com o mesmo "Código Identificador", per-

mitindo o reconhecimento da relação entre o valor limite e a classe de equivalência gerada.

- Válida e Inválida: são produzidas classes de equivalência tanto as condições aceitas positivamente quanto as que causam situações de alerta ou erro. A classe de equivalência válida é criada com as mesmas informações do valor limite, enquanto a inválida é criada com o oposto da condição do valor limite e com o "Código Identificador" acrescido do sufixo " - INV", facilitando o reconhecimento do requisito de teste gerador.

  Apesar de atualmente só fornecer suporte a técnica de teste funcional, espera-se ampliar o apoio oferecido em trabalhos futuros, detalhados na Seção 8.4.

Figura 6.11: Tela exibida ao requisitar a atividade *Gerar Requisito de Teste*.

6.2.2.4 Atividades: Cadastrar, Alterar, Remover, Visualizar ou Listar Classes de Projeto

Diagramas e *Estruturas da Solução* são compostas de classes de projeto, que podem ou não estar relacionadas. Por causa da restrição do cronograma, a inserção e visualização de classes de projeto foram realizadas por meio de formulários, sem a usual apresentação visual. Assim, futuramente essa implementação visual será trabalhada.

Essas classes de projeto devem estar previamente cadastradas (*Atividade: Cadastrar Classe de Projeto*), para que possam ser relacionadas e empregadas. O cadastro de classes de projeto pode ser requisitado na aba de "Classe de Projeto", durante a manipulação da estrutura de um diagrama de classes ou modelo conceitual ou ao gerenciar a estrutura da solução de um padrão. As atividades de controle de diagramas são detalhadas na Seção 6.2.3.5 e as de estrutura da solução na Seção 6.2.2.5.

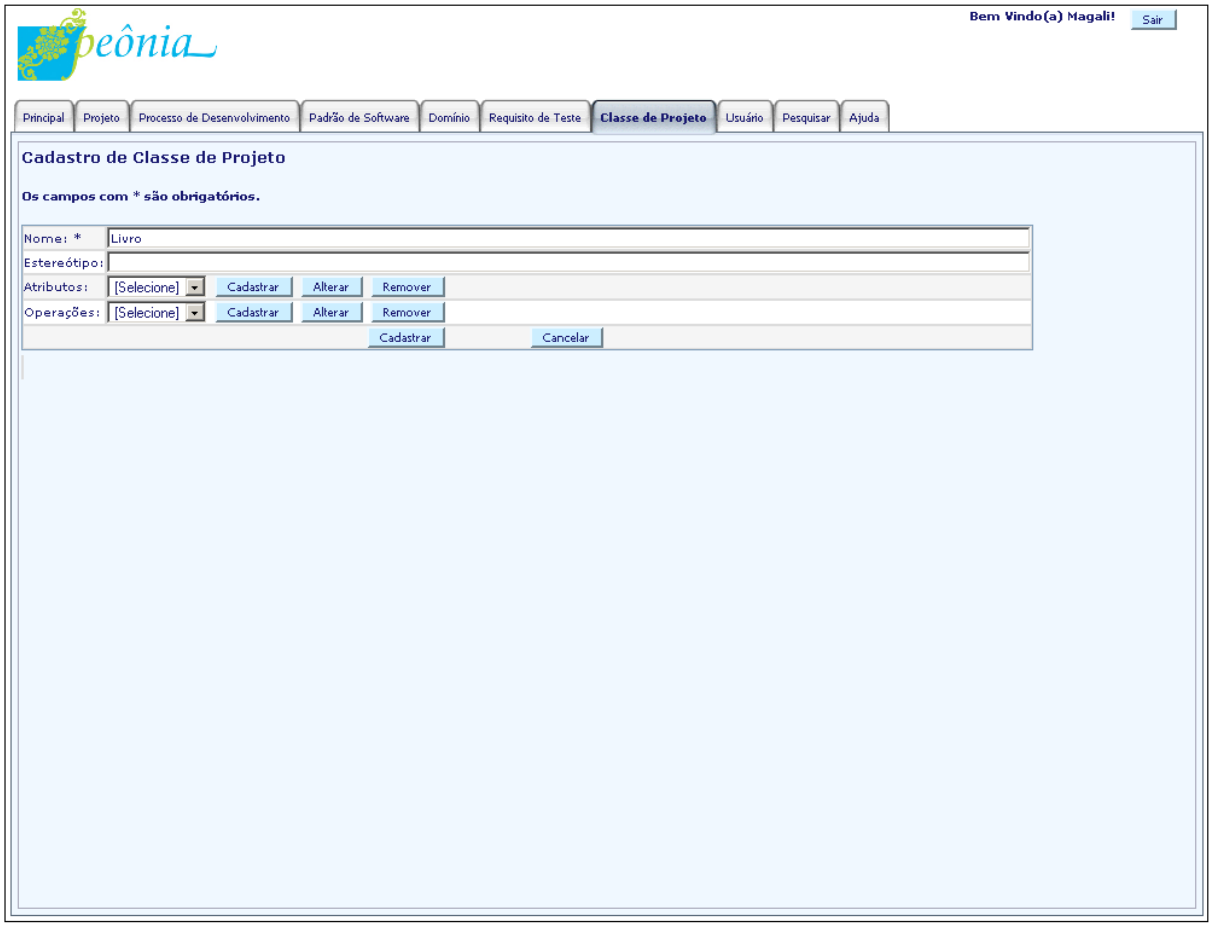

Figura 6.12: Tela exibida ao requisitar a atividade *Cadastrar Classe de Projeto*.

Cada classe de projeto é composta obrigatoriamente por "Nome" e podem conter opcionalmente "Estereótipo", atributos e operações. Ao requisitar uma das atividades de

gerenciamento de atributos e operações (*Atividades: Cadastrar, Alterar ou Remover Atributos* e *Cadastrar, Alterar ou Remover Operações*) um formulário é exibido na parte inferior da tela, como o exemplificado pela seta "A" vinho na Figura 6.12, para que os dados sejam preenchidos. Para que o cadastro de atributos e operações seja realizado com sucesso, é necessário informar pelo menos o campo "Nome" desses elemento e, opcionalmente, também pode ser fornecido "Tipo" para o atributo e "Parâmetro" para a operação.

Após informar os atributos e operações, eles são inseridos nas listas mostrada pela seta "B" cinza e "C" azul marinho, respectivamente, e podem ser alterados, removidos ou visualizados a qualquer momento (*Atividades: Alterar, Remover ou Visualizar Atributo* ou *Alterar, Remover ou Visualizar Operações*), bastando selecioná-lo. Todos os elementos inseridos na lista de atributos e operações da classe de projeto são salvos no ambiente Peônia apenas quando confirmada a operação de cadastro ou alteração (*Atividades: Cadastrar* ou *Alterar Classe de Projeto*).

Classes de projeto podem ser visualizadas (*Atividade: Visualizar Classe de Projeto*) escolhendo-as na lista de classes de projeto (*Atividade: Listar Classes de Projeto*) apresentada ao usuário na aba "Classe de Projeto". Destaca-se que a remoção (*Atividade: Remover Classe de Projeto*) só é possível se a classe de projeto não for utilizada em diagramas ou estruturas da solução.

6.2.2.5 Atividades: Cadastrar, Alterar, Remover, Visualizar ou Listar Padrões de Software

Como mencionado na Seção 2.4, não existe um consenso quanto ao formato de documentação de padrões de software (Braga et al., 2001). Para os padrões de software do ambiente Peônia foram utilizados os elementos obrigatórios propostos pelo padrão *"Presença de Elemento Obrigatório"* (*Mandatory Element Present*), da linguagem de padrões de Meszaros e Doble (1996), e as informações complementares recomendadas por Appleton (2000). Além disso, podem ser associadas aos padrões de software cadastrados no ambiente Peônia informações sobre requisitos de teste e domínios e criada uma estrutura da solução, como a utilizada por Gamma et al. (1995).

Essas informações são fornecidas em três etapas (*Atividade: Cadastrar Padrão de Software*). Na primeira etapa, ilustrada na Figura 6.13, são informados os elementos obrigatórios e complementares, cujos campos são: "Nome do Padrão", "Problema", "Solução", "Contexto", "Força", "Exemplo", "Contexto Resultante", "Raciocínio", "Usos Conhecidos" e "Comentários Gerais" (*Atividades: Cadastrar* ou *Alterar Essência do Padrão de Software*).

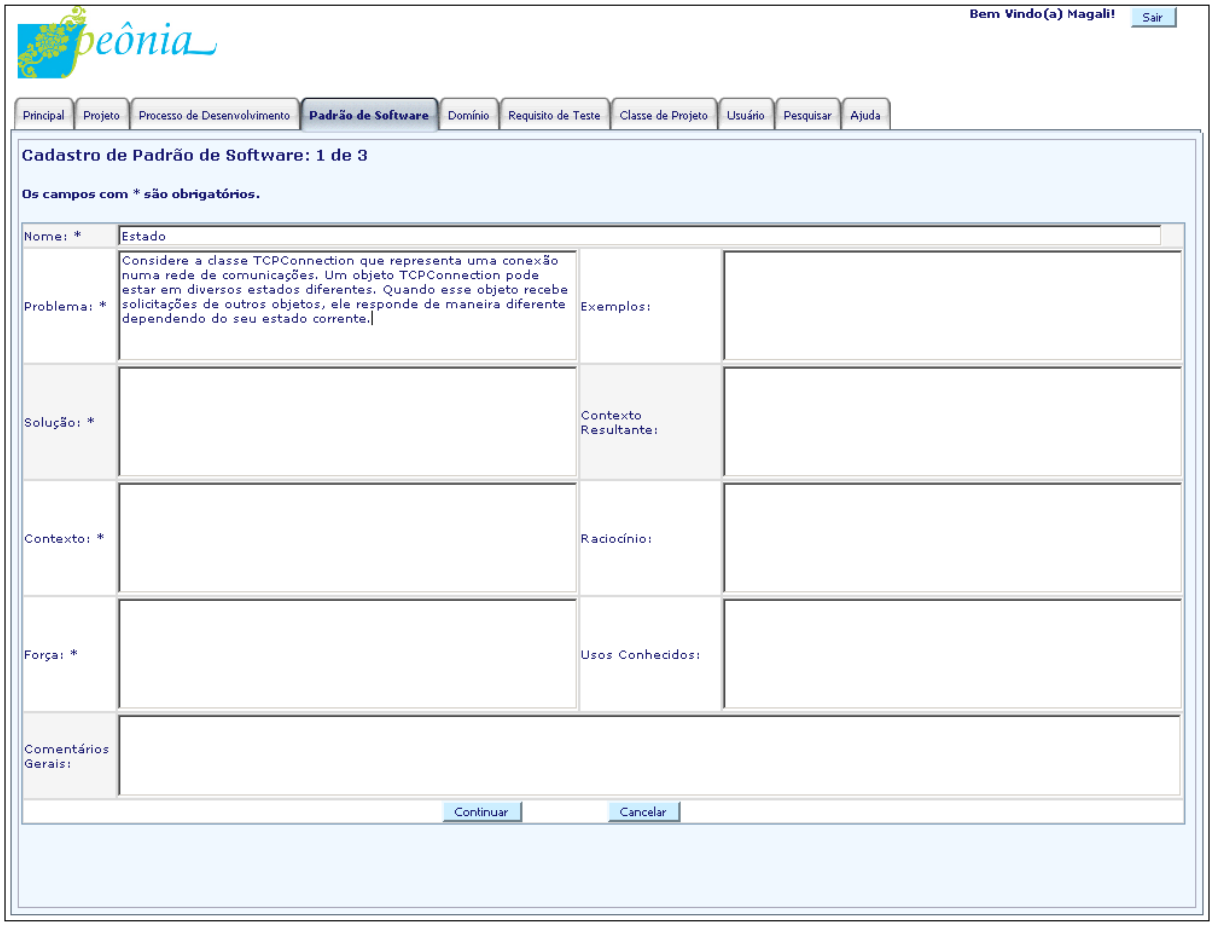

Figura 6.13: Tela da primeira parte da atividade *Cadastrar Padrão de Software*.

Na segunda etapa, apresentada na Figura 6.14, o usuário pode realizar a associação de domínios (*Atividade: Associar Domínio ao Padrão de Software*), padrões relacionados (*Atividade: Associar Padrões de Software*) ou requisitos de teste (*Atividade: Associar Requisito de Teste ao Padrão de Software*), sinalizados respectivamente pelas setas "A" vinho, "B" cinza e "C" azul marinho. Na atividade de cadastro de padrão de software, apenas os domínios e requisitos de teste inseridos pelo usuário são apresentados nas listas indicadas pelas setas "A" vinho e "C" azul marinho. No entanto, se for executada a atividade de alteração (*Atividade: Alterar Padrão de Software*), além dos domínios e requisitos de teste do usuário, são exibidos os elementos já associados ao padrão (*Atividades: Listar Domínios do Padrão de Software* ou *Listar Requisitos de Teste do Padrão de Software*), independente de terem sido ou não cadastrados pelo indivíduo que está modificando, permitindo que o usuário altere as associações (*Atividades: Desassociar Domínio do Padrão de Software* ou *Desassociar Requisito de Teste do Padrão de Software*).

Com relação à lista sinalizada pela seta "B" cinza na Figura 6.14, todos os padrões de software do ambiente Peônia são apresentados para que o usuário possa relacionar

o padrão que está sendo cadastrado com os outros existentes. Durante a atividade de alteração, os padrões já relacionados são exibidos assinalados (*Atividade: Listar Padrões de Software Associados*), permitindo que o usuário remova a seleção e, conseqüentemente, a relação entre os padrões de software (*Atividade: Desassociar Padrões de Software*).

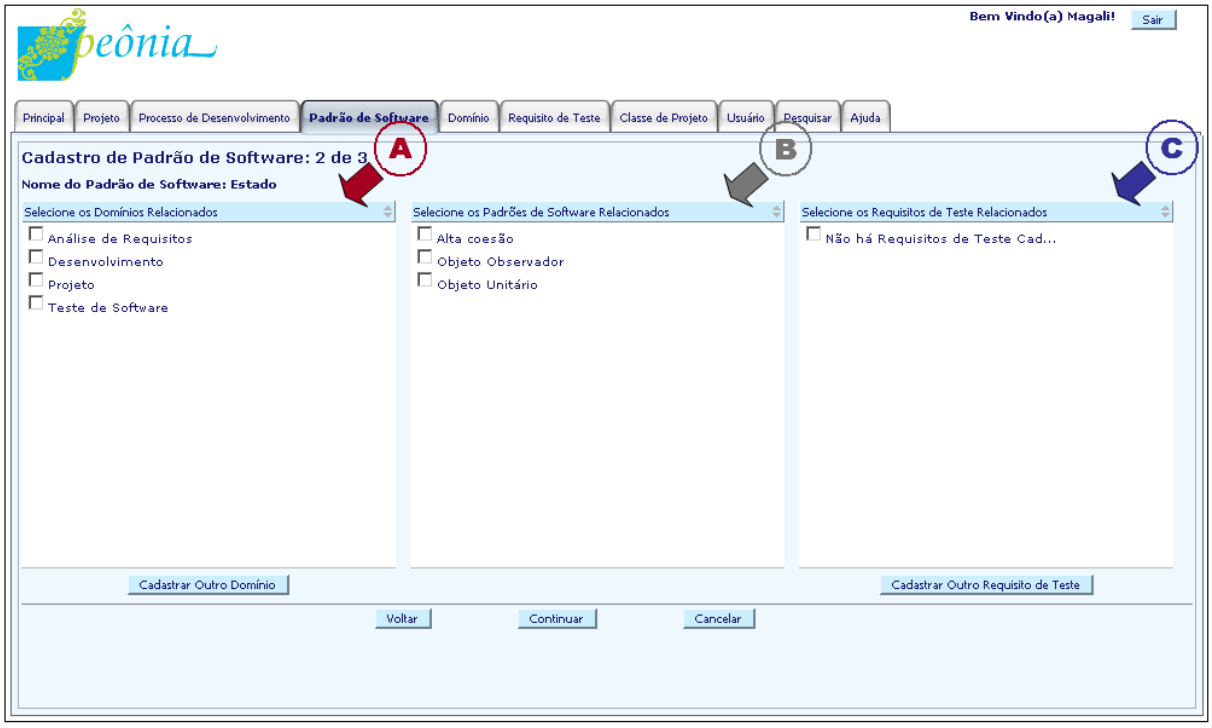

Figura 6.14: Tela da segunda parte da atividade *Cadastrar Padrão de Software.*

Na última etapa, ilustrada na Figura 6.15, o usuário pode relacionar classes de projeto formando uma estrutura que representa a solução proposta pelo padrão de software (*Atividade: Cadastrar Estrutura da Solução*). Assim como na segunda etapa, apenas as classes de projeto cadastradas pelo usuário são exibidas para serem selecionadas, a não ser que seja uma alteração (*Atividade: Alterar Estrutura da Solução*), sendo exibidas as classes de projeto da estrutura da solução (*Atividade: Visualizar Estrutura da Solução*), independente de quem as cadastrou.

A seta "A" vinho na Figura 6.15 mostra o formulário que deve ser utilizado para informar os elementos que irão compor a estrutura da solução. O usuário pode selecionar uma ou relacionar duas classes de projeto. Além disso, também pode informar o tipo, nome e cardinalidade do relacionamento. Após fornecer os dados, deve ser escolhida a opção para criar a relação de classes de projeto (*Atividade: Associar Classes de Projeto*), sinalizada pela seta "B" cinza, e o ambiente Peônia a adiciona à estrutura da solução, como indicado pela seta "C" azul marinho na Figura 6.16.

Depois que um relacionamento entre classes de projeto é inserido, o usuário pode alterá-lo selecionando-o na lista, modificando seus dados e clicando na opção de alteração de associação (*Atividade: Alterar Associação Entre Classes de Projeto*), apontada pela seta "D" preta na Figura 6.16. Ao escolher um dos elementos da lista, as informações das classes de projeto da associação são apresentadas no formulário indicado pela seta "E" verde. O usuário também pode remover a associação entre classes de projeto da estrutura da solução selecionando a opção sinalizada pela seta "F" rosa (*Atividade: Desassociar Classes de Projeto*).

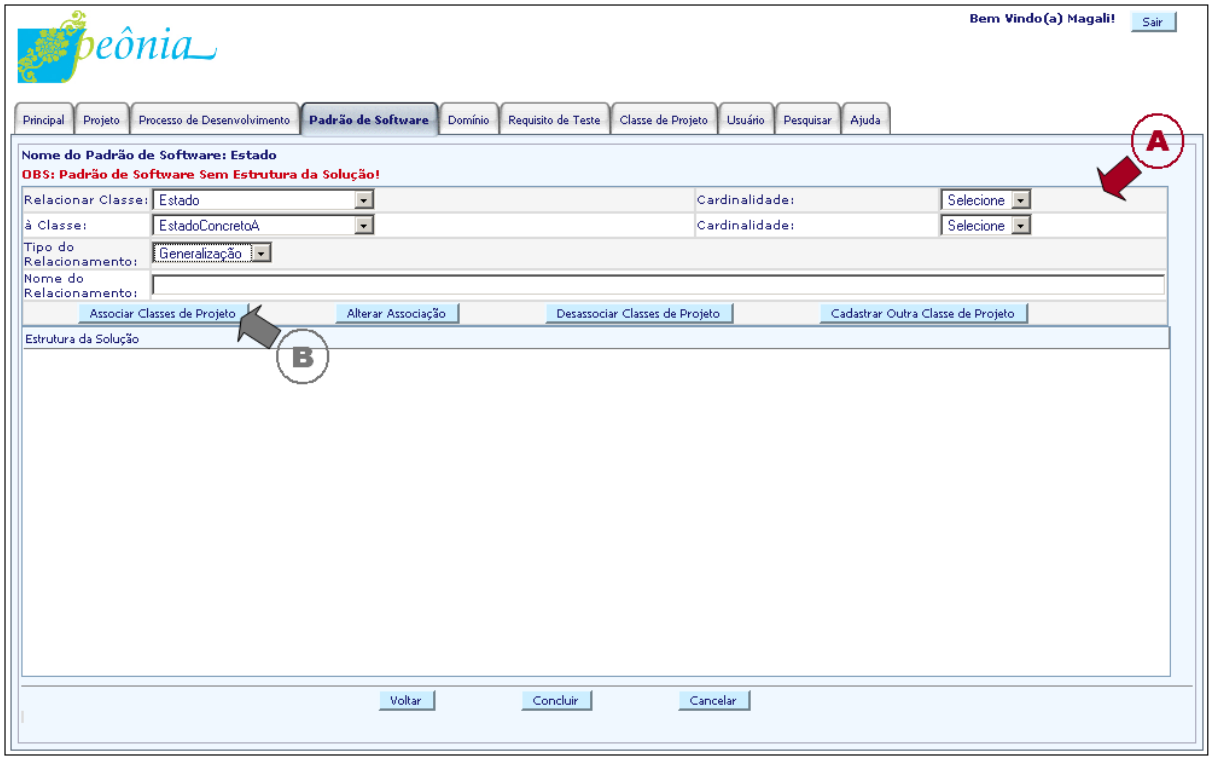

Figura 6.15: Tela da terceira parte da atividade *Cadastrar Padrão de Software*.

Após o cadastro, o padrão de software é inserido no "Repositório de Padrões de Software do Ambiente" e pode ser visualizado por todos os usuários. Esse repositório é a lista de padrões (*Atividade: Listar Padrões de Software*) indicada pela seta "A" vinho na Figura 6.17. Ao selecioná-lo, apenas parte das informações são exibidas, sinalizado pela seta "B" cinza. Para visualizar as informações completas, a opção indicada pela seta "C" azul marinho na Figura 6.17 deve ser escolhida (*Atividade: Visualizar Padrão de Software*), gerando a tela da Figura 6.18.

A visualização do padrão de software inclui a exibição das informações cadastradas nas duas primeiras etapas (*Atividades: Visualizar Essência do Padrão de Software, Listar Domínios do Padrão de Software, Listar Requisitos de Teste do Padrão de Software* e

Listar Padrões de Software Associados). Caso o padrão possua uma estrutura da solução, como indicado pela seta "A" vinho na Figura 6.18, o usuário pode requerer a visualização da sua lista de relacionamento entre classes de projeto selecionando a opção sinalizada pela seta "B" cinza (*Atividade: Visualizar Estrutura da Solução*) e produzindo a tela da Figura 6.19.

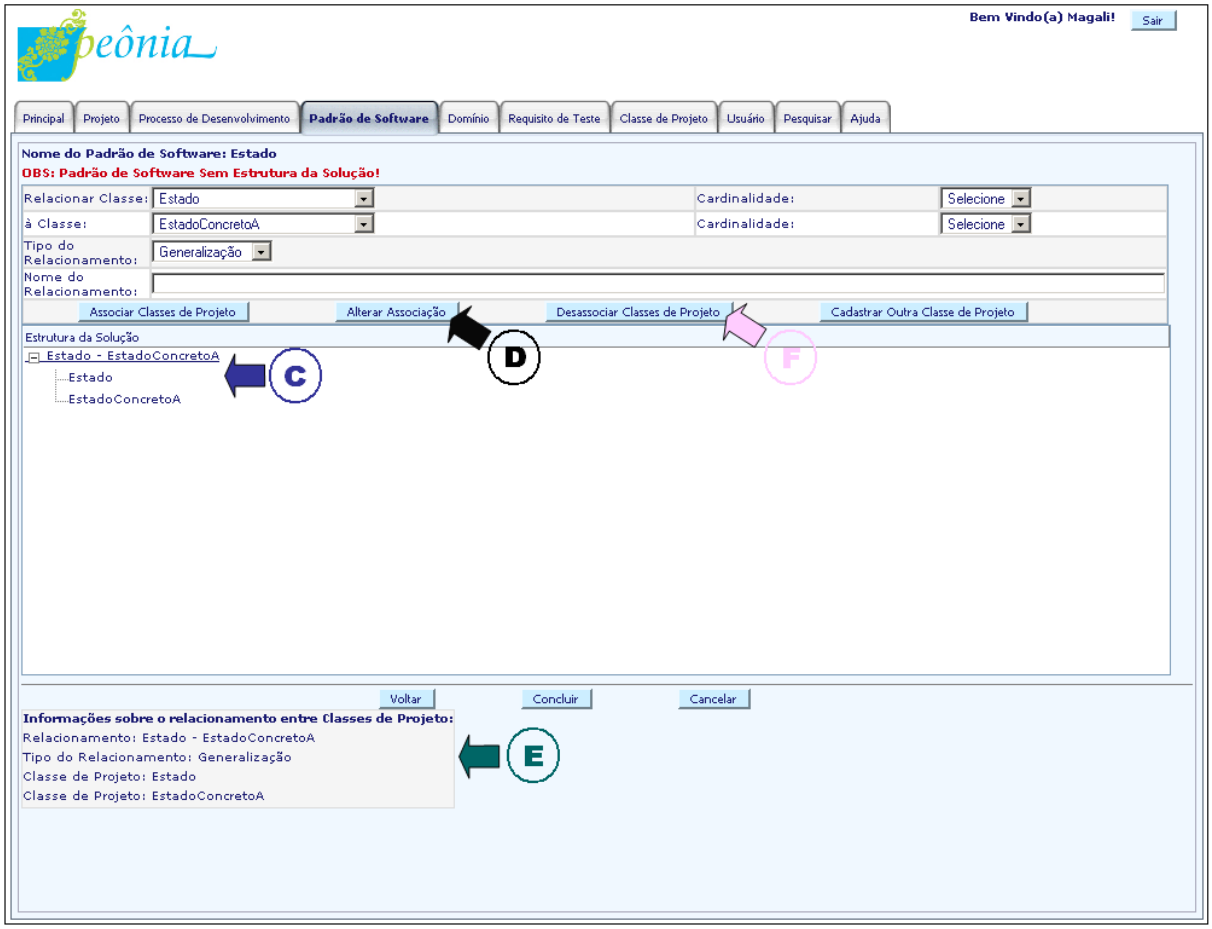

Figura 6.16: Tela exibida durante as atividades *Associar, Alterar Associação* ou *Desassociar Classes de Projeto*.

Apesar do padrão de software inserido ser visível para todos, só quem o cadastrou ou um *Administrador* tem autorização para alterá-lo ou removê-lo (*Atividades: Alterar* ou *Remover Padrão de Software*). Um padrão de software só pode ser removido se não estiver: associado às fases ou atividades de processos de desenvolvimento, relacionado a outros padrões ou empregado em diagramas. A remoção de um padrão inclui a remoção da estrutura da solução (*Atividade: Remover Estrutura da Solução*) e de todas as informações associadas (*Atividades: Remover Essência do Padrão de Software, Desassociar Domínio do Padrão de Software, Desassociar Requisito de Teste do Padrão de Software* e *Desassociar Padrões de Software*).

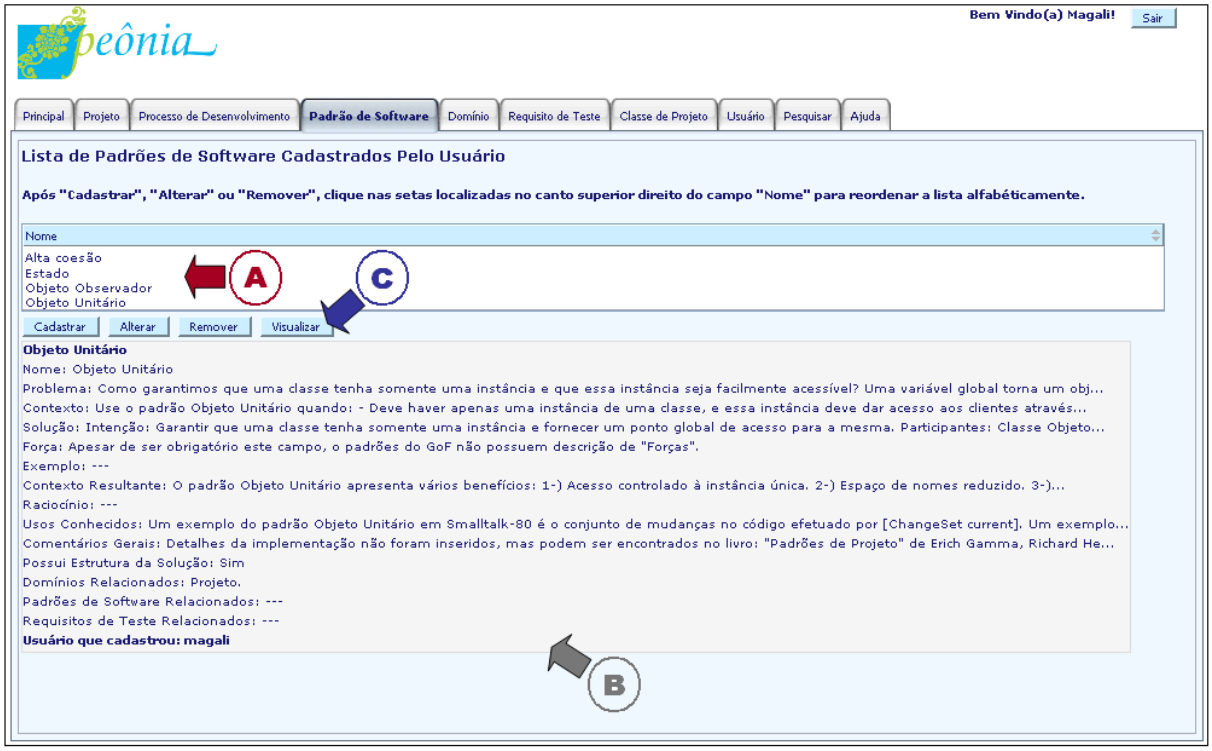

Figura 6.17: Tela exibida ao requisitar a atividade *Listar Padrões de Software.*

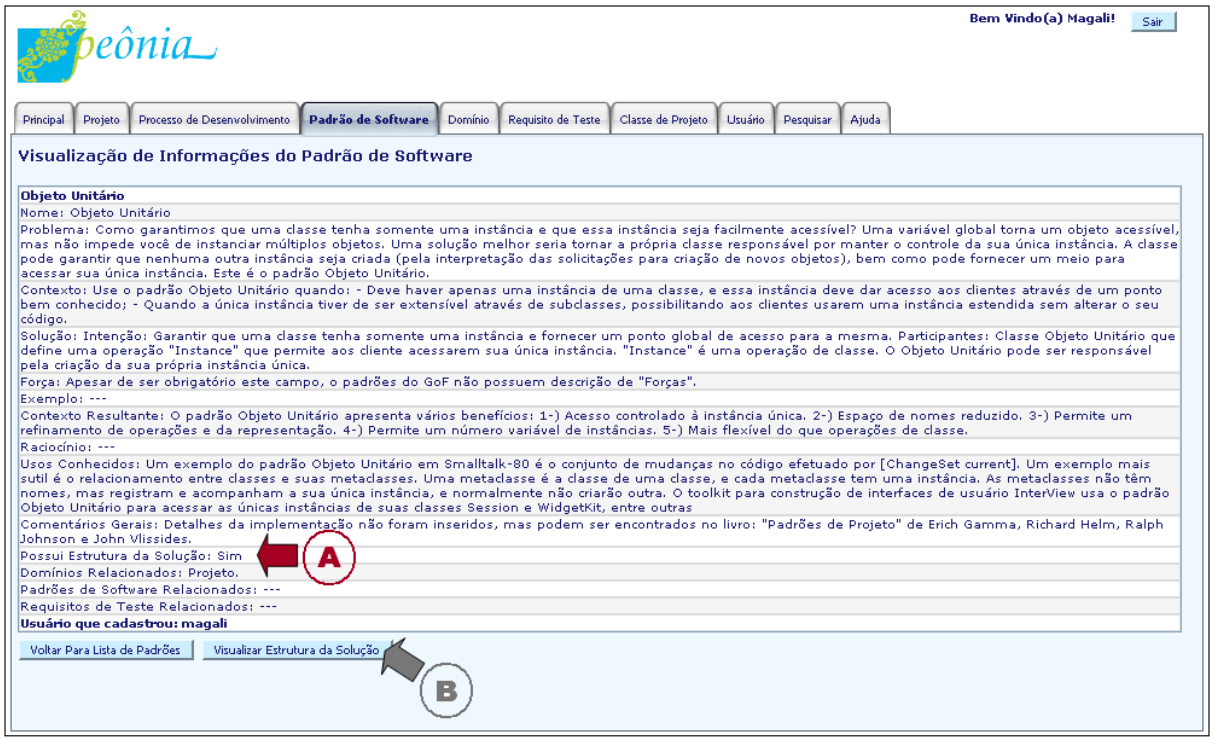

Figura 6.18: Tela exibida ao requisitar a atividade *Visualizar Padrão de Software.*

Figura 6.19: Tela exibida ao requisitar a atividade *Visualizar Estrutura da Solução*.

A associação de padrões de software às fases e atividades de um processo de desenvolvimento é realizada durante as atividade de gerência de processo, descritas na Seção 6.2.2.7.

6.2.2.6 Atividades: Cadastrar, Alterar, Remover, Visualizar ou Listar Fases, Atividades, Artefatos e Papéis

Os elementos fundamentais do RUP foram utilizados para a elaboração da estrutura dos processos de desenvolvimento do ambiente Peônia, assim como foram utilizados para a definição do Método Para Desenvolvimento Utilizando Padrões de Software, descrito na Seção 5.4.

Para indicar quais fases compõem um processo, é necessário que essas fases estejam previamente cadastradas (*Atividade: Cadastrar Fase*). Da mesma forma, atividades devem ser inseridas (*Atividade: Cadastrar Atividade*) para que possam ser atribuídas a fases. Por fim, com o cadastro antecipado de artefatos (*Atividade: Cadastrar Artefato*) e papéis (*Atividade: Cadastrar Papel*), é possível associá-los a atividades.

Fases, atividades, artefatos e papéis possuem o campo obrigatório "Nome", que é informado pelo usuário. O "Nome" deve ser único para cada usuário, ou seja, é permitido que usuários distintos tenham nomes iguais de fases, atividades, artefatos e papéis. Esse critério foi utilizado para:

- Evitar repetição desnecessária de cadastro de fases, atividades, artefatos e papéis, além de reduzir problemas conceituais e confusão na identificação dos elementos.

- Permitir que fases, atividades, artefatos e papéis de usuários diferentes tenham flexibilidade conceitual entre os usuários, pois um nome de elemento pode representar um conjunto de informações diferente para cada um dos usuários.

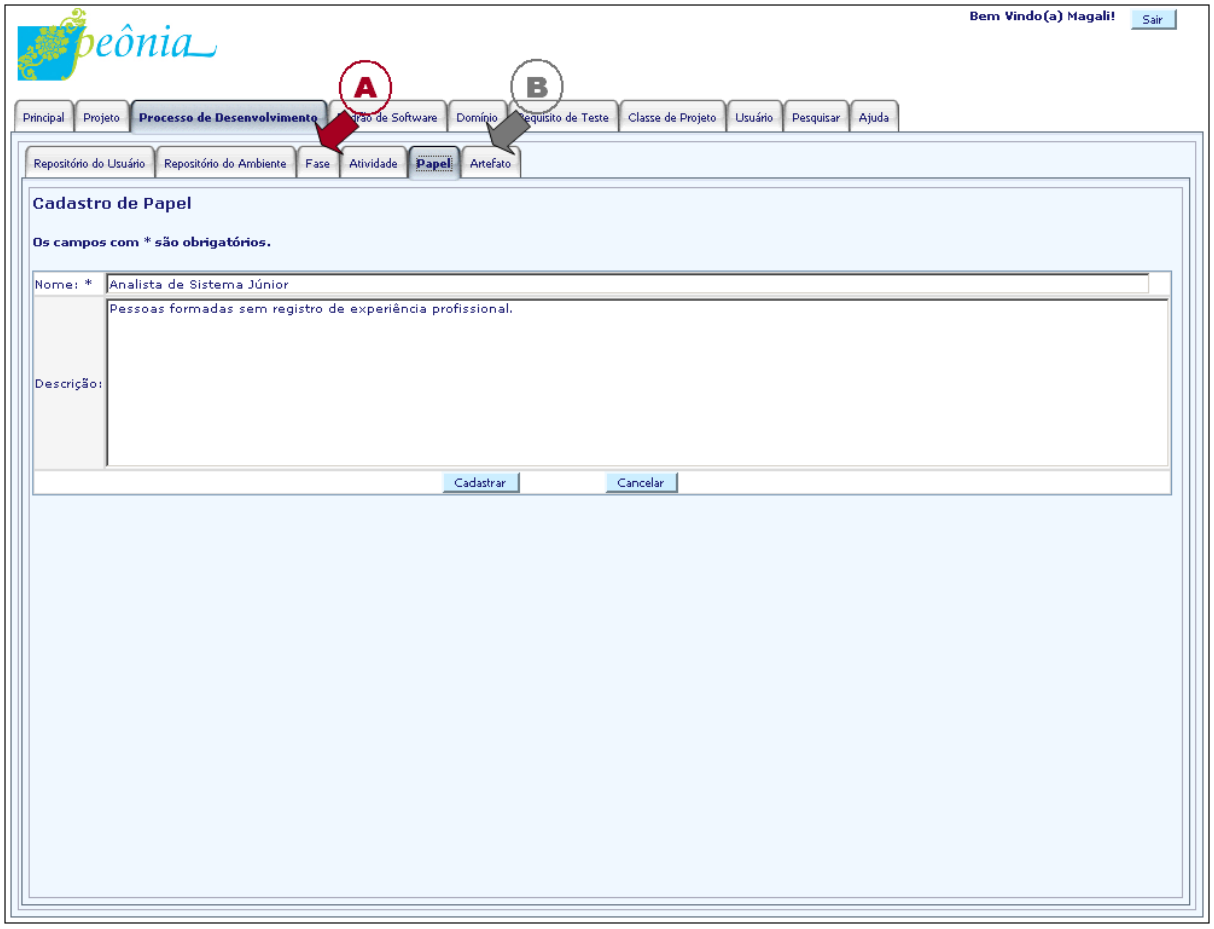

Figura 6.20: Tela exibida ao requisitar a atividade *Cadastrar Papel*.

Por exemplo, dois usuários do ambiente Peônia, "Engenheiro de Software A" e "Engenheiro de Software B", desejam cadastrar o papel "Analista de Sistema Júnior". Para o "Engenheiro de Software A", "Analista de Sistema Júnior" são pessoas com faculdade na área de computação que não possuem experiência profissional. Já para o "Engenheiro de Software B" esse papel é dado apenas a indivíduos formados que obrigatoriamente tenham feito estágio em alguma empresa. Assim, esses dois usuários tem idéias diferentes sobre o papel de "Analista de Sistema Júnior" e o ambiente Peônia permite que ambos cadastrem esse papel com esse nome, cada um com o seu significado.

No entanto, caso o "Engenheiro de Software A" queira subdividir esse papel, não é possível inserir outro "Analista de Sistema Júnior" com outras informações, o que pode confundir o usuário ao visualizar dois papéis com mesmo nome, sendo necessário conferir sempre qual deseja utilizar ou qual está sendo empregado. Assim, para subdividir é ne-

cessário cadastrar um outro papel, por exemplo, “Analista de Sistema Júnior 2”, tornando mais clara a visualização e consistente o conceito de “Analista de Sistema Júnior”.

Na Figura 6.20, é ilustrado o cadastro de papel, que contém os mesmo campos que os cadastros de fase e artefato. Nessa figura, a seta “A” vinho indica a aba (*tab*) que deve ser selecionada para realizar o cadastro de fase e a seta “B” cinza aponta a aba onde é efetuado o cadastro de artefato.

Além dos mesmos campos de fase, artefato e papel, a gerência de atividades possui o campo opcional “Papel Executor da Atividade”, indicando qual a pessoa responsável pela execução da atividade. Na Figura 6.21, a seta “A” vinho indica como é feita a atribuição de uma atividade a um papel executor durante o cadastro de atividade.

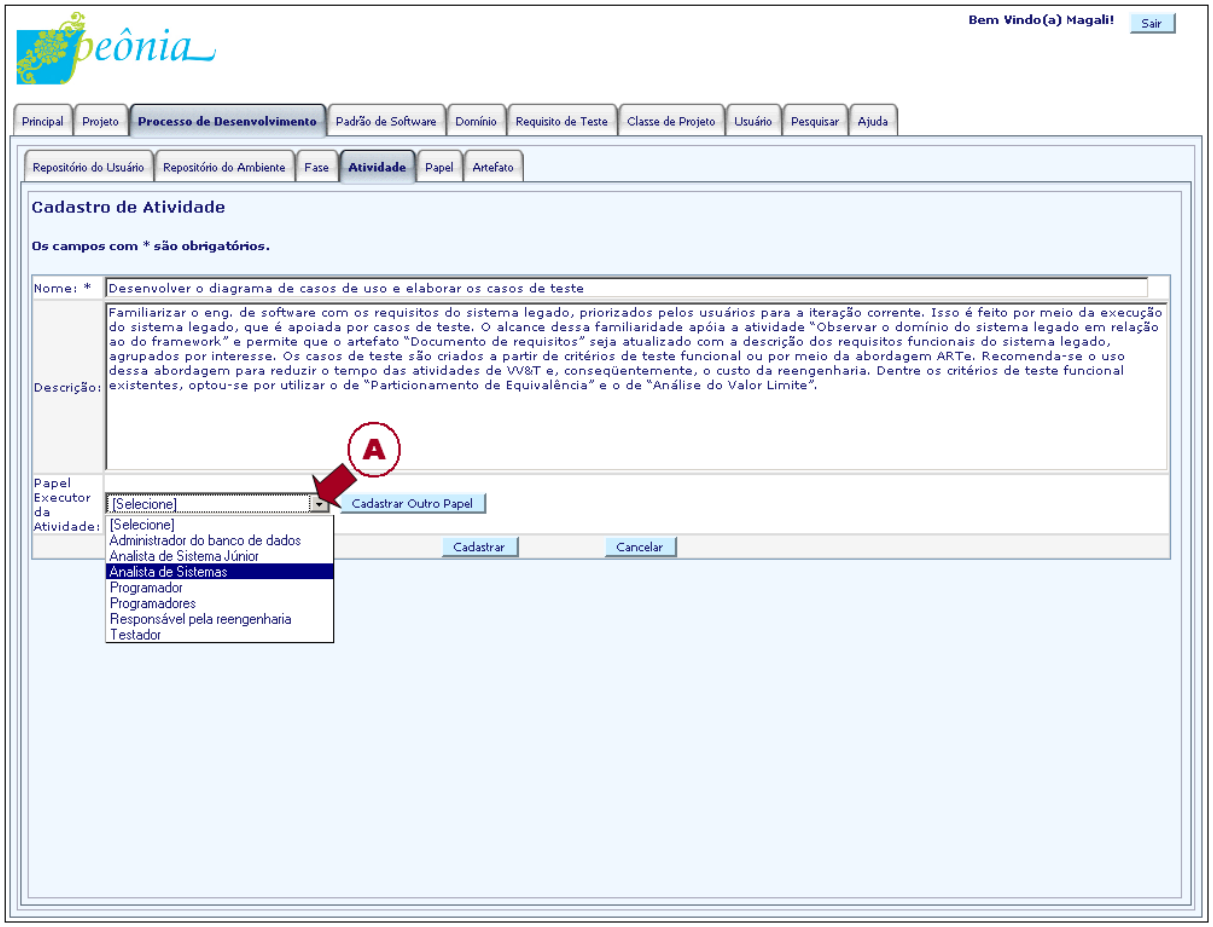

Figura 6.21: Tela exibida ao requisitar a atividade *Cadastrar Atividade*.

Após o cadastro, esses elementos podem ser alterados (*Atividades: Alterar Fase, Alterar Atividade, Alterar Artefato* ou *Alterar Papel*) ou removidos (*Atividades: Remover Fase, Remover Atividade, Remover Artefato* ou *Remover Papel*) do ambiente Peônia. A atividade de remoção só é permitida se o elemento não estiver sendo utilizado por processos de desenvolvimento.

Antes de realizar as operações de cadastro, alteração e remoção, uma lista de fases (*Atividade: Listar Fases*), atividades (*Atividade: Listar Atividades*), artefatos (*Atividade: Listar Artefatos*) e papéis (*Atividade: Listar Papéis*) para que o usuário possa procurar um elemento ou visualizar suas informações (*Atividades: Visualizar Fase, Visualizar Atividade, Visualizar Artefato* ou *Visualizar Papel*).

6.2.2.7 Atividades: Cadastrar, Alterar, Remover, Visualizar ou Listar Processos de Desenvolvimento do Repositório e do Ambiente

Processos de desenvolvimento devem ser previamente cadastrados para que possam ser empregados em projetos (*Atividade: Cadastrar Processo de Desenvolvimento*). Esse cadastro é realizado em duas etapas: inserção de informações sobre o processo (*Atividade: Cadastrar Essência do Processo de Desenvolvimento*) e definição da sua estrutura (*Atividade: Cadastrar Estrutura do Processo de Desenvolvimento*), composta por fases, atividades e artefatos (*Atividades: Associar Fases ao Processo de Desenvolvimento, Associar Atividades às Fases* e *Associar Artefatos às Atividades*).

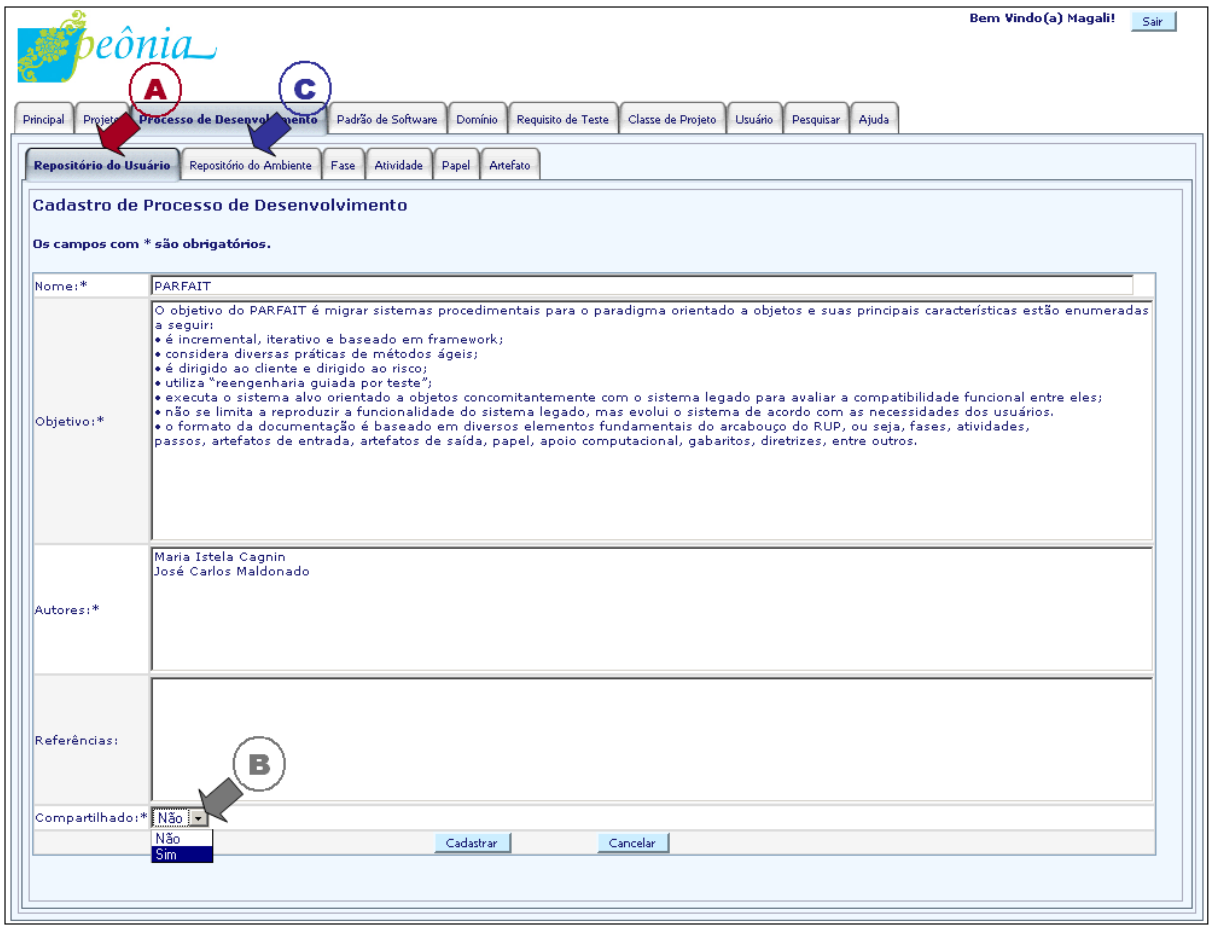

Figura 6.22: Tela exibida ao requisitar a atividade *Cadastrar Processo de Desenvolvimento.*

Dois tipos de repositório de processo de desenvolvimento são oferecidos:

- "Repositório do Usuário": local onde são armazenados todos os processos de desenvolvimento cadastrados pelo usuário. Apenas o usuário pode visualizar e utilizar esses processos.
- "Repositório do Ambiente": local onde são armazenados os processos de desenvolvimento compartilhados pelo usuário. Todos os usuários do ambiente Peônia podem visualizar e utilizar esses processos, no entanto, apenas administradores e o usuário que o cadastrou podem alterá-lo ou removê-lo.

Figura 6.23: Tela do *Repositório do Ambiente*.

O processo de desenvolvimento é inserido, primeiramente, no "Repositório do Usuário". Caso opte por compartilhá-lo, esse processo também é cadastrado no "Repositório do Ambiente". O cadastro de um novo processo é obrigatoriamente realizado na aba "Repositório do Usuário", indicado na Figura 6.22 pela seta "A" vinho. A seta "B" cinza mostra o campo da atividade de cadastro de processo onde o usuário informa se deseja compartilhá-lo. O campo obrigatório "Nome" é informado pelo usuário para identificar

um processo de desenvolvimento na sua sessão. Assim, não é permitido nomes repetidos de processos para um mesmo usuário.

No "Repositório do Ambiente", cuja aba é sinalizada na Figura 6.22 pela seta "C" azul marinho, apenas alteração (*Atividade: Alterar Processo de Desenvolvimento*) e remoção (*Atividade: Remover Processo de Desenvolvimento*) de processos são permitidos, sendo que somente usuários do tipo *Administrador*, ou o *Engenheiro de Software* que o cadastrou, têm autorização para executar essas atividades. Para outros usuários do tipo *Engenheiro de Software* é permitido apenas visualizar e empregar esses processos nos projetos que criarem. O "Repositório do Ambiente" apresentado para usuários do tipo *Engenheiros de Software* é ilustrado na Figura 6.23.

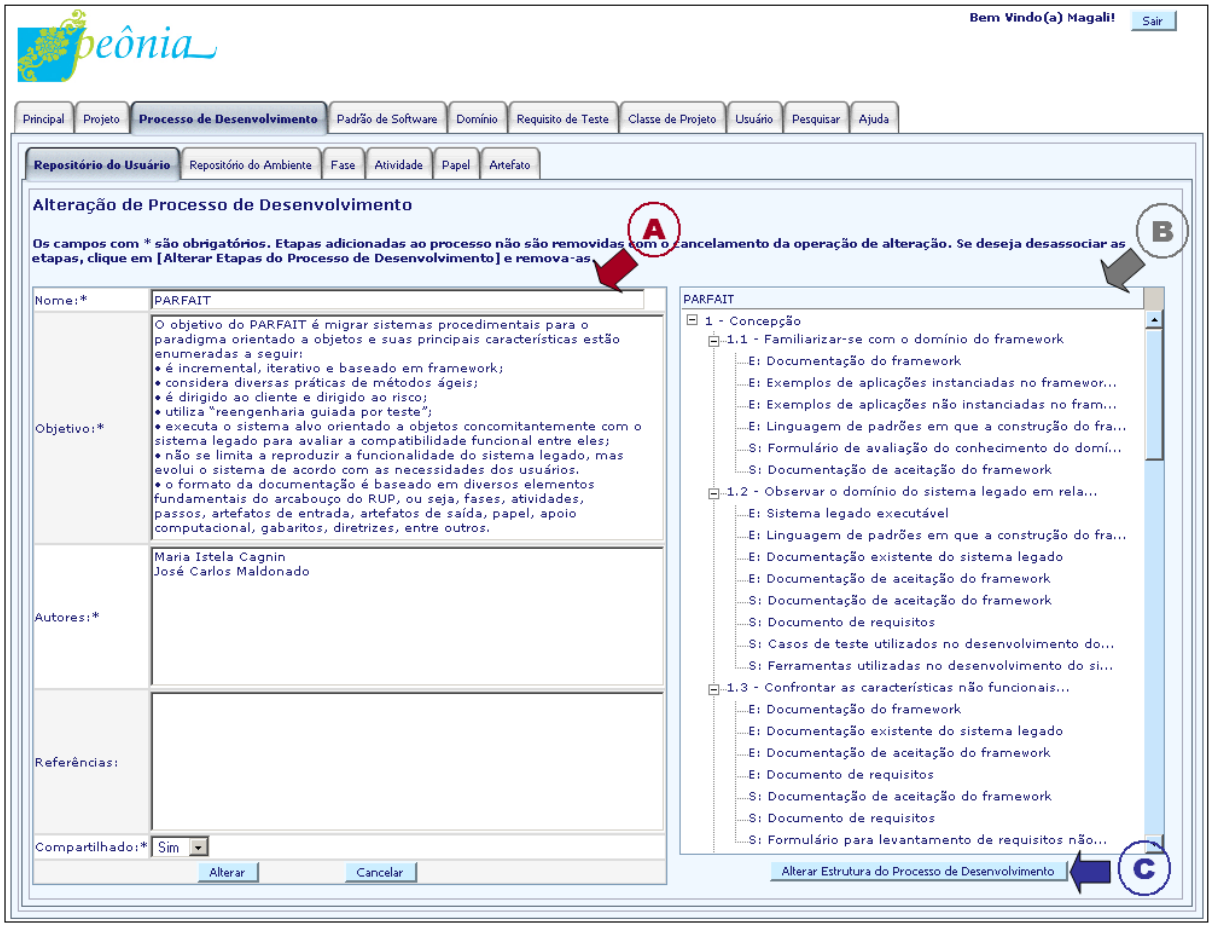

Figura 6.24: Tela exibida ao requisitar a atividade *Alterar Processo de Desenvolvimento*.

Nas listas dos repositórios (*Atividade: Listar Processos de Desenvolvimento*), ao selecionar um processo de desenvolvimento, as informações essenciais e a estrutura desse processo são exibidas (*Atividades: Visualizar Processo de Desenvolvimento, Visualizar Essência do Processo de Desenvolvimento* e *Visualizar Estrutura do Processo de Desen-*

volvimento). A visualização de informações de processo é exemplificada na Figura 6.23 pela seta "A" vinho.

Ao modificar as informações essenciais de um processo de desenvolvimento (*Atividade: Alterar Essência do Processo de Desenvolvimento*), cujo formulário é mostrado pela seta "A" vinho na Figura 6.24, a estrutura já cadastrada do processo é exibida (*Atividades: Visualizar Estrutura do Processo de Desenvolvimento, Listar Fases do Processo de Desenvolvimento, Listar Atividades da Fase* e *Listar Artefatos da Atividade*), indicada pela seta "B" cinza, permitindo que o usuário avalie a necessidade de também alterá-la *Atividade: Alterar Estrutura do Processo de Desenvolvimento.*

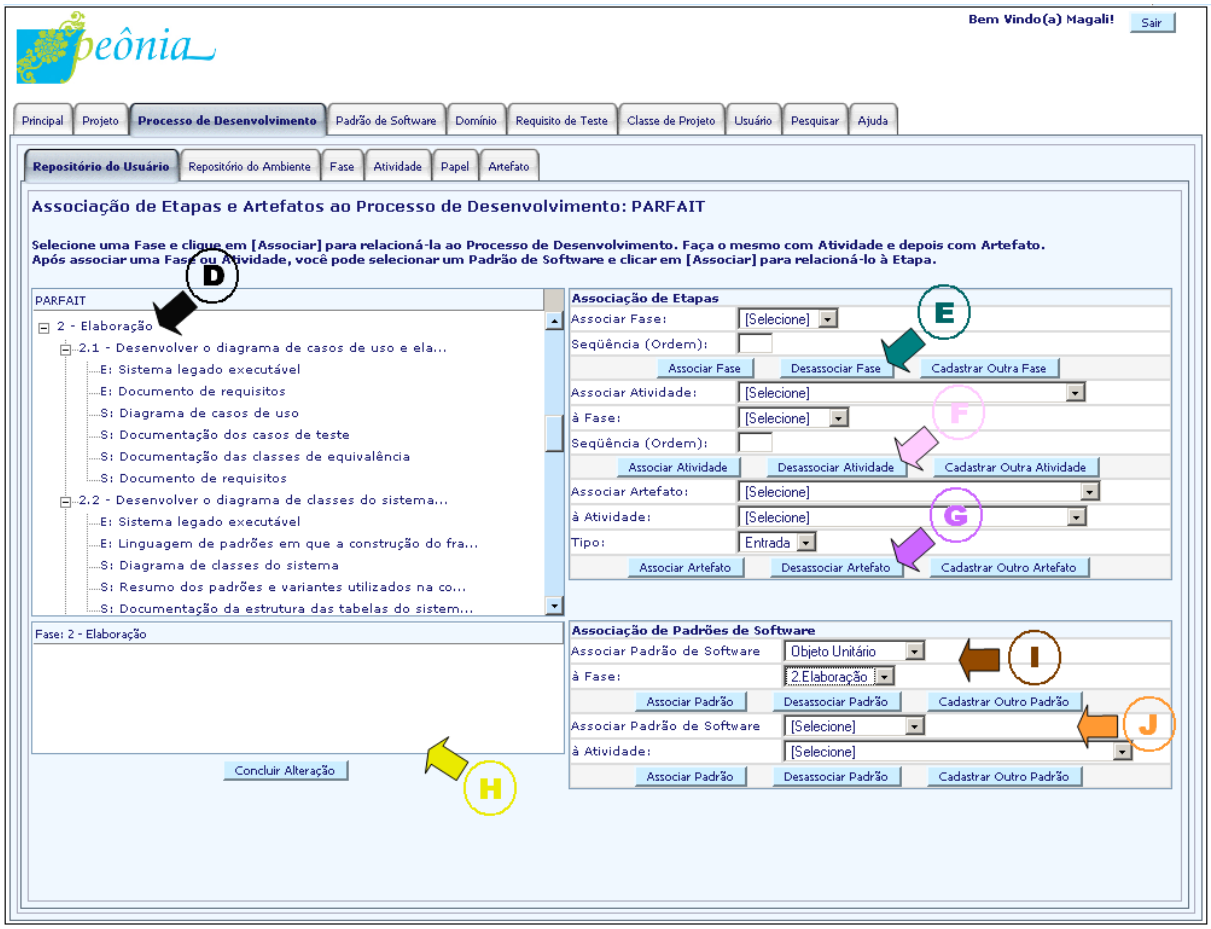

Figura 6.25: Tela exibida ao requisitar a atividade *Alterar Estrutura do Processo de Desenvolvimento.*

Caso a opção sinalizada pela seta "C" azul marinho na Figura 6.24 seja selecionada, a tela da Figura 6.25 é gerada, permitindo a modificação da estrutura do processo de desenvolvimento. O usuário pode optar por remover uma fase do processo (*Atividade: Desassociar Fase do Processo de Desenvolvimento*) selecionando-a na lista, como apontada pela seta "D" preta e escolhendo a opção mostrada pela seta "E" verde na Figura 6.25. Ao desassociar uma fase, todas as suas atividades e artefatos também são desassociados.

As remoções de atividades e artefatos da estrutura do processo de desenvolvimento (*Atividades: Desassociar Atividade da Fase* e *Desassociar Artefato da Atividade*) também são realizadas selecionando-os na lista e escolhendo as opções indicadas pelas setas "F" rosa e "G" lilás, respectivamente. Da mesma forma que a fase, ao desassociar uma atividade, todos os seus artefatos também são desassociados.

Destaca-se que a remoção de fases, atividades e artefatos da estrutura do processo de desenvolvimento não acarreta a exclusão do elemento no ambiente Peônia, ocorrendo apenas a remoção do seu relacionamento com o processo.

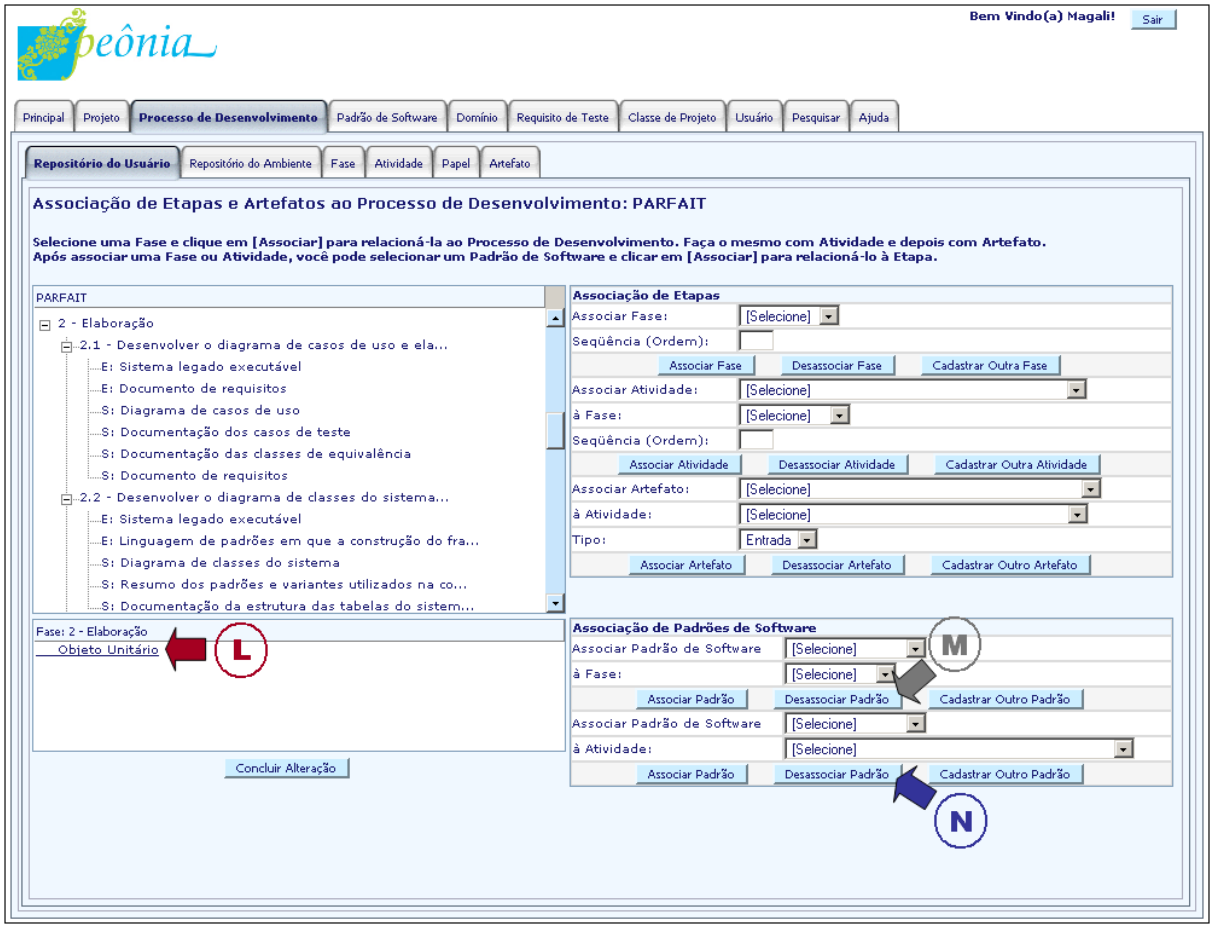

Figura 6.26: Tela exemplificando as atividades *Associar ou Desassociar Padrões de Software às Fases ou Atividades.*

O ambiente Peônia propõe a utilização de padrões de software associado às fases e atividades de um processo de desenvolvimento, facilitando o emprego em projetos. Essa associação é realizada durante as atividades de gerência da estrutura do processo de desenvolvimento (*Cadastrar* ou *Alterar Estrutura do Processo de Desenvolvimento*). Assim, ao selecionar uma fase ou atividade da estrutura do processo de desenvolvimento, caso tenham padrões de software relacionados, esses padrões são exibidos na caixa de texto indicada pela seta "H" amarela. As associações de padrões de software às fases e ativi-

dades (*Atividade: Associar Padrão de Software à Fase* e *Associar Padrão de Software à Atividade*) são realizadas preenchendo os campos do formulário indicado pelas setas "I" marrom e "J" laranja, respectivamente.

O padrão de software associado é adicionado à lista de padrões da fase, como sinalizado pela seta "L" vinho na Figura 6.26. Assim como a desassociação de elementos do processo de desenvolvimento, as remoções de associações de padrões de software (*Atividades: Desassociar Padrão de Software da Fase* ou *Desassociar Padrão de Software da Atividade*) são realizadas selecionando-o na lista e escolhendo a opção mostradas pelas setas "M" cinza e "N" azul marinho.

Com relação a remoção de processos de desenvolvimento, a atividade só é permitida caso o processo não esteja sendo utilizado em projetos do ambiente Peônia. Quando um processo de desenvolvimento é removido, tanto as informações essenciais quanto a sua estrutura são excluídas (*Atividades: Remover Essência do Processo de Desenvolvimento* e *Remover Estrutura do Processo de Desenvolvimento*).

6.2.3 Fase: Desenvolvimento do Projeto

Após serem executadas as atividades da fase de alimentação, projetos podem ser criados e gerenciados. Esses projetos devem utilizar processos de desenvolvimento que contém padrões de software associados, estimulando usuários a empregá-los. As atividades relacionadas a gerência de projetos e pesquisa de elementos do ambiente Peônia são executadas durante a fase de *Desenvolvimento do Projeto*, ilustradas no início da Seção 6.2 pela Figura 6.2.

Na Seção 6.2.3.1, são listadas as atividades de gerência de projeto, inclusive todas as atividades relacionadas à utilização de processo de desenvolvimento. Na Seção 6.2.3.2, é explicada a funcionalidade de abertura de projeto, necessária para a execução do projeto. Na Seção 6.2.3.3, são explicadas as atividades de adaptação de um processo de desenvolvimento ao contexto do projeto do usuário e a visualização das informações adaptadas dos elementos desse processo. Na Seção 6.2.3.4, são explicadas as funcionalidades ligadas ao emprego de padrões de software na execução de fases, atividades ou artefatos. Na Seção 6.2.3.5, são listadas as operações de gerência de diagramas. Na Seção 6.2.3.6, é explicada abertura de diagrama, assim como as atividades que podem ser executadas no gerenciamento da estrutura de um diagrama aberto. Na Seção 6.2.3.7, é dada uma visão de como é realizada a verificação da realização adequada e correta do projeto desenvolvido pelo usuário. Por fim, na Seção 6.2.3.8, são listadas as opções de pesquisa disponíveis para agilizar a localização de elementos, melhorando o apoio fornecido ao desenvolvimento de projetos.

6.2.3.1 Atividades: Cadastrar, Alterar, Remover, Visualizar ou Listar Projetos

Um projeto é cadastrado e identificado unicamente pelo campo "Nome" (*Atividade: Cadastrar Projeto*). A associação de um processo de desenvolvimento não é obrigatória para a conclusão do cadastro de um projeto, no entanto, para executá-lo é necessário que essa associação seja realizada (*Atividade: Associar Processo de Desenvolvimento ao Projeto*).

Na Figura 6.27, a seta "A" vinho indica onde deve ser realizada a escolha do processo a ser utilizado no projeto. Nesse campo, são listados todos os processos de desenvolvimento do usuário e os compartilhados, sendo que os últimos citados são marcados com o caractere asterisco (*). Ao ser selecionado um processo, um resumo sobre o processo é apresentado, como sinalizado pela "B" cinza. Destaca-se que todo projeto é cadastrado com a situação "Executando", como mostrado pela seta "C" azul marinho, e tem esse estado alterado de acordo com a evolução do projeto. Essa situação pode ser modificada manualmente pelo usuário durante a atividade de alteração de projeto (*Atividade: Alterar Projeto*), ou requerendo a verificação da conformidade da execução do projeto (*Atividade: Finalizar Projeto*), detalhada na Seção 6.2.3.7.

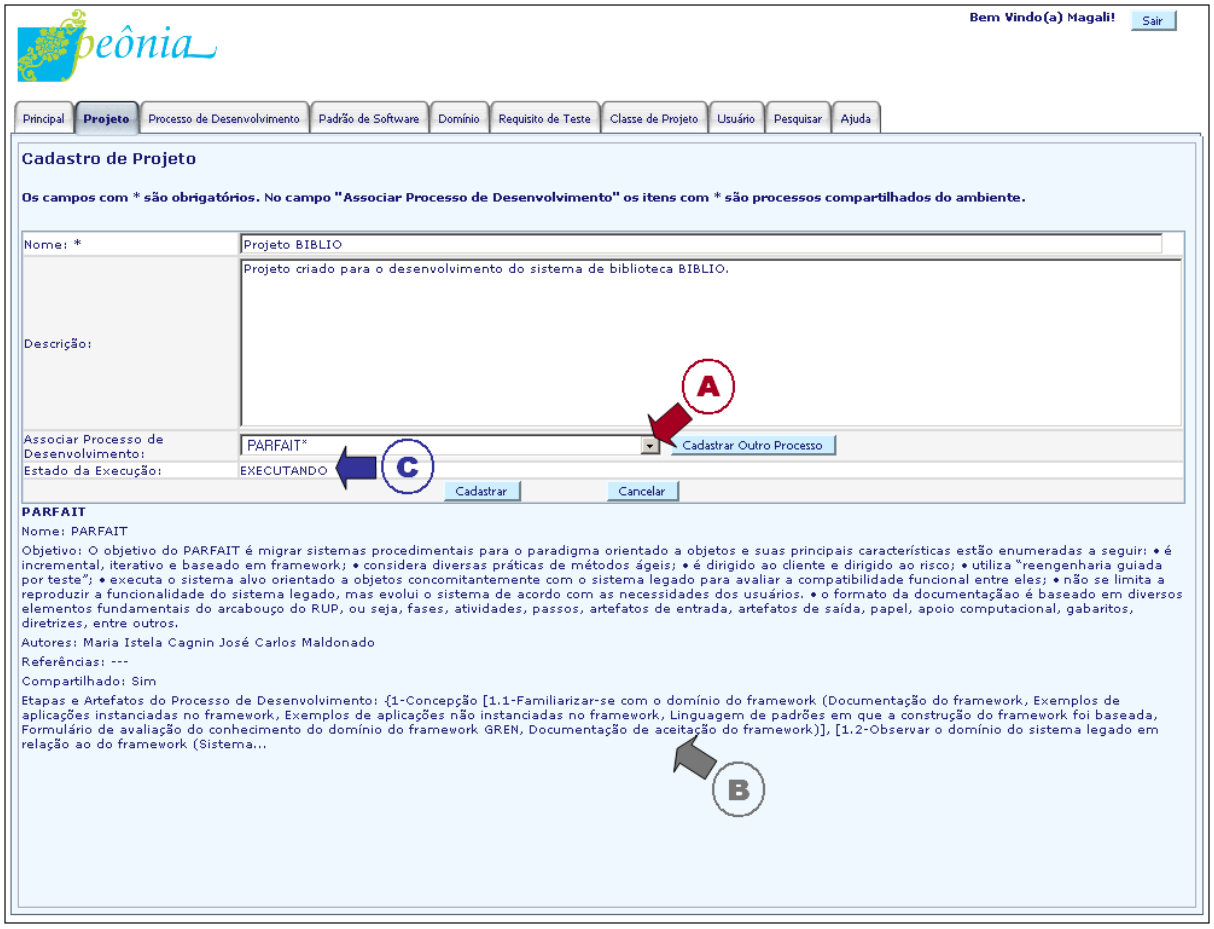

Figura 6.27: Tela exibida ao requisitar a atividade *Cadastrar Projeto*.

Ao associar um processo de desenvolvimento ao projeto, automaticamente são criadas instâncias do processo de desenvolvimento associado (*Atividade: Instanciar Processo de Desenvolvimento do Projeto*) e das suas fases, atividades e artefatos (*Atividades: Instanciar Fases do Processo de Desenvolvimento, Instanciar Atividades das Fases* e *Instanciar Artefatos das Atividades*). Essas instâncias são utilizadas para identificar separadamente cada execução de um processo de desenvolvimento dentro do ambiente Peônia.

Após o cadastro, o projeto é adicionado em uma lista (*Atividade: Listar Projetos*), como mostrado pela seta "A" vinho na Figura 6.28, onde deve ser selecionado para que seja aberto e possa ser executado (*Abrir Projeto*). A opção para abertura de projeto é indicada pela seta "B" cinza na Figura 6.28 e é explicada na Seção 6.2.3.2. Ao selecionar um projeto, as suas informações são apresentadas no quadro apontado pela seta "C" azul marinho (*Atividade: Visualizar Projeto*).

Ao remover um projeto (*Atividade: Remover Projeto*), tanto as informações cadastrais quanto as instâncias criadas para controlar a execução de um processo de desenvolvimento são removidas definitivamente do ambiente Peônia (*Atividades: Remover Instância do Processo de Desenvolvimento do Projeto, Remover Instâncias das Fases do Processo de Desenvolvimento, Remover Instâncias das Atividades das Fases* e *Remover Instâncias dos Artefatos da Atividade*), não podendo ser recuperadas.

Figura 6.28: Tela exibida ao requisitar a atividade *Listar Projetos*.

6.2.3.2 Atividade: Abrir Projeto

Após ser cadastrado, um projeto pode ser aberto (*Atividade: Abrir Projeto*) e as etapas e artefatos do processo de desenvolvimento associado podem ser adaptados e executados para a criação do software desejado. Ao abrir um projeto, as fases, atividades e artefatos

do processo de desenvolvimento associado são exibidos na parte superior esquerda da tela, indicada pela seta "A" vinho na Figura 6.29.

A parte inferior esquerda, sinalizada pela seta "B" cinza, é utilizada para a exibição de padrões de software do ambiente Peônia. Ao abrir o projeto, todos os padrões do repositório são exibidos. Caso seja selecionada uma fase ou atividade, apenas os padrões de software associados à etapa escolhida são exibidos. A qualquer momento o usuário pode visualizar todos os padrões do ambiente Peônia selecionando a opção indicada pela seta "C" azul marinho (*Atividade: Listar Padrões de Software*). Ao passar o mouse sobre um padrão, um resumo sobre os seus campos obrigatórios é apresentado. A visualização completa pode ser realizada selecionando o atalho indicado pela seta "D" preta.

Nas abas do lado esquerdo, apontada pela seta "E" verde, são efetuadas as atividades referentes a execução e manipulação das informações do projeto, explicadas nas próximas seções.

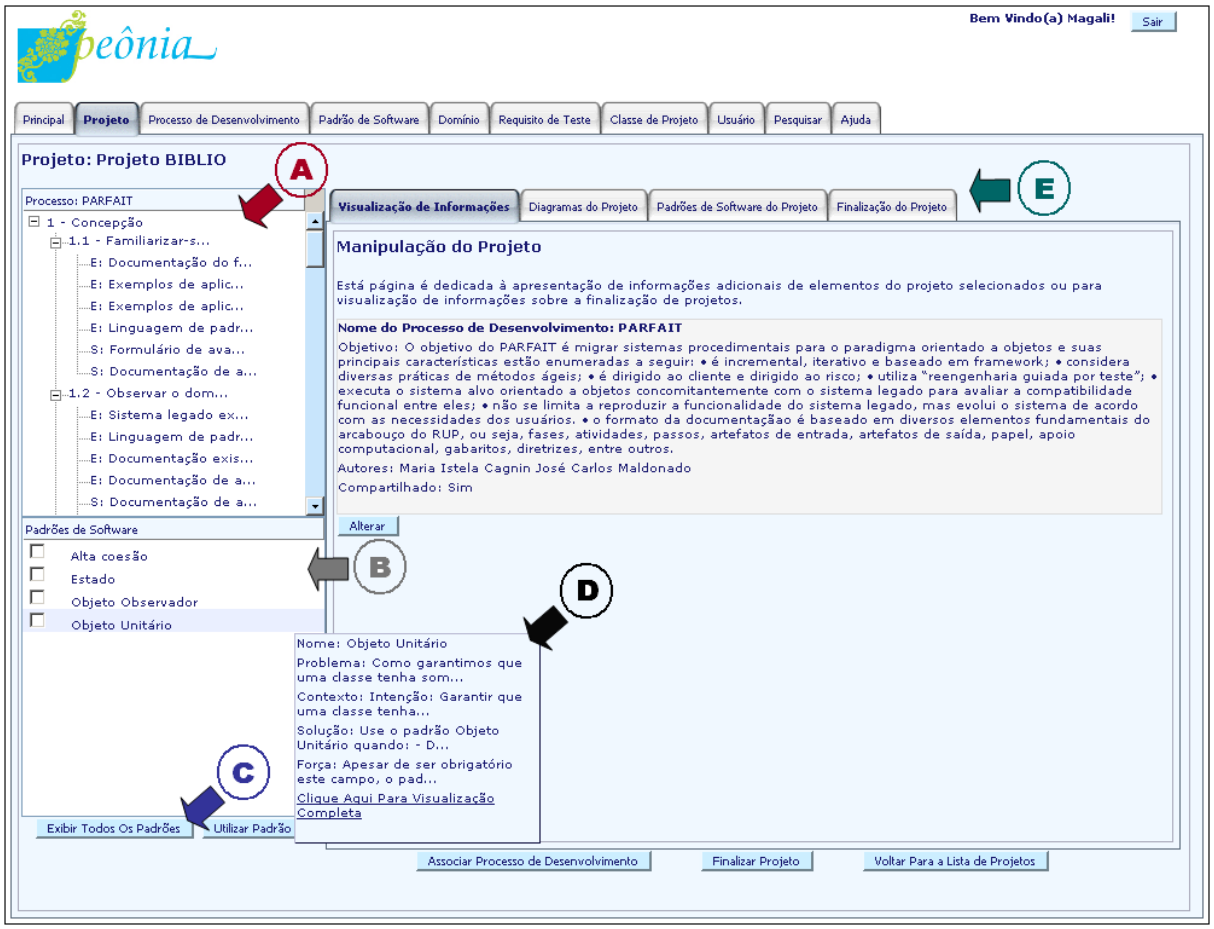

Figura 6.29: Tela exibida ao requisitar a atividade *Abrir Projeto*.

6.2.3.3 Atividades: Adaptar Processo de Desenvolvimento Instanciado e Visualizar Dados Sobre Processo de Desenvolvimento Adaptado

Como mencionado na Seção 2.3, um processo de desenvolvimento flexível deve ser adaptável. Assim, no ambiente Peônia o usuário pode selecionar em cada um dos seus projetos o que deseja efetivamente utilizar num processo de desenvolvimento, alterando a obrigatoriedade das fases, atividades e artefatos (*Atividade: Adaptar Processo de Desenvolvimento Instanciado*), como ilustrado na Figura 6.30.

Essa alteração de obrigatoriedade é necessária para que o ambiente Peônia possa finalizar um projeto sem produzir erros por causa de elementos obrigatórios no processo de desenvolvimento cadastrado, mas que são opcionais no projeto executado. Outra razão da adaptação é evitar que o usuário deixe de ser alertado se uma etapa do processo for pulada, ou um artefato essencial deixar de ser produzido, pois elementos opcionais no processo cadastrado podem ser transformados em obrigatórios no projeto executado. Assim, na finalização do projeto essas falhas são detectadas.

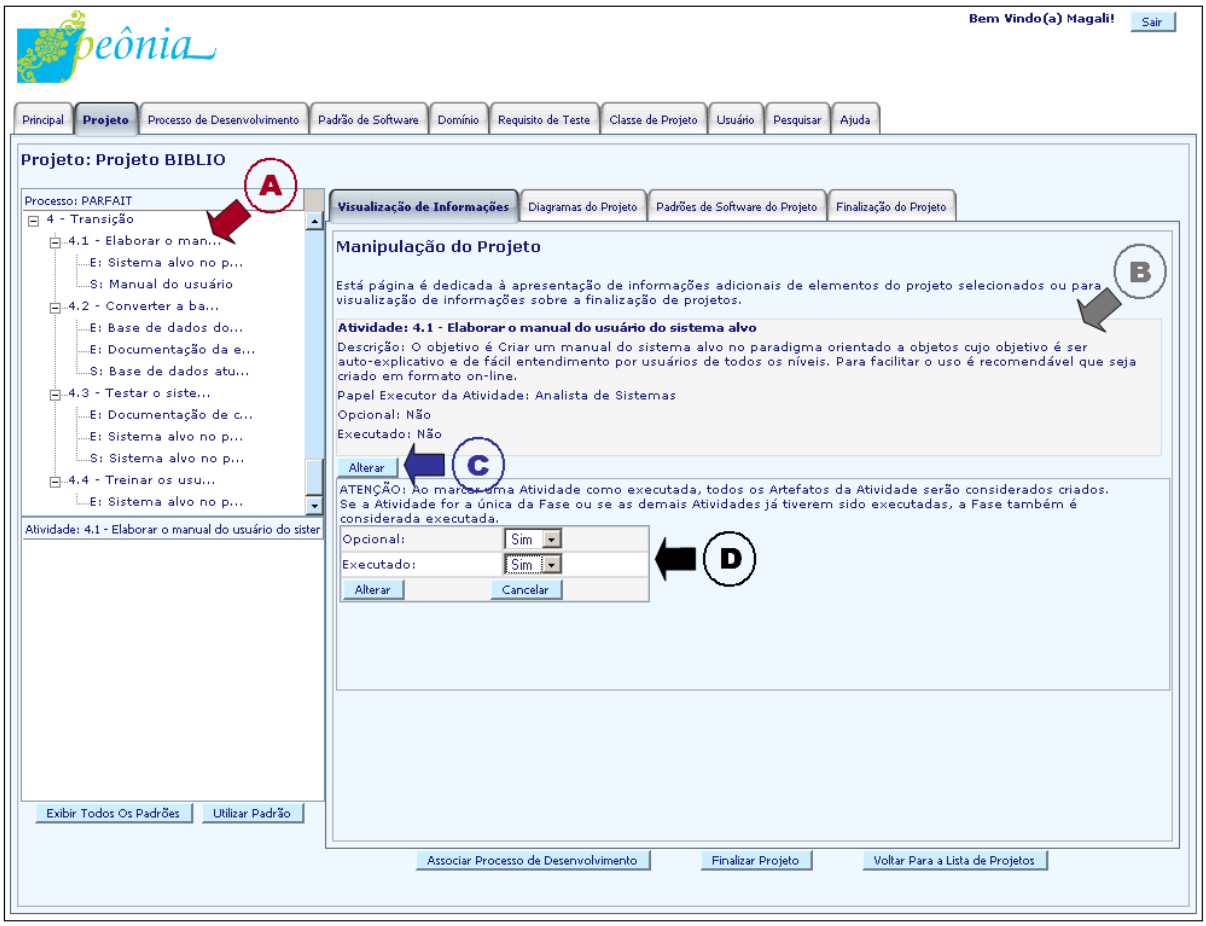

Figura 6.30: Tela exibida nas atividades *Adaptar Processo de Desenvolvimento Instanciado* e *Visualizar Dados Sobre Processo de Desenvolvimento Adaptado*.

O usuário deve selecionar o elemento que terá a obrigatoriedade alterada, como por exemplo, o apontado pela seta "A" vinho na Figura 6.30. Os dados do elemento selecionado são exibidos (*Atividade: Visualizar Dados Sobre Processo de Desenvolvimento Adaptado*), como indicado pela seta "B" cinza, e a opção de alteração, sinalizada pela seta "C" azul marinho, deve ser selecionada para que o formulário para modificação da obrigatoriedade seja apresentado e preenchido, como indicado pela seta "D" preta.

6.2.3.4 Atividades: Utilizar ou Remover o Uso do Padrão de Software nas Fases, Atividades ou Artefatos

Para fases, atividades e artefatos, o uso de padrões pode ser registrado no ambiente Peônia como documentação. O emprego de padrões de software em fases, atividades, ou artefatos pode ser realizado selecionado o elemento na estrutura do processo de desenvolvimento, indicada pela seta "A" vinho na Figura 6.31, escolhendo os padrões a serem utilizados, como os sinalizados pela seta "B" cinza, e executando a opção mostrada pela seta "C" azul marinho (*Atividades: Utilizar ou Remover o Uso do Padrão de Software na Fase*, *Utilizar ou Remover o Uso do Padrão de Software na Atividade* ou *Utilizar ou Remover o Uso do Padrão de Software no Artefato*).

Em seguida, na aba "Padrões de Software do Projeto", sinalizada pela seta "D" preta, é apresentada a fase, artefato, ou atividade escolhida, apontada pela seta "E" verde, e os padrões de software que serão empregados, indicada pela seta "F" rosa, para que o usuário confirme e conclua a operação.

Como nesse primeiro momento é oferecido suporte apenas para o desenvolvimento de diagramas, a utilização de padrão de software em artefatos também pode ser realizada com a criação desses diagramas (*Atividade: Utilizar ou Remover o Uso do Padrão de Software no Artefato*). Informações sobre como um padrão de software é inserido e removido de um diagrama são apresentadas na Seção 6.2.3.6.

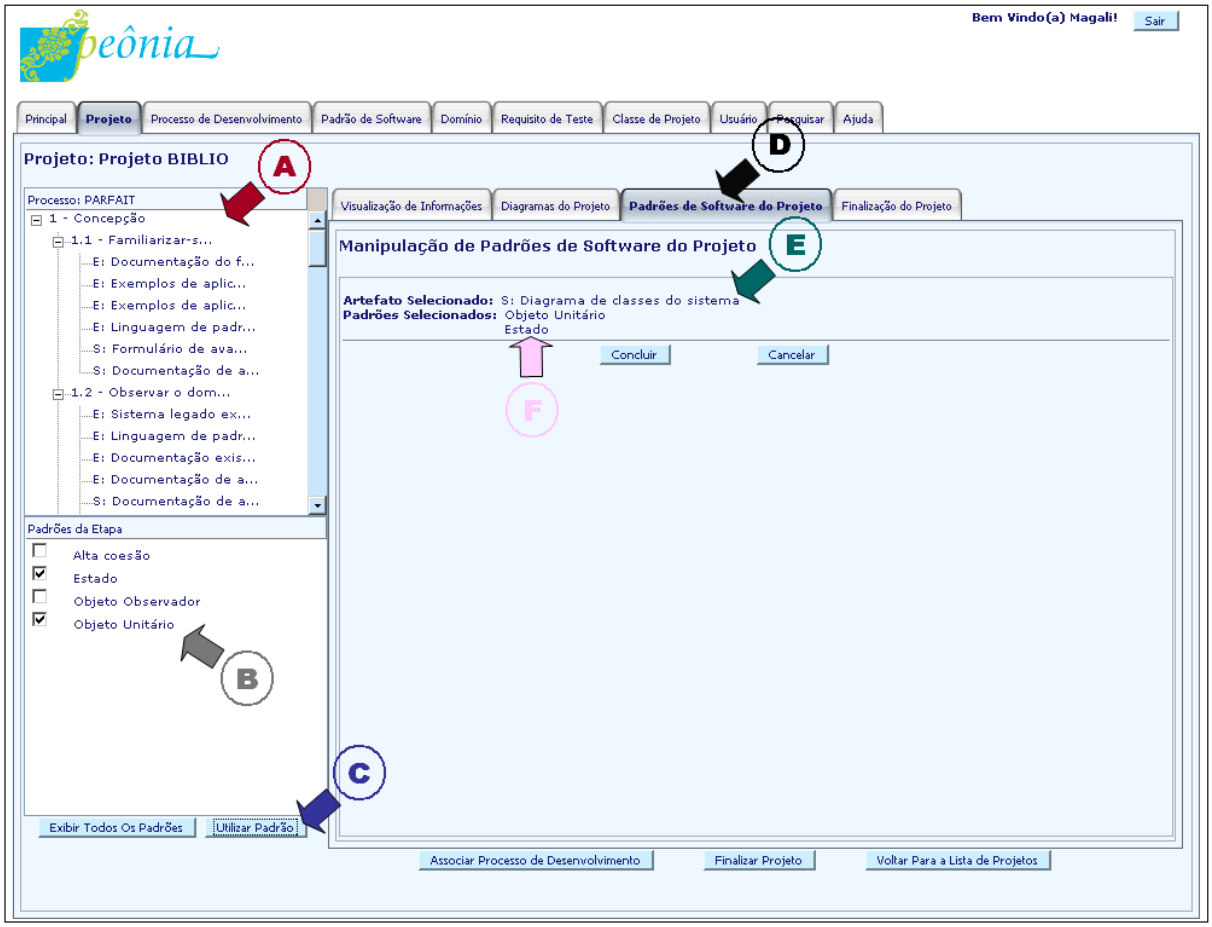

Figura 6.31: Tela exibida ao requisitar as atividades *Utilizar Padrão de Software na Fase* ou *Utilizar Padrão de Software na Atividade*.

6.2.3.5 Atividades: Cadastrar, Alterar, Remover, Visualizar ou Listar Diagramas

Diagramas são criados durante o projeto de aplicações, refinando o levantamento realizado durante a etapa de análise e facilitando a implementação. Assim, o ambiente Peônia fornece o apoio para o desenvolvimento de diagramas. Nessa primeira versão, apenas o apoio a criação de modelos conceituais e diagramas de classes é fornecido, no entanto, a definição da estrutura dos diagramas foi realizado de tal maneira que facilita a adição futura de outros tipos de diagramas. Detalhes sobre como foram projetados os conceitos ligados ao elemento *Diagrama* são explicados na Seção 7.6.

Um diagrama é cadastrado (*Atividade: Cadastrar Diagrama*) informando obrigatoriamente o "Nome", "Tipo de Diagrama" e "Tipo de Artefato". O nome é utilizado para que o usuário localize-o com maior facilidade. É permitido que diagramas possuam nomes repetidos. Com relação ao tipo de diagrama, é necessário escolher entre os dois tipos oferecidos atualmente pelo ambiente Peônia, ou seja, modelo conceitual ou diagrama de classes, como indicado pela seta "A" vinho na Figura 6.32. Também é necessário seleci-

onar os tipos de artefatos para classificar o diagrama, permitindo reconhecer quais dos artefatos do processo de desenvolvimento são executados com a criação do diagrama.

Concluindo o cadastro, o diagrama é inserido e exibido em uma lista (*Atividade: Listar Diagramas*), onde deve ser selecionado para que possa ser visualizado, alterado, removido ou aberto (*Atividades: Visualizar, Alterar, Remover* ou *Abrir Diagrama*). Ao remover um diagrama, todas as informações cadastrais e estruturais são removidas do ambiente Peônia. Somente no diagrama aberto é realizada efetivamente a modelagem de aplicações com a inserção de padrões de software e classes de projeto. Mais informações a respeito da manipulação da estrutura dos diagramas são dadas na Seção 6.2.3.6.

Planeja-se em trabalhos futuros adicionar as funcionalidade para que esses diagramas possam ser exportados e importados de outras ferramentas, como descrito na Seção 8.4.

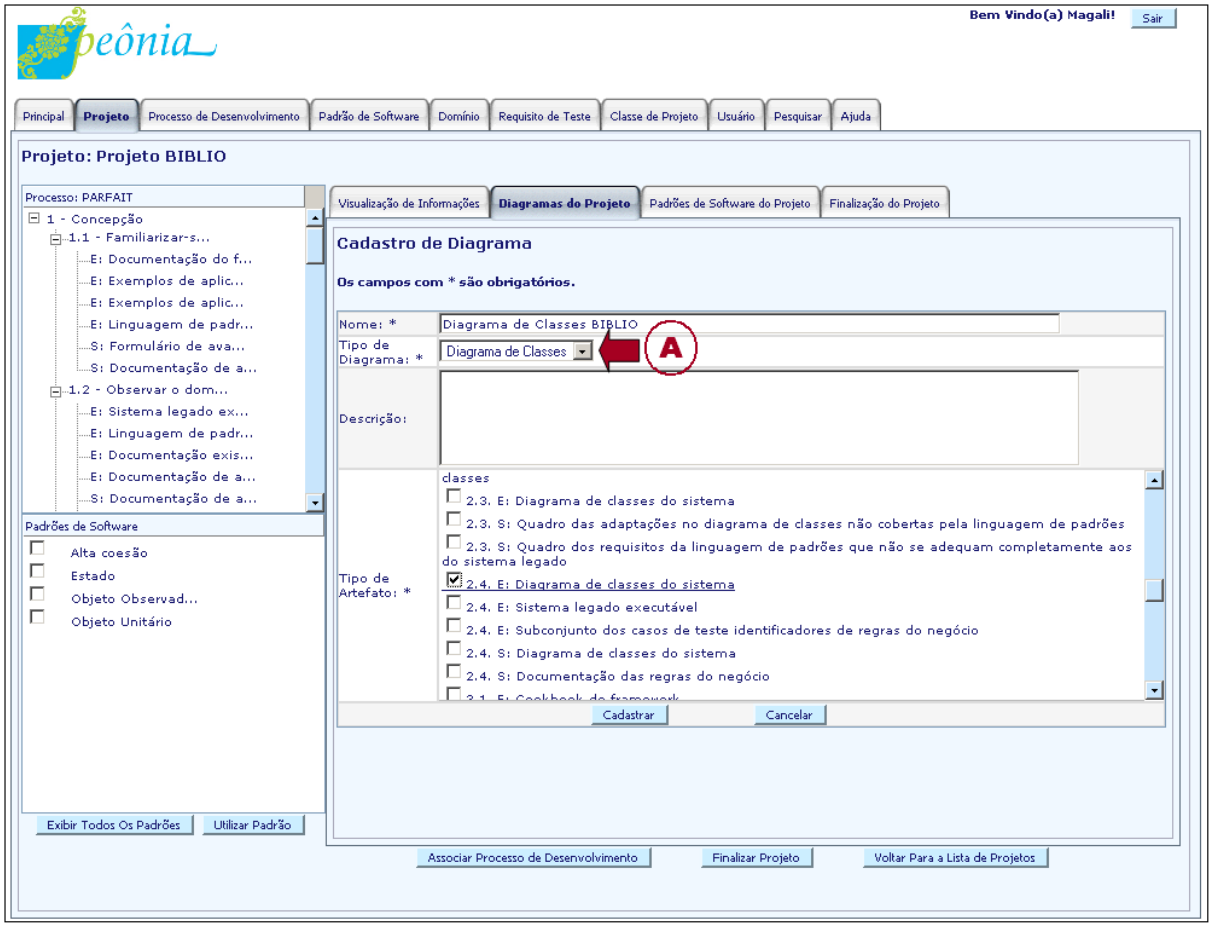

Figura 6.32: Tela exibida ao requisitar a atividade *Cadastrar Diagrama*.

6.2.3.6 Atividade: Abrir Diagrama

Após cadastrar informações caracterizando o diagrama, o usuário pode abrir (*Atividade: Abrir Diagrama*) e alterar a sua estrutura adicionando ou removendo classes de projeto

(*Atividades: Utilizar ou Remover o Uso da Classe de Projeto no Diagrama*) e padrões de software (*Atividades: Utilizar ou Remover o Uso do Padrão de Software no Diagrama*). Por causa das restrições de cronograma, não foi possível implementar a manipulação visual das classes de projeto de um diagrama. Assim, atualmente essas informações são enviadas por meio de formulários, cujos campos permitem informar sete tipos de relacionamento entre classes de projeto: associação, agregação, composição, especialização, generalização, herança e realização.

As classes de projeto podem ser inseridas no diagrama diretamente ou relacionadas a outras classes de projeto (*Atividade: Associar Classes de Projeto*) por meio do preenchimento do formulário indicado pela seta "A" vinho na Figura 6.33.

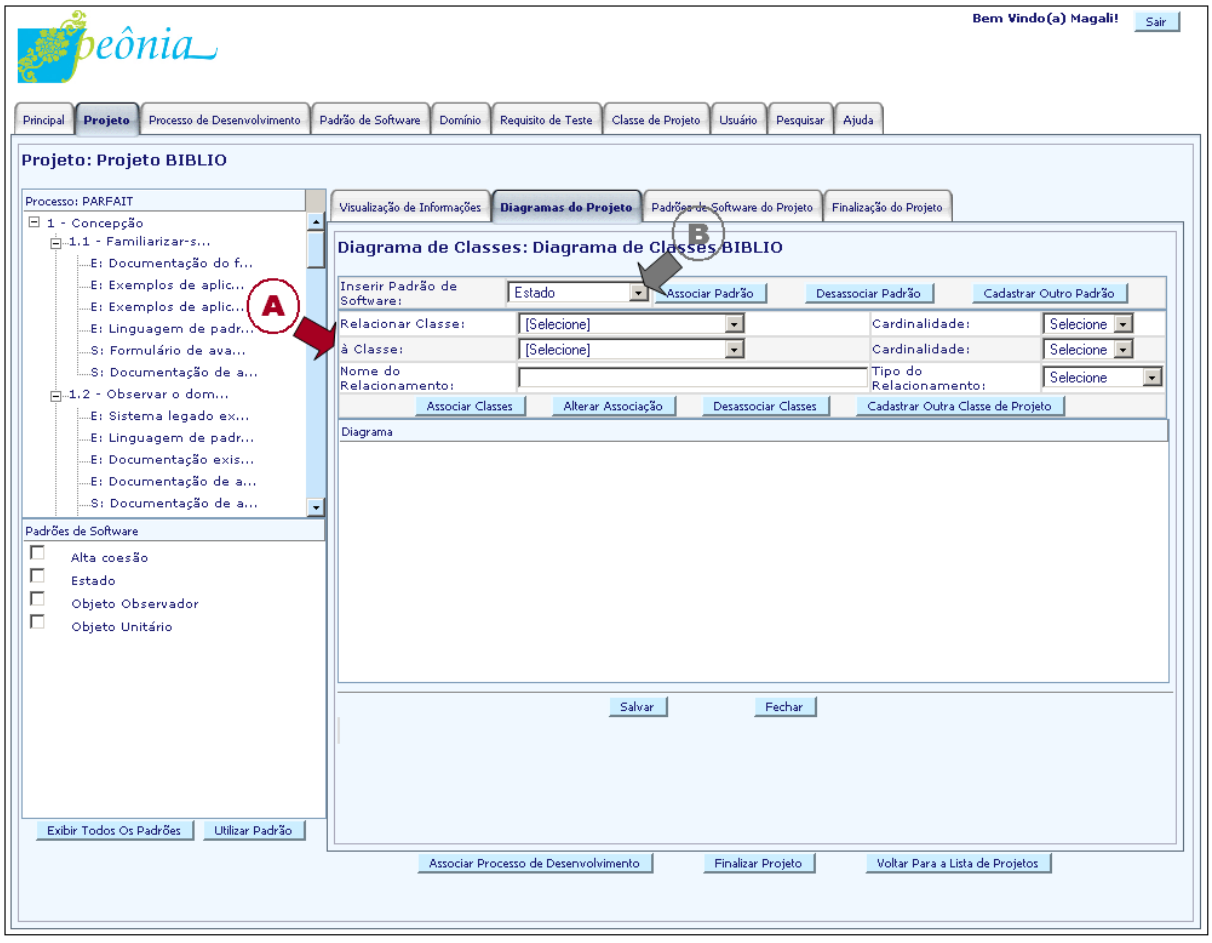

Figura 6.33: Tela exibida ao requisitar a atividade *Abrir Diagrama*.

Ao selecionar um padrão de software no campo sinalizado pela seta "B" cinza na Figura 6.33, duas situações são possíveis: o padrão pode possuir estrutura da solução, ou não. No primeiro caso, ilustrado na Figura 6.34, uma lista das classes de projeto e dos relacionamentos entre essas classes, como mostrada pela seta "C" azul marinho, é exibida para que o usuário possa adaptar o nome dessas classes de projeto ou dos relacionamentos

para o contexto do seu projeto (*Atividade: Instanciar Padrão de Software*). Ao selecionar o elemento que será modificado, as informações atuais são apresentadas no formulário, indicado pela seta "D" preta, onde também ocorre a alteração. Ao confirmar o emprego do padrão de software, uma instância para cada componente da estrutura da solução é criada com o novo nome informado. Para os componentes inalterados, a instância é criada com os nomes do componentes da estrutura da solução cadastrada durante a fase de *Alimentação do Peônia* do ambiente Peônia.

No entanto, se o padrão não possuir estrutura da solução, condição necessária para ser empregado em um diagrama, uma mensagem informativa é apresentada ao usuário que tem a opção de documentar o emprego do padrão de software no artefato (*Atividade: Utilizar ou Remover o Uso do Padrão de Software no Artefato*).

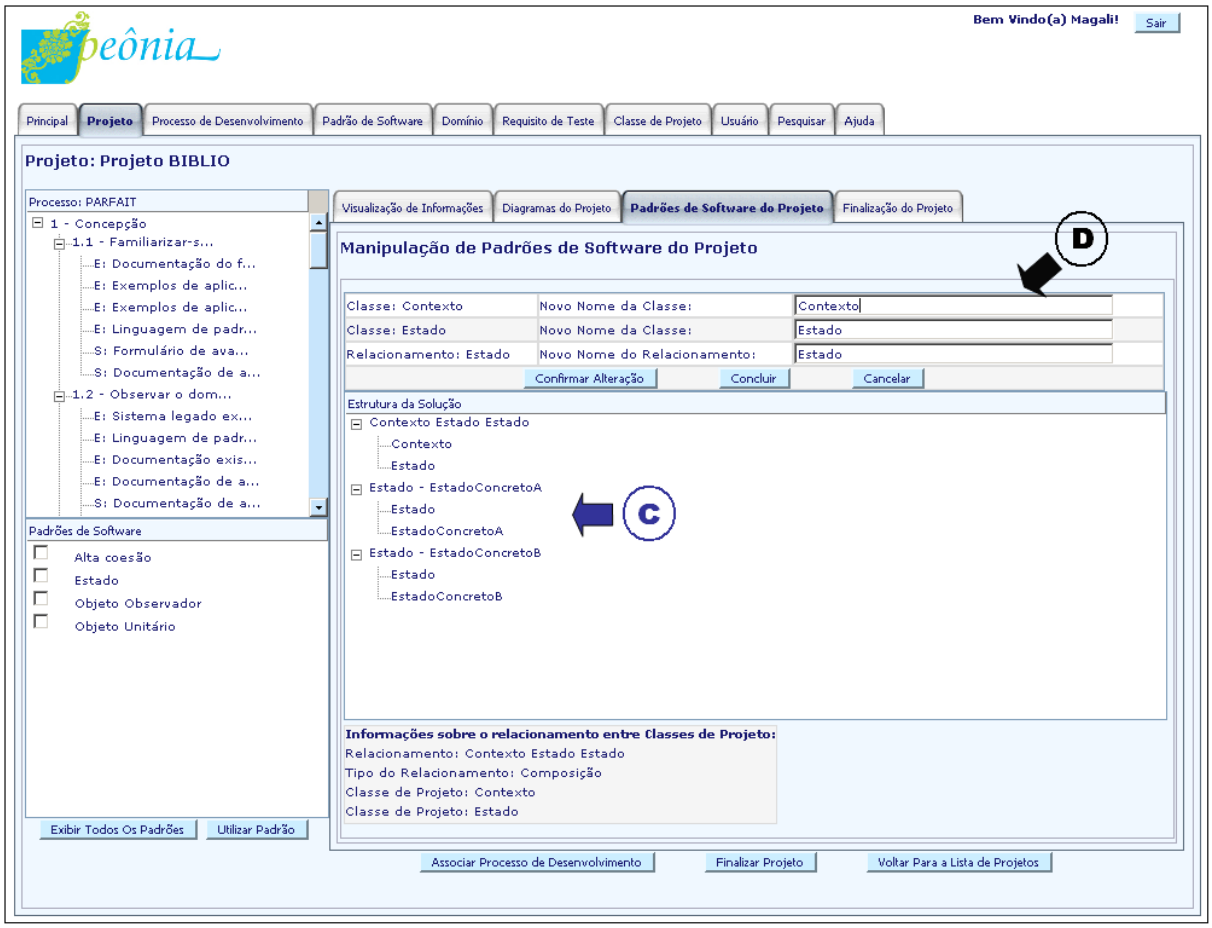

Figura 6.34: Tela exibida para execução da atividade *Instanciar Padrão de Software*.

Na Figura 6.35, é apresentado o diagrama após ser requerido o uso de um padrão de software com estrutura da solução e com o nome de alguma de suas classes de projeto adaptadas ao contexto do projeto, como mostrada pela seta "E" verde. As informações apresentadas no diagrama podem ser modificadas pelo usuário, no entanto, a retirada

de um dos componentes da estrutura da solução implica na remoção de toda a instância do padrão de software utilizado no diagrama (*Atividade: Remover Instância do Padrão de Software*). Essa decisão foi tomada, pois a remoção de um elemento da estrutura do padrão de software pode descaracterizar a solução, com a possibilidade de não resolver o problema ao qual é proposto. Assim, sem alguns elementos da sua estrutura da solução não é possível afirmar que foi empregado o padrão de software.

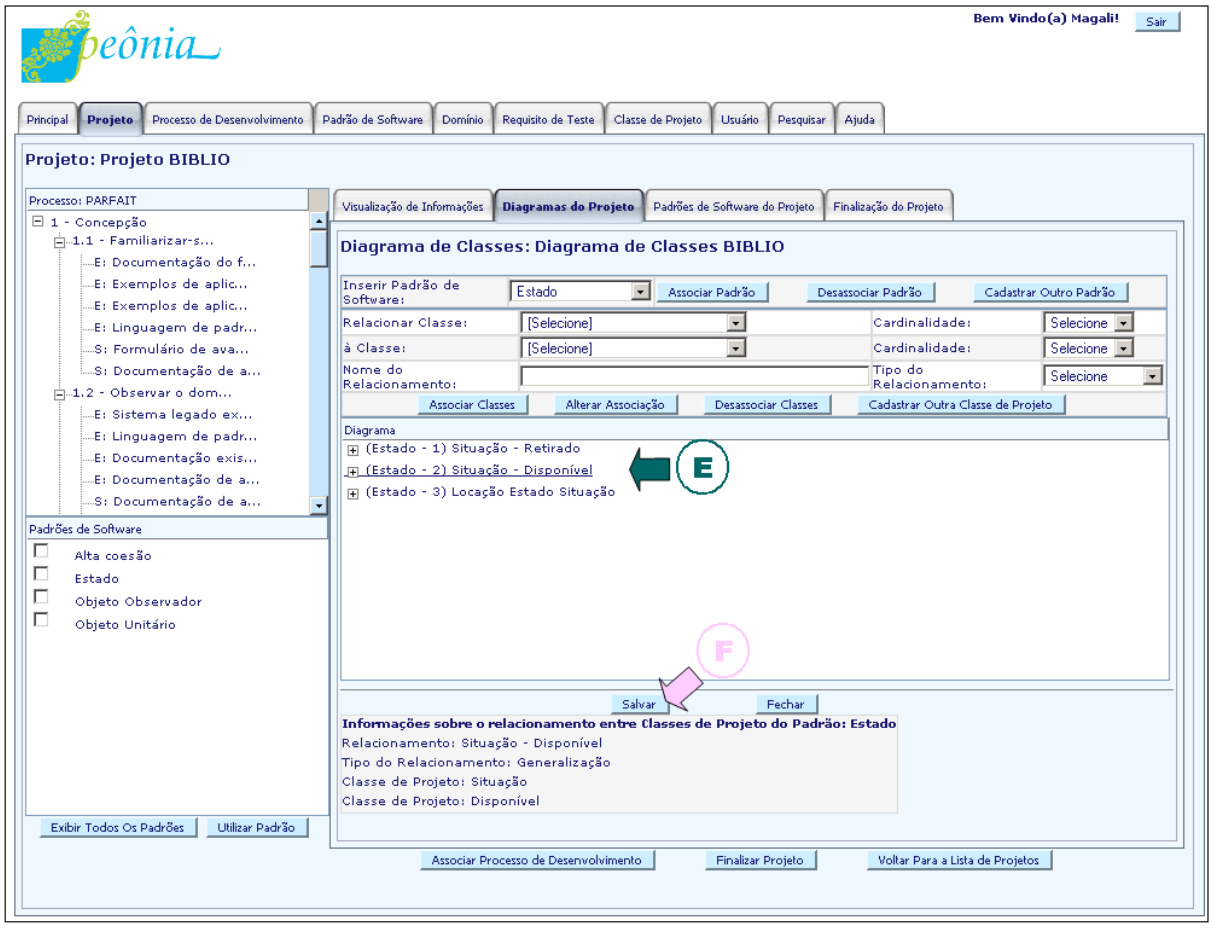

Figura 6.35: Tela exibida após executar a atividade *Instanciar Padrão de Software*.

Requisitos de teste cadastrados podem ser utilizados no diagrama, independentemente de estarem ou não associados aos padrões de software (*Atividades: Utilizar Requisito de Teste Não Associado ao Padrão de Software* ou *Utilizar Requisito de Teste Associado ao Padrão de Software*). Assim como os outros elementos do diagrama, requisitos de teste podem ser removidos a qualquer momento (*Atividades: Remover o Uso do Requisito de Teste Não Associado ao Padrão de Software* ou *Remover o Uso do Requisito de Teste Associado ao Padrão de Software*).

É importante ressaltar que todas as informações inseridas na estrutura do diagrama são armazenadas no ambiente Peônia somente quando a opção para salvar o diagrama é selecionada (*Atividade: Salvar Diagrama*), mostrada pela seta "F" rosa na Figura 6.35.

6.2.3.7 Atividade: Finalizar Projeto

Após executar todas as fases e atividades e gerar todos os artefatos necessários ao projeto, o usuário pode finalizá-lo (*Atividade: Finalizar Projeto*), ou seja, informar que concluiu o desenvolvimento da sua aplicação. Essa atividade pode ser realizada selecionando as opções indicadas pelas setas "A" vinho e "B" cinza na Figura 6.36.

Figura 6.36: Tela do resultado produzido ao requisitar a atividade *Finalizar Projeto*.

Ao acionar a atividade de finalização de projeto, o ambiente Peônia realiza uma verificação automática averiguando se as fases, atividades e artefatos obrigatórios foram executados corretamente. Caso algum elemento obrigatório esteja faltando, uma mensagem alertando o usuário é produzida na aba de "Finalização de Projeto", como mostrado pela seta "C" azul marinho.

Se todos os elementos obrigatórios tiverem sido executados, o ambiente Peônia altera a situação do projeto para "Finalizado" e apresenta essa informação para o usuário. Caso existam fases, atividades e artefatos opcionais não executados, além da mensagem de sucesso sobre a finalização do projeto, também são listados os elementos opcionais não executados para que o usuário possa ter uma visão completa do que foi realmente feito no seu projeto.

6.2.3.8 Atividade: Pesquisar Elementos

Para que o usuário possa localizar rapidamente informações no ambiente Peônia, é oferecida a funcionalidade de pesquisa em dez tipos de elementos (*Atividades: Pesquisar Artefato, Atividade, Diagrama, Domínio, Fase, Padrão de Software, Papel, Processo de Desenvolvimento, Projeto* ou *Requisito de Teste*). O tipo de elemento a ser procurado deve ser escolhido na opção indicada pela seta "A" vinho na Figura 6.37.

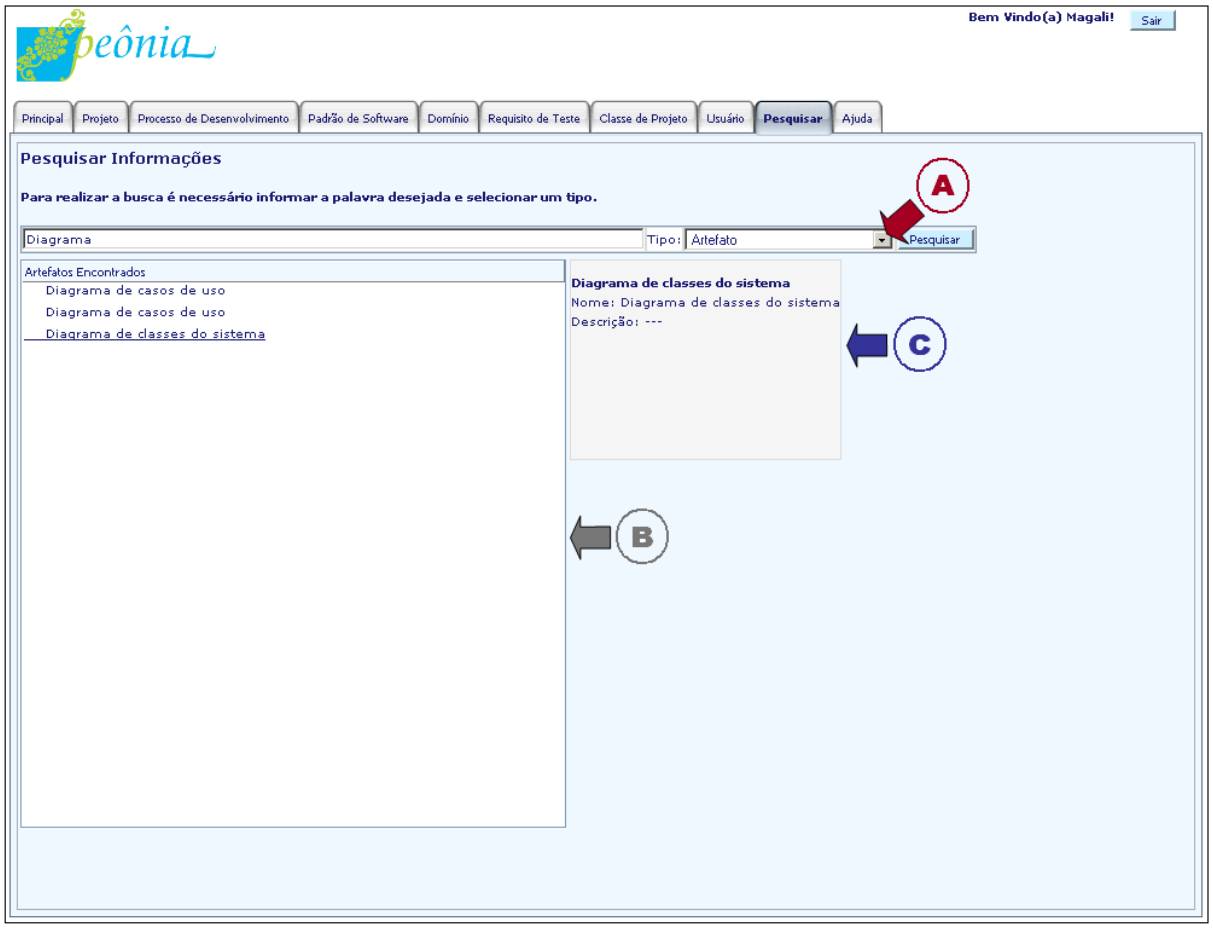

Figura 6.37: Tela exibida ao requisitar a atividade *Pesquisar*.

Para realizar a consulta, o usuário deve informar o nome, ou parte dele, para que o ambiente Peônia retorne uma lista com os elementos desejados no quadro mostrado pela

seta “B” cinza. Informações a respeito dos elementos retornados na busca são exibidas quando selecionados, como apontado pela seta “C” azul marinho.

6.3 Considerações Finais

Para auxiliar usuários no emprego de padrões de software durante todas as fases de um processo de desenvolvimento, o ambiente Peônia foi produzido. Após a análise do domínio, diversas funcionalidades foram levantadas e, por causa da restrição do cronograma, 168 foram priorizadas.

A análise do domínio levou em consideração o desenvolvimento de uma aplicação que auxiliasse usuários no emprego de padrões de software e, por isso, as 168 funcionalidades necessárias para a utilização do ambiente Peônia foram listadas e priorizadas. Neste capítulo, essas funcionalidades foram detalhadas, sendo que o projeto do ambiente Peônia será explicado no Capítulo 7.

Por ser um projeto incremental, melhorias e outras funcionalidades serão adicionadas futuramente, como por exemplo, as atividades para *Exportar* e *Importar Diagramas*, necessárias para a integração com outras ferramentas de apoio ao desenvolvimento de aplicações. Detalhes sobre trabalhos futuros são dados na Seção 8.4.

Capítulo

7

Ambiente Peônia - Projeto

7.1 Considerações Iniciais

As funcionalidades do ambiente Peônia, listadas no Capítulo 6, foram criadas a partir da análise do problema que usuários possuem em empregar padrões de software no desenvolvimento das suas aplicações. Neste capítulo são explicadas as atividades executadas e artefatos criados durante o projeto do ambiente Peônia.

Na Seção 7.2, é descrita a análise do problema encontrado, ou seja, a dificuldade do emprego de padrões de software no desenvolvimento de aplicações, e são resumidos os modelos desenvolvidos durante o projeto do ambiente Peônia. Na Seção 7.3, são dados exemplos da estrutura dos documentos criados durante o levantamento de requisitos. Na Seção 7.4, é explicado o plano de projeto criado para o desenvolvimento do ambiente Peônia. Na Seção 7.5, é detalhada a arquitetura adotada e as tecnologias utilizadas em cada uma das camadas. Na Seção 7.6, é explicada a estrutura de tabelas adotada para armazenar as informações do ambiente Peônia. Na Seção 7.7, são apresentadas algumas figuras de diagramas de casos de uso, atividades e seqüência, exemplificando os modelos criados durante a análise e projeto. Na Seção 7.8, são mostrados alguns modelos UX criados para definir as telas e mapas de navegação. Na Seção 7.9, são listados os conceitos, é dada uma visão sobre como estão distribuídos os elementos que compõem o ambiente Peônia e como são acessadas as suas funcionalidades. Na Seção 7.10, são feitas considerações a respeito da implementação e sobre como foi realizado o teste do

ambiente Peônia. Por fim, na Seção 7.11, são apresentadas as considerações finais sobre este capítulo.

7.2 Análise do Domínio

Segundo o estudo de caso realizado por Bianchini (2007), dos processos e métodos estudados, o processo proposto por (Conallen, 2002) e o *WAE*, detalhados na Seção 2.3.1, são os mais abrangentes com relação às necessidades exigidas pelas características específicas de aplicações *Web*. Também são os mais adequados para a implementação de sistemas com muita interação com o banco de dados, como é o caso de portais, utilizados no estudo de caso realizado por Bianchini (2007). Como o ambiente Peônia, assim como portais, executa muitas operações no banco de dados, para o projeto foram utilizados o processo proposto por (Conallen, 2002) e método *WAE*.

Primeiramente, foi realizada uma *Análise do Problema Percebido* ao utilizar padrões de software no desenvolvimento de aplicações. Por causa da dificuldade de usuários, inclusive da desenvolvedora deste projeto de mestrado, na localização, memorização e aplicação de padrões de software existente na literatura, foi realizado um levantamento das tecnologias existentes que auxiliam o emprego de padrões de software. Em seguida, observando a utilidade de um ambiente *Web* que fornecesse apoio ao emprego de padrões de software e à associação de requisitos de teste a esses padrões tentando minimizar o tempo despendido nas atividades de VV&T, foi proposto o desenvolvimento do ambiente Peônia. A análise do problema encontrado, a revisão bibliográfica detalhada sobre os assuntos envolvidos no problema e o a solução proposta foram formalizados em Chan (2005).

Após essa primeira análise, observou-se a dificuldade de identificar o padrão de software mais adequado a ser utilizado durante a execução das etapas de um processo e na criação de artefatos. Essa dificuldade de adequação pode ser de diferentes tipos, mas a considerada neste projeto é referente ao problema encontrado quando um padrão pode ser aplicado em diversos artefatos e etapas, apesar de serem classificados para um contexto específico. Por exemplo, a GRN (Braga et al., 1999, 2000) contém padrões de análise, mas podem ser utilizados tanto durante a análise de domínio, quanto na construção de modelos do projeto, pois possuem estruturas da solução com classes que podem ser reusadas em diagramas de classes e modelos conceituais. Nem sempre as diferenças de contexto são óbvias, o que pode confundir os usuários de padrões de software.

Com o intuito de minimizar o problema na avaliação da adequação do emprego de padrões de software em artefatos e etapas de um processo, além de dividir para melhorar a visualização e minimizar erros no emprego, é proposto que padrões de software sejam associados às etapas de um processo de desenvolvimento e apresentados ao usuário quando

essas etapas são executadas. Assim, essas funcionalidades foram implementadas no ambiente Peônia, para fornecer um apoio maior e melhor ao desenvolvimento de aplicações apoiado por padrões de software.

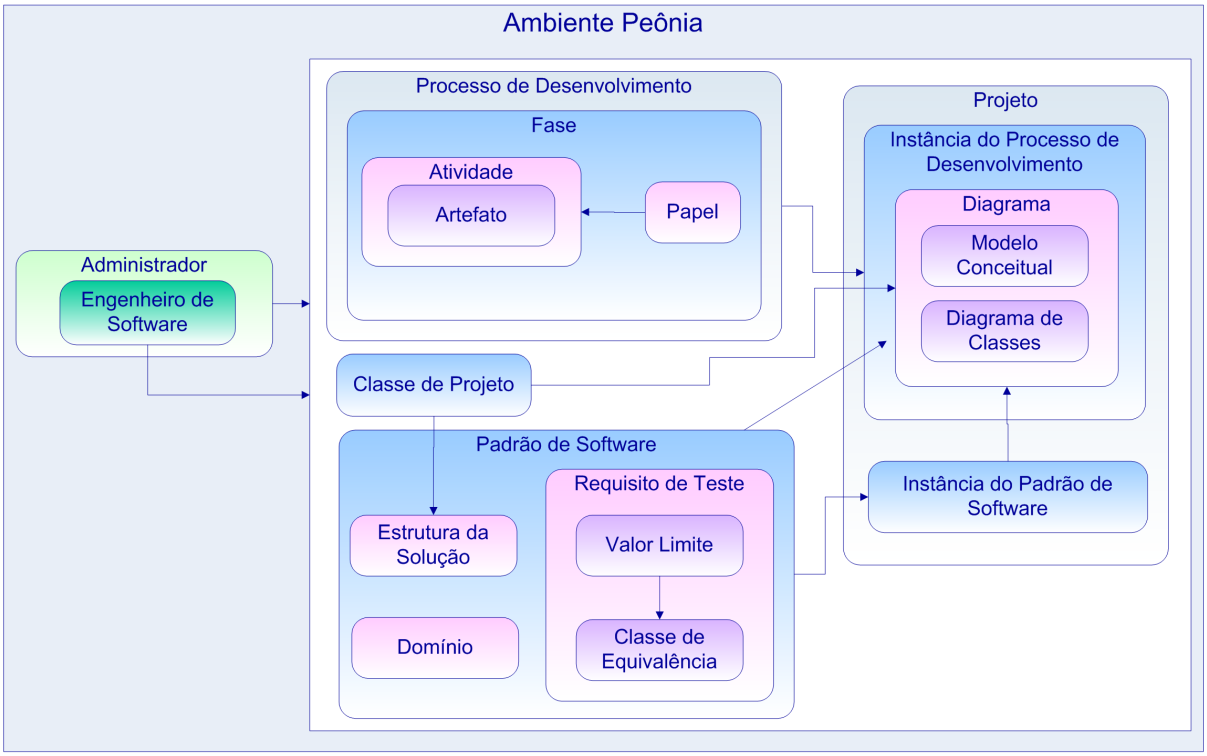

Figura 7.1: Relacionamento entre os conceitos do ambiente Peônia.

Durante a análise do problema, esboços dos relacionamentos entre os conceitos mencionados eram criados, melhorando a avaliação sobre a necessidade de adicionar outros conceitos. Ao fim da fase de análise do problema, o esboço dos conceitos do ambiente Peônia é apresentado na Figura 7.1.

Como aplicações *Web* possuem a vantagem de serem utilizadas por usuários em qualquer local e facilitarem a troca de informações, optou-se por desenvolver um ambiente *Web*. Para atender às necessidades específicas desse tipo de aplicação, o processo de desenvolvimento proposto por Conallen (2002), que recomenda o emprego do WAE como extensão para a modelagem de aplicações *Web*, foi escolhido para ser empregado nesse projeto.

A partir desse esboço foram executadas as fases *Desenvolver Modelo de Domínio* e *Análise do Problema Compreendido*, criando e adaptando o glossário e modelo conceitual. Ao final da execução, produziu-se o modelo conceitual ilustrado na Figura 7.2. Detalhes sobre a classificação empregada para diferenciar esses 22 (vinte e dois) conceitos do modelo são apresentados na Seção 7.9.

O próximo passo foi *Desenvolver o Documento de Visão* contendo o escopo e finalidades do ambiente Peônia. Neste projeto de mestrado, o Documento de Visão foi criado e incrementado com o glossário e o modelo de domínio e renomeado como Documento de Requisitos. Mais informações sobre o Documento de Requisitos são dadas na Seção 7.3.

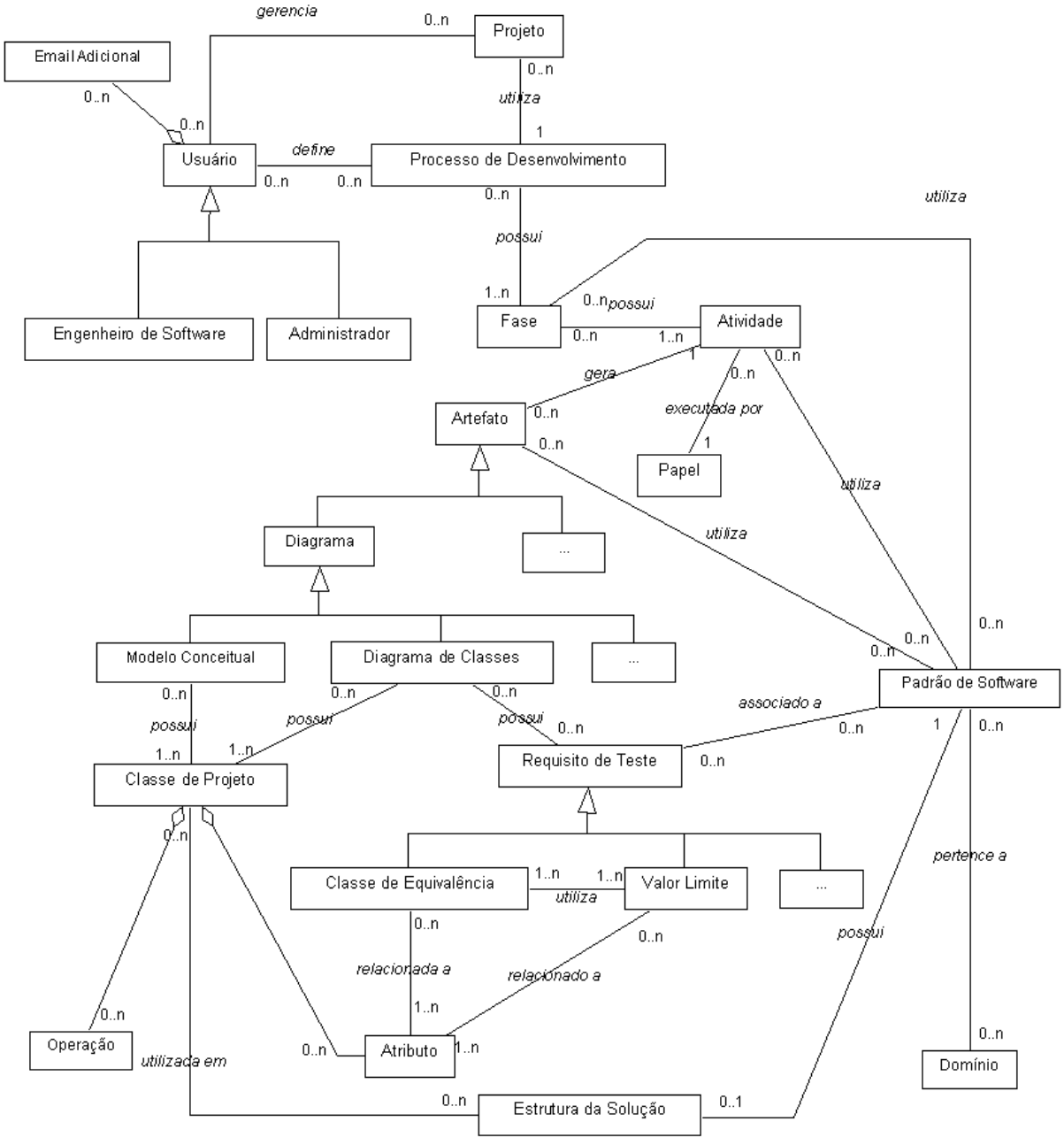

Figura 7.2: Modelo Conceitual do Ambiente.

7.3 Documentos de Requisitos

Como mencionado na Seção 7.2, para simplificar e reduzir a quantidade de documentos produzidos durante o desenvolvimento do ambiente Peônia, o glossário, com as definições, acrônimos e abreviaturas; o esboço do relacionamento entre os conceitos, ilustrado na Figura 7.1; e os requisitos levantados com a análise do problema foram adicionados ao Documento de Visão, formando um único documento denominado Documento de Requisitos. Uma amostra da estrutura dos requisitos desse documento é apresentada na Tabela 7.1.

Tabela 7.1: Requisitos da funcionalidade *Utilizar Padrão de Software no Diagrama*

- **R.8: Utilizar Padrão de Software no Diagrama**
  - **R.8.1:** O ambiente deve permitir aos usuários devidamente logados no ambiente utilizar um Padrão de Software em um Diagrama. O Padrão é utilizado em um Diagrama por meio da instanciação da sua Estrutura da Solução.
  - **R.8.2:** O ambiente deve permitir que a instância do Padrão seja salvo independentemente do Padrão de Software. Assim, um Padrão pode ser utilizado de maneiras diferentes e diversas vezes em um Diagrama.
  - **R.8.3:** O ambiente deve exibir para o usuário os itens que compõe a Estrutura da Solução do Padrão de Software e requerer o preenchimento dos campos necessário para a instanciação do Padrão. Os itens serão exibidos para que o usuário possa identificar melhor qual nome atribuir a cada um deles.
  - **R.8.4:** O ambiente deve solicitar os campos necessários para a instanciação de um Padrão de Software. Para cada Classe e Relacionamento da Estrutura da Solução do Padrão de Software, os campos oferecidos para preenchimento são: "Novo Nome da Classe" e "Novo Nome do Relacionamento". O preenchimento desses campos é opcional.
  - **R.8.5:** O ambiente deve fornecer, juntamente com os campos a serem preenchidos, uma opção para salvar as informações sobre a instanciação do Padrão de Software no Diagrama, sendo que os dados serão armazenados somente após escolher a opção para salvar.
  - **R.8.6:** O ambiente deve fornecer a possibilidade de cancelar a instanciação do Padrão de Software durante o preenchimento dos campos.
  - **R.8.7:** O ambiente deve permitir a remoção de uma instância do Padrão de Software. O usuário poderá selecioná-lo em uma lista contendo os Padrões instanciados.

Caso de Uso: Instanciar Padrão de Software	
Atores Principais: Administrador ou Engenheiro de Software	
Interessados e Interesses: - Engenheiro de Software: deseja criar uma instância de um padrão de software para ser utilizado em um diagrama. A instanciação é realizada para que os elementos de um padrão de software sejam renomeados adequadamente com o contexto do projeto.	
Pré-Condição: Administrador ou Engenheiro de Software autenticado no ambiente.	
Pós-Condição: Uma instância de um padrão de software é criada e associada a um diagrama.	
Fluxo Básico	
Ator	**Ambiente**
1. O usuário está preenchendo o formulário de inserção de itens em um diagrama, seleciona um padrão de software na lista.	2. O ambiente verifica se foi selecionado um dos padrões de software da lista.
	3. O ambiente exibe a página de instanciação de padrão de software com o formulário que deve ser preenchido. São apresentadas todas as classes e relacionamentos da estrutura da solução e o formulário para alterar o nome desses campos ("Novo Nome da Classe" e "Novo Nome do Relacionamento"). Também são exibidos três botões, sendo eles "Confirmar Alteração", "Concluir" e "Cancelar".
4. O usuário seleciona um dos relacionamentos entre classes de projeto da estrutura da solução para ter os nomes alterados.	5. O ambiente preenche o formulário de alteração de nome da estrutura da solução como o nome das classes e do relacionamento selecionado.
6. O usuário preenche os campos do formulário alterando o nome das classes e relacionamentos da estrutura da solução.	
7. O usuário clica no botão "Confirmar Alteração" para enviar as informações.	8. O ambiente cria a instância de cada classe e relacionamento da estrutura da solução com o novo nome. Se alguma classe ou relacionamento não for alterado, é criada a instância com o nome original cadastrada na estrutura da solução.
	9. O ambiente salva as informações da instancia do padrão de software.
	10. O ambiente exibe a página anterior, ou seja, o diagrama de classes que estava sendo alterado pelo usuário.
Fluxo Alternativo 1	
1a. O usuário está preenchendo o formulário de inserção de itens em um diagrama, seleciona um padrão de software e clica na opção de "Associar Padrão".	
	1. O ambiente executa os passos 2 e 3.
2. O usuário executa os passos 4, 6 e 7.	3. O ambiente executa os passos 5, 8, 9 e 10.
Fluxo Alternativo 2	
1b. O usuário não seleciona um dos padrões de software e clica no botão "Associar Padrão".	
	1. O ambiente verifica se foi selecionado um dos padrões de software da lista.
	2. O ambiente exibe uma mensagem informando que é necessário selecionar um dos padrões de software apresentados.
Fluxo Alternativo 3	
4a, 6a. O usuário desiste de realizar a operação. Ele não seleciona um dos padrões de software; seleciona e não confirma; não preenche o formulário; ou começa a preencher e não confirma.	
1. O usuário clica no botão "Cancelar" para interromper a instanciação.	2. O ambiente não realiza a instanciação do padrão de software.
	3. O ambiente exibe a página anterior, ou seja, o diagrama de classes que estava sendo alterado pelo usuário.
Fluxo Alternativo 4	
6b. O usuário não preenche os campos opcionais para alterar o nome dos relacionamentos.	
1. O usuário executa o passo 7.	2. O ambiente executa os passos 5, 8, 9 e 10.

Figura 7.3: Caso de uso detalhado da atividade *Instanciar Padrão de Software*.

Por ser uma aplicação *Web*, permitindo muita interação com o usuário, e por ter muitos conceitos envolvidos optou-se por descrever com mais detalhes os casos de uso, para que a interação entre usuário e ambiente fossem melhor avaliados. Assim, foi criado também o Documento de Casos de Uso Detalhados, com explicações, passos e fluxos alternativos de execução de todas as funcionalidades do ambiente Peônia.

Além disso, esse documento foi criado para facilitar a criação do diagrama de atividades e seqüência e as atividades de VV&T. Um exemplo de caso de uso detalhado é ilustrado na Figura 7.3. Destaca-se que foi utilizado o padrão "Requisitos Encapsulados" (Ramos et al., 2007) para remover requisitos duplicados do documento.

Ao final da análise do problema e domínio, foram levantadas 168 funcionalidades essenciais, mencionadas no Capítulo 6, para que o ambiente Peônia forneça um apoio básico ao desenvolvimento de aplicações.

Outras funcionalidades, encontradas na documentação do ambiente Peônia, foram levantadas, mas não foram implementadas por restrições de cronograma, ou por serem de baixa prioridade. Algumas delas serão implementadas num futuro próximo por serem consideradas de maior prioridade, como é o caso das atividades *Exportar Diagrama* e *Importar Diagrama*.

7.4 Plano de Projeto

Como todas as atividades do desenvolvimento do ambiente Peônia foram realizadas por uma única analista de sistemas, foi criado um documento com o *Plano de Projeto* resumido. As seguinte regras foram estabelecidas para o projeto:

- O desenvolvimento deve ser realizado de maneira incremental;
- Cada versão produzida em uma iteração deve ser um protótipo operacional do ambiente Peônia;
- Para cada versão produzida com as iterações, um arquivo compactado deve ser criado com o nome do ambiente e a data da versão.
- Deve ser seguido o processo de desenvolvimento proposto por Conallen (2002), produzindo os artefatos recomendados.
- Como a produção dos artefatos são apenas recomendações dadas por Conallen (2002), eventualmente, por restrições de cronograma ou por repetição de informação, alguns artefatos podem deixar de ser produzidos ou adaptados. Caso haja necessidade, artefatos adicionais também podem ser incluídos.

Além disso, também foram definidos os artefatos que deveriam ser produzidos durante o projeto do ambiente Peônia sendo eles: documento de requisitos, modelo de casos de uso, modelo conceitual, modelo UX, diagramas de seqüência, diagramas de atividades, diagrama de classes, plano de teste e manual *online* de auxílio ao usuário. A aprovação dos artefatos estava sujeita apenas à orientadora deste projeto de mestrado, não tendo sido definidos critérios formais sobre o formato e regras de validação dos artefatos produzidos.

Após as análises feitas e os documentos produzidos, iniciou-se o projeto da aplicação. Esse projeto foi realizado com a definição do modelo arquitetural, apresentado na Seção 7.5, com as divisões das responsabilidades de cada uma das camadas do ambiente Peônia, foi desenvolvido o MER (Modelo Entidade Relacionamento) e realizado o mapeamento desse MER para o MRel (Modelo Relacional) para gerar corretamente as tabelas do banco de dados, ambos descritos na Seção 7.6. Também foram construindos os diagramas de casos de uso, classes, atividades e seqüência, explicados na Seção 7.7 e criados os modelos UX, detalhados na Seção 7.8, utilizados para definir a estrutura da interface visualizada pelo usuário. Por fim, foi realizada a implementação e teste da aplicação, como descrito na Seção 7.10.

7.5 Modelo Arquitetural

Na Figura 7.4 é apresentada o modelo arquitetural do ambiente Peônia. É utilizada a arquitetura de três camadas e o padrão arquitetural MVC (Krasner e Pope, 1988). Na **Camada de Apresentação**, a **Visão** é responsável por armazenar as informações enviadas pelo **Navegador do Cliente** e enviá-las para o **Controlador**, que por sua vez, é responsável por processar as informações recebidas, transformando-as em solicitações para a **Camada de Aplicação**. Na **Camada de Aplicação**, o **Modelo** realiza o processamento da lógica do negócio, enviando os dados para a **Camada de Persistência**, ou requisitando informações dela. A **Camada de Persistência** realiza a busca por dados e o armazenamento de informações no **Banco de Dados** do ambiente Peônia.

Como mencionado na Seção 4.4, o *Framework* ZK foi utilizado para o desenvolvimento da Camada de Apresentação do ambiente Peônia. A Visão e o Controlador foram implementados separadamente. A Visão foi implementada utilizando os componentes fornecidos pelas linguagens XUL, XHTML e ZUML para o desenvolvimento de cada página apresentada para o usuário, sendo que essas páginas estão em arquivos no formato "ZUL". Por sua vez, o Controlador utiliza o componente (*tag*) "zscript", que permite a renderização de código na linguagem Java nas páginas ZUML. O Controlador contém métodos de manipulação de dados, que estão separados em arquivos do tipo "ZS". Esses métodos de manipulação são responsáveis pelo tratamento inicial das informações enviadas

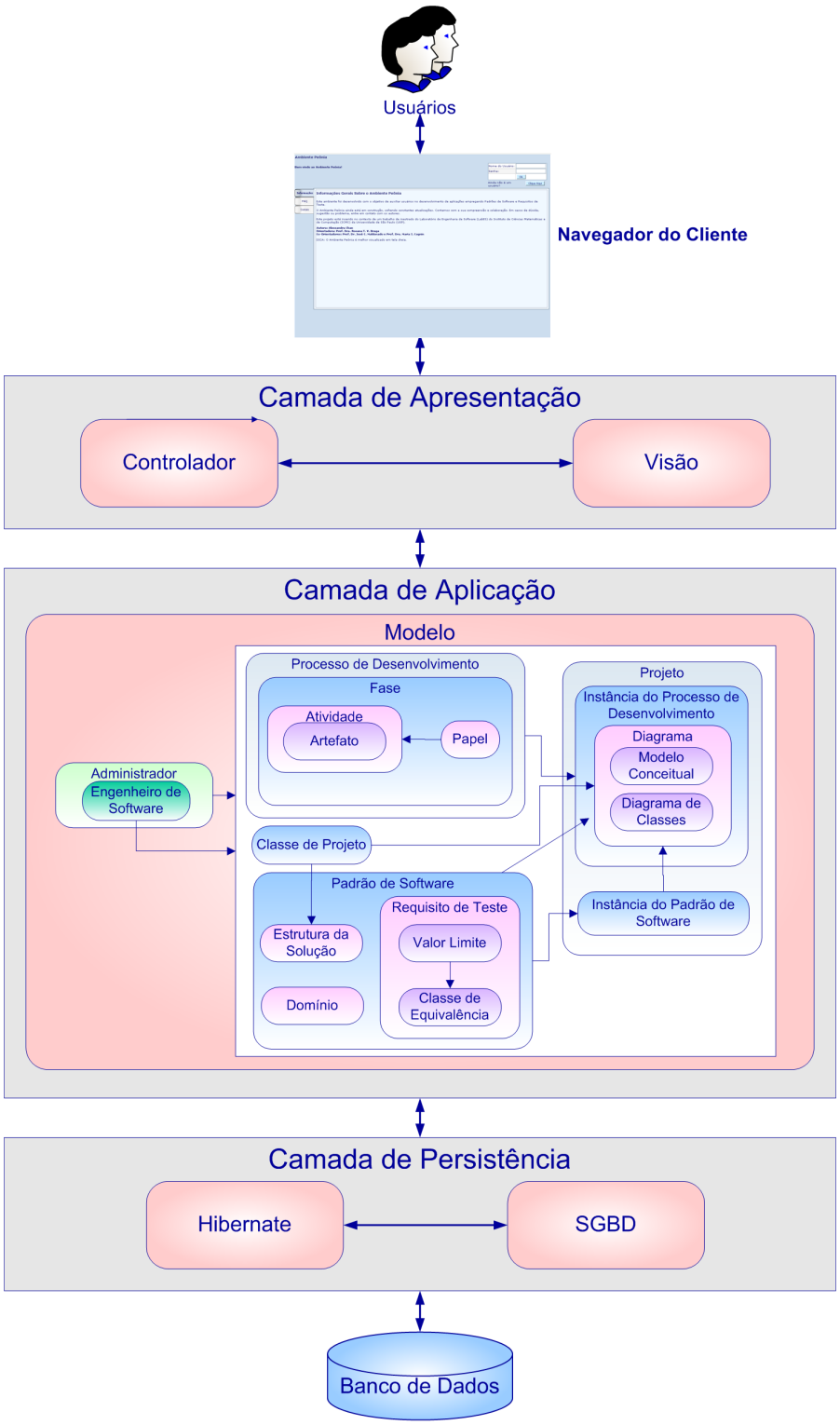

Figura 7.4: Modelo arquitetural do ambiente.

pelo usuário, como reconhecimento de erros e conversão para classes reconhecidas pela Camada de Aplicação. Também é por meio dos métodos do Controlador que os dados

enviados pela Camada de Aplicação são convertidos e posicionados para que os usuários os compreendam.

```
public class SoftwarePattern {

    private Long code;
    private Long userCode;

    private String name;
    private String problem;
    private String solution;
    private String context;
    private String forces;
    private String example;
    private String resultingContext;
    private String rationale;
    private String knownUse;
    private String comments;
    private boolean solutionStructure;

    public SoftwarePattern(){
    }

    public Long getCode(){
        return code;
    }

    private void setCode(Long code){
        this.code = code;
    }

    public Long getUserCode(){
        return userCode;
    }

    public void setUserCode(Long userCode){
        this.userCode = userCode;
    }

    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name = name;
    }

    public String getProblem(){
        return problem;
    }

    public void setProblem(String problem){
        this.problem = problem;
    }

    public String getSolution(){
        return solution;
    }

    public void setSolution(String solution){
        this.solution = solution;
    }

    public String getContext(){
        return context;
    }

    public void setContext(String context){
        this.context = context;
    }

    public String getForces(){
        return forces;
    }

    public void setForces(String forces){
        this.forces = forces;
    }

    public String getExample(){
        return example;
    }

    public void setExample(String example){
        this.example = example;
    }

    public String getResultingContext(){
        return resultingContext;
    }

    public void setResultingContext(String resultingContext){
        this.resultingContext = resultingContext;
    }

    public String getRationale(){
        return rationale;
    }

    public void setRationale(String rationale){
        this.rationale = rationale;
    }

    public String getKnownUse(){
        return knownUse;
    }

    public void setKnownUse(String knownUse){
        this.knownUse = knownUse;
    }

    public String getComments(){
        return comments;
    }

    public void setComments(String comments){
        this.comments = comments;
    }

    public boolean isSolutionStructure() {
        return solutionStructure;
    }

    public void setSolutionStructure(boolean solutionStructure) {
        this.solutionStructure = solutionStructure;
    }
}
```

Figura 7.5: Classe de Persistência do Padrão de Software.

Para a Camada de Persistência foi utilizado o *framework* Hibernate, pois oferece mecanismos para realizar a comunicação transparente entre o Banco de Dados e a Camada de Aplicação, além de outras facilidades, como geração automática de tabelas, controle

de concorrência, a linguagem HQL e a rica documentação. O mapeamento das tabelas e de configurações relacionadas à comunicação com o SGBD foram realizadas em XML e a implementação das classes de persistência realizadas em Java. O *framework* Hibernate denomina classe de persistência o JavaBean contendo os atributos que representam as colunas das tabelas do SGBD e os métodos de acesso a esses atributos. A classe de persistência do conceito *Padrão de Software* do ambiente Peônia é apresentada na Figura 7.5.

Utilizando o padrão DAO (*Data Access Object*) (Sun Microsystems, Inc., 2003a), que propõe a criação de um objeto para encapsular o acesso e comunicação à base de dados, os métodos de comunicação, que utilizam o *framework* Hibernate para acessar o SGBD, foram implementados em classes Java com o mesmo nome da classe de persistência acrescidas do sufixo "DAO". Nesses métodos de comunicação as informações enviadas pelo usuário são tratadas para que sejam armazenadas no SGBD de maneira consistente, seguindo as regras de negócio do ambiente Peônia. As classes de persistência, com os atributos, e as classes DAO, com os métodos de manipulação de informações e comunicação com o SGBD, formam a Camada de Aplicação do ambiente Peônia.

7.6 Estrutura do Banco de Dados

Apesar do processo proposto por Conallen (2002) não mencionar a necessidade de desenvolver o modelo da base de dados, em razão da complexidade dos relacionamento entre os conceitos do ambiente Peônia, não foi possível criar as tabelas do banco de dados sem o auxílio de um MER. Assim, analisando os requisitos documentados e o modelo conceitual, ilustrado na Figura 7.2 da Seção 7.2, foi desenvolvido o MER apresentado na Figura 7.6.

A partir do MER, foi realizado o mapeamento para um MRel contendo todos os campos, tabelas e chaves para que as informações produzidas pela execução das funcionalidades possam ser corretamente armazenadas no banco de dados do ambiente Peônia. Por fim, utilizando os dados do MRel foram criados os arquivos XML contendo os mapeamentos exigidos pelo *framework* Hibernate para que possa criar e gerenciar as tabelas da aplicação.

Com relação às tabelas criadas, algumas decisões de projeto podem ser destacadas. Optou-se por não criar uma tabela contendo *Estrutura da Solução* por ser uma entidade fraca com uma relação de cardinalidade 1:1 (um para um) com *Padrão de Software*. Criou-se apenas um campo indicando a existência da *Estrutura da Solução* na tabela que representa um *Padrão de Software*.

Também foi decidido que não havia necessidade de criar uma tabela com a *Instância da Estrutura da Solução*, pois é suficiente que os relacionamentos instanciados da *Estrutura*

da Solução contenham a identificação sobre qual *Padrão de Software* e *Diagrama* pertencem. Como é permitido que um *Padrão de Software* seja aplicado mais que uma vez em um *Diagrama*, foi necessário criar nos relacionamentos instanciados um campo adicional identificando a ordem da instância empregada, ou seja, se o relacionamento instanciado pertence a primeira utilização, ou segunda, e assim por diante.

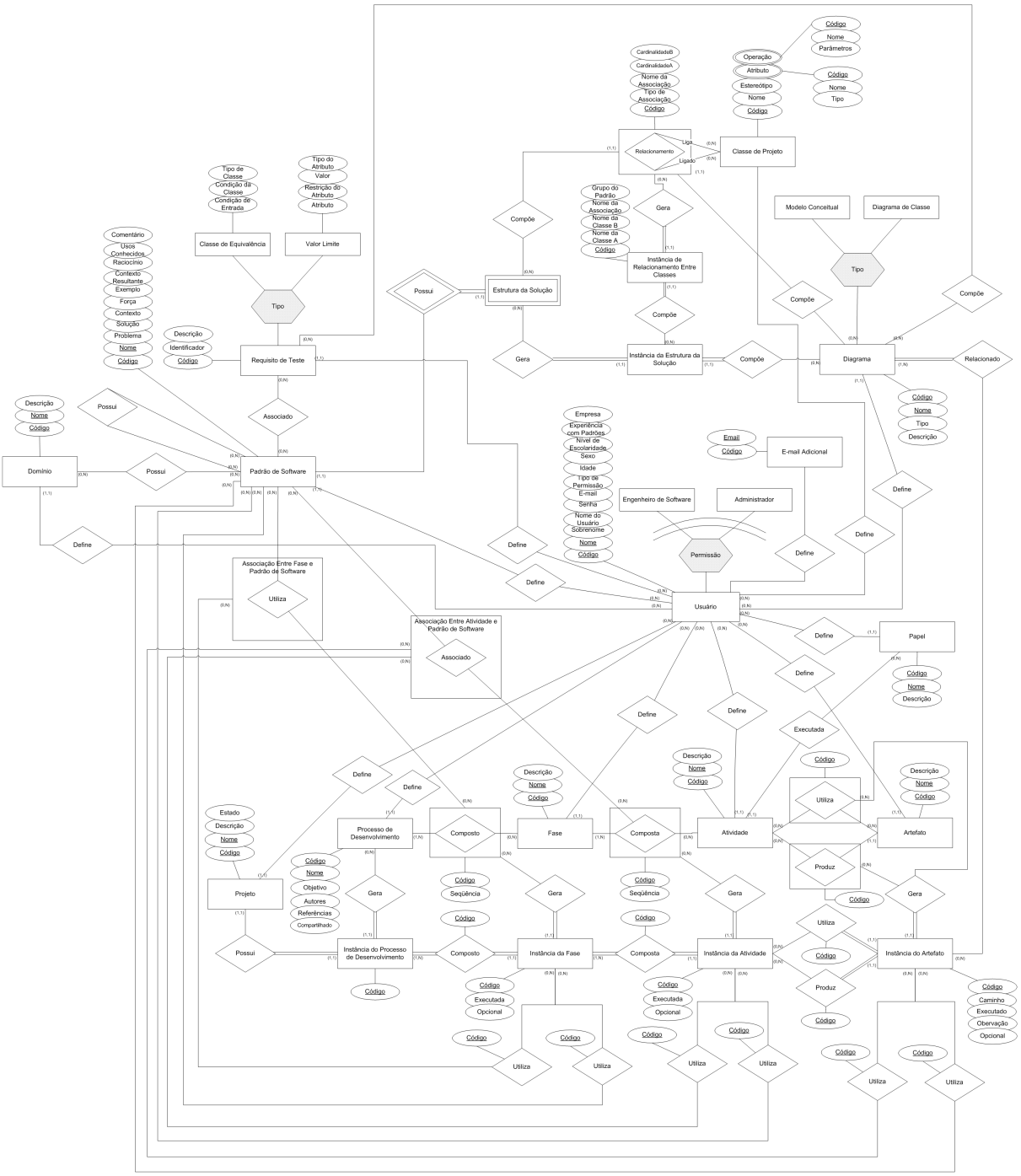

Figura 7.6: Modelo Entidade Relacionamento do Ambiente Peônia.

Para a generalização que envolve o elemento *Requisito de Teste* optou-se por criar uma tabela contendo as informações comuns dos sub-conceitos *Classe de Equivalência* e *Valor Limite*. É na tabela de *Requisito de Teste* que fica o controle dos códigos que identificam os registros no banco de dados, repassando para os sub-conceitos. Assim, é permitido que requisitos de testes gerados automaticamente pelo ambiente Peônia contenham os mesmos dados comuns, armazenados na tabela *Requisito de Teste*, e o mesmo código identificador, para que seja possível identificar a partir de qual *Valor Limite* foi gerada a *Classe de Equivalência*. Outras informações a respeito da atividade *Gerar Requisito de Teste* são descrita na Seção 6.2.2.3.

Como os sub-conceitos *Modelo Conceitual* e *Diagrama de Classes* não possuem atributos especializados, apenas o elemento *Diagrama* foi transformado em uma tabela com um campo que identifique o seu tipo. A opção por criar uma tabela genérica de *Diagrama* foi realizada para permitir que outros diagramas sejam facilmente adicionados ao ambiente Peônia. Por meio do campo de identificação do tipo, é possível localizar facilmente onde devem ser procuradas informações específicas, caso seja necessário criar tabelas para armazená-las.

Assim como *Diagrama*, para tratar a generalização que envolve *Usuário*, optou-se por criar um campo identificador que armazene o tipo de usuário, ou seja, se é *Administrador* ou *Engenheiro de Software*. Além disso, nesse campo é possível inserir dois outros tipos, criados para facilitar o controle do ambiente Peônia: *Pendente* e *Removido*. Detalhes sobre como é realizada a gerência de usuários são encontrados na Seção 6.2.1.1.

7.7 Modelos de Análise e Projeto

A partir dos requisitos levantados e do modelo conceitual, ilustrado anteriormente na Figura 7.2, foi desenvolvido o diagrama com os casos de uso do ambiente Peônia. Partes desse diagrama são apresentadas nas Figuras 7.7, 7.8, 7.10, 7.11 e 7.9.

Por opção de projeto, a lista de elementos sempre é exibida ao usuário para que possa visualizar o que já existe cadastrado, como uma tentativa de minimizar repetições. Assim, as funcionalidades de cadastro, alteração e remoção dependem do caso de uso que lista os elementos desejados. Na Figura 7.7, o cadastro, alteração e remoção de padrão de software incluem o caso de uso *Listar Padrões de Software*, sendo que para a alteração e remoção, o padrão deve ser selecionado na lista e assim as informações são exibidas para que o usuário, como explicado na Seção 6.2.2.5 possa ter certeza ao requerer a operação.

Os casos de uso relacionados à *Estrutura da Solução*, ilustrados na Figura 7.7, e à associação de domínios, requisitos de teste e outros padrões de software ao padrão de software, exibidos Figura 7.8, só podem ser realizados durante o cadastro ou alteração de

um padrão de software. A remoção do padrão de software inclui também a remoção da estrutura da solução e das associações com domínios, requisitos de teste e outros padrões de software. O gerenciamento de requisitos de teste, inclusive das classes de equivalência e valores limites, é ilustrado na Figura 7.9.

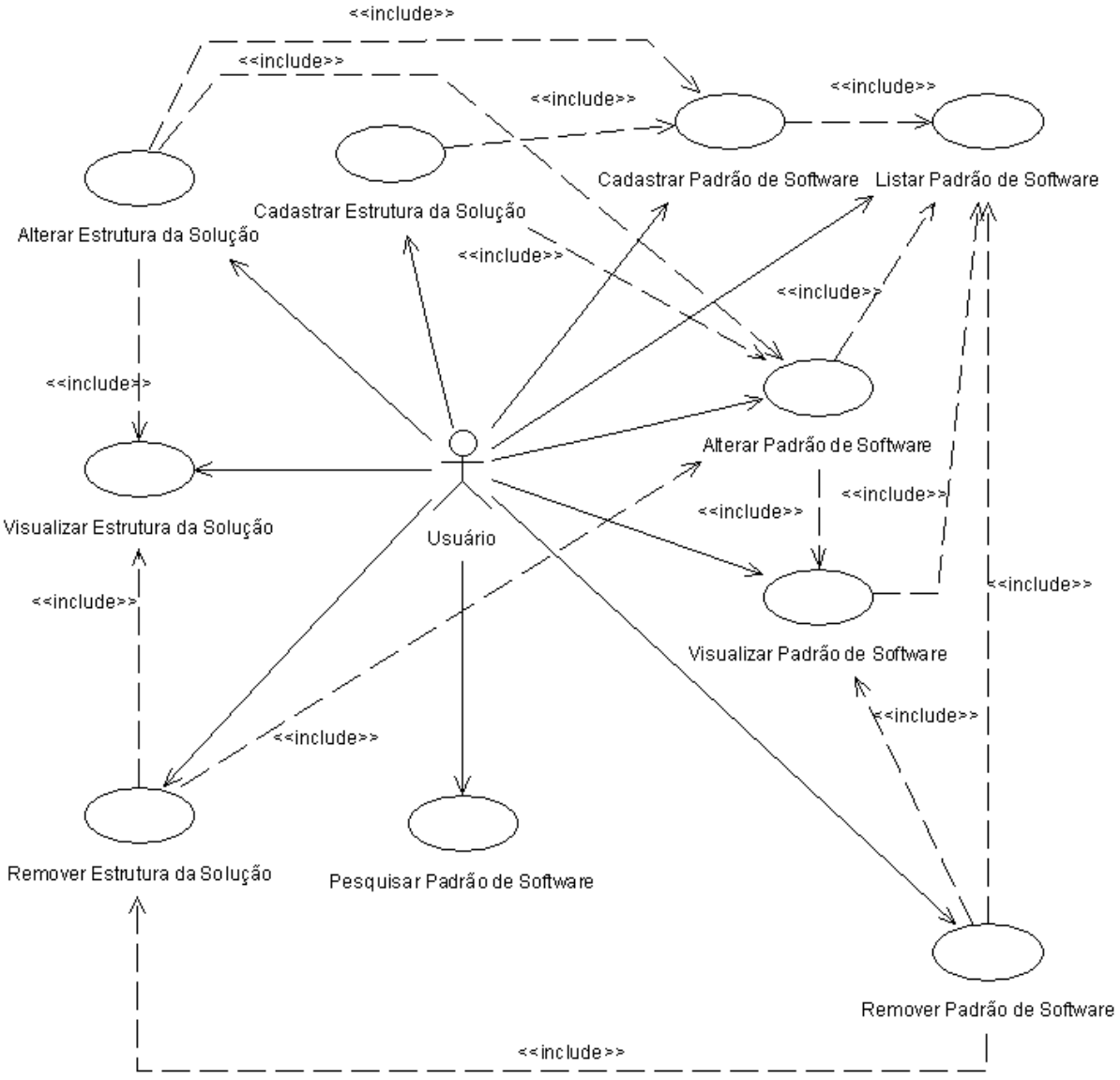

Figura 7.7: Diagrama de Casos de Uso Para Gerenciar Padrão de Software e Estrutura da Solução.

Na Figura 7.10, são apresentados os casos de uso relacionados ao emprego de um padrão de software em diagramas. Se o padrão possuir uma estrutura da solução, uma instância do padrão de software é criada para que o usuário possa alterar o nome das classes de projeto e relacionamentos que a compõem, como explicado na Seção 6.2.2.5.

Também são exibidos na Figura 7.10 os casos de uso para a associação de padrões de software às fases e atividades que é realizada durante a gerência de um processo de desenvolvimento, cujos casos de uso são ilustrados na Figura 7.11.

Em seguida, os casos de uso foram analisados para que os diagramas de atividades e seqüência fossem criados, detalhando fluxo de execução das atividades e a interação entre os elementos do ambiente Peônia. É possível observar na Figura 7.12 a ordem das ações executadas pelo usuário e como o ambiente Peônia responde a esses estímulos durante a atividade de cadastro de padrão de software.

Por sua vez, a Figura 7.13 ilustra a seqüência de métodos que são executados pelo ambiente Peônia para efetivamente realizar o cadastro de padrão de software no banco de dados.

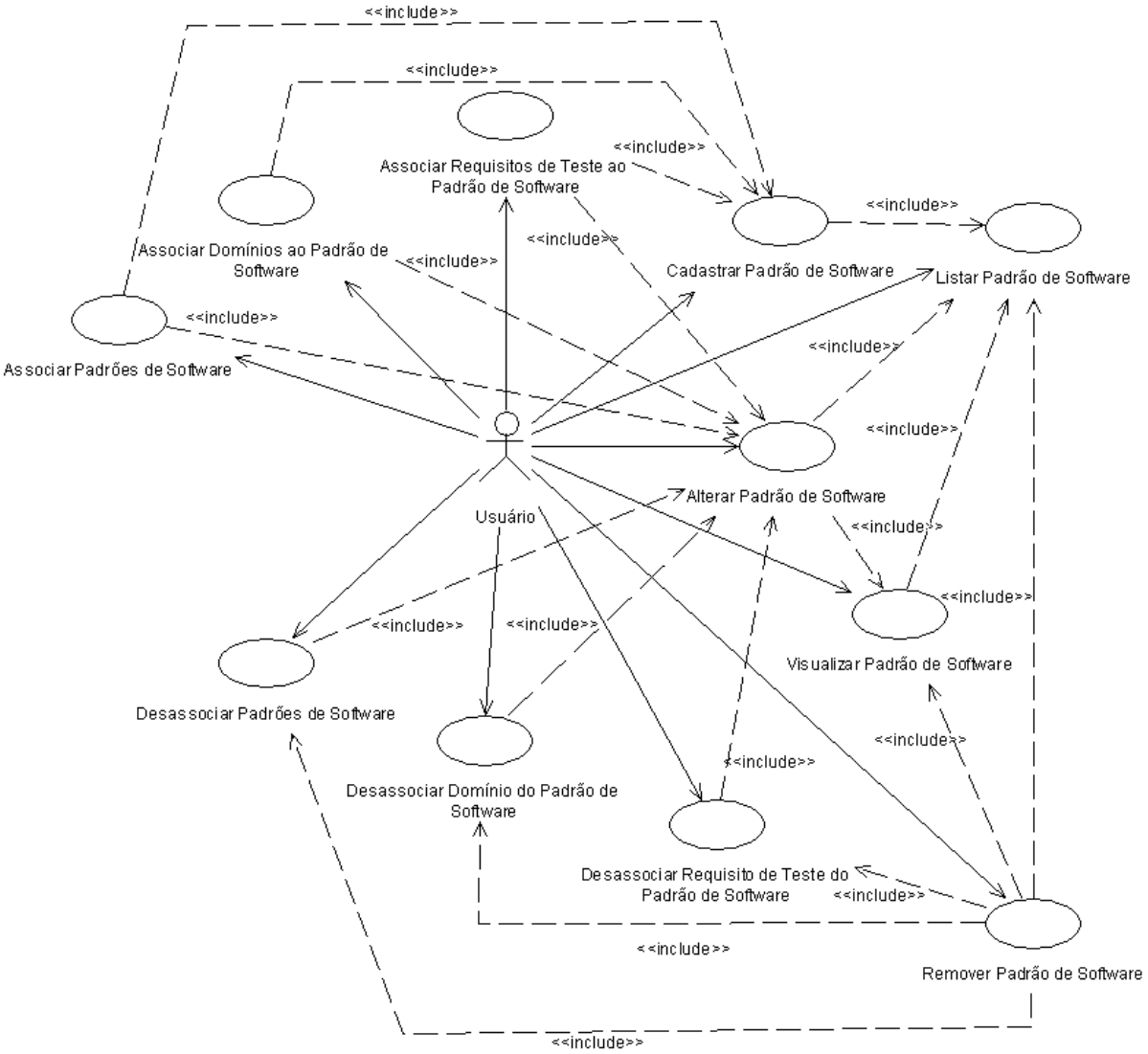

Figura 7.8: Diagrama de Casos de Uso Para Associar Domínios, Requisitos de Teste e Outros Padrões a um Padrão de Software.

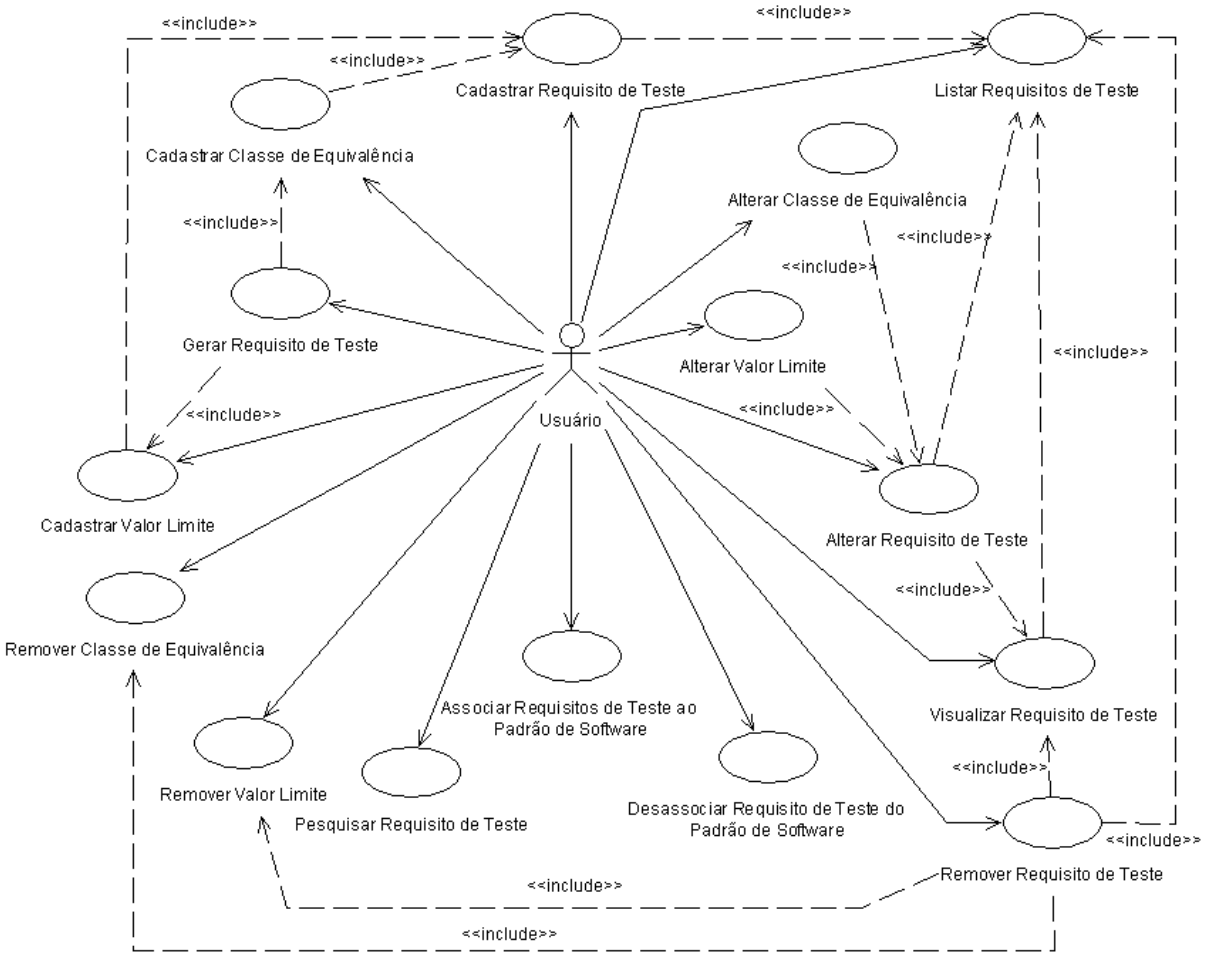

Figura 7.9: Diagrama de Casos de Uso Para Gerenciar Requisito de Teste.

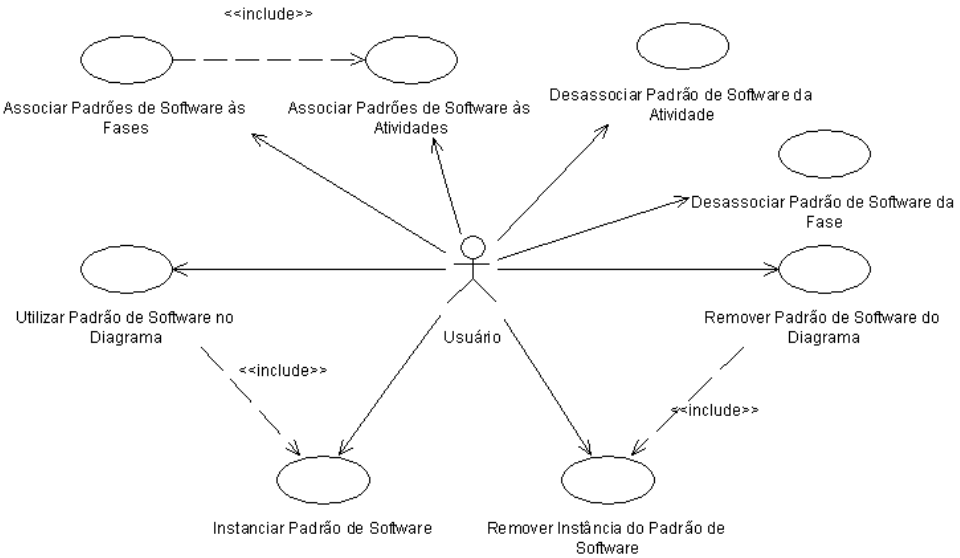

Figura 7.10: Diagrama de Casos de Uso Para Associar Padrão de Software às Etapas de um Processo de Desenvolvimento e Utilizar em Diagramas.

Outro exemplo de diagrama de seqüência é apresentado na Figura 7.14, que lista a ordem que os métodos devem ser executados para realizar a inserção de um processo de desenvolvimento, inclusive para a construção da sua estrutura e a associação de padrões de software a suas etapas.

Após a criação dos diagramas que permitem ter uma visão da interação entre os elementos do ambiente Peônia, foi desenvolvido o diagrama de classes com informações sobre o projeto, métodos e comunicação entre classes, que em razão do seu tamanho e complexidade será disponibilizado futuramente junto com a documentação do ambiente Peônia.

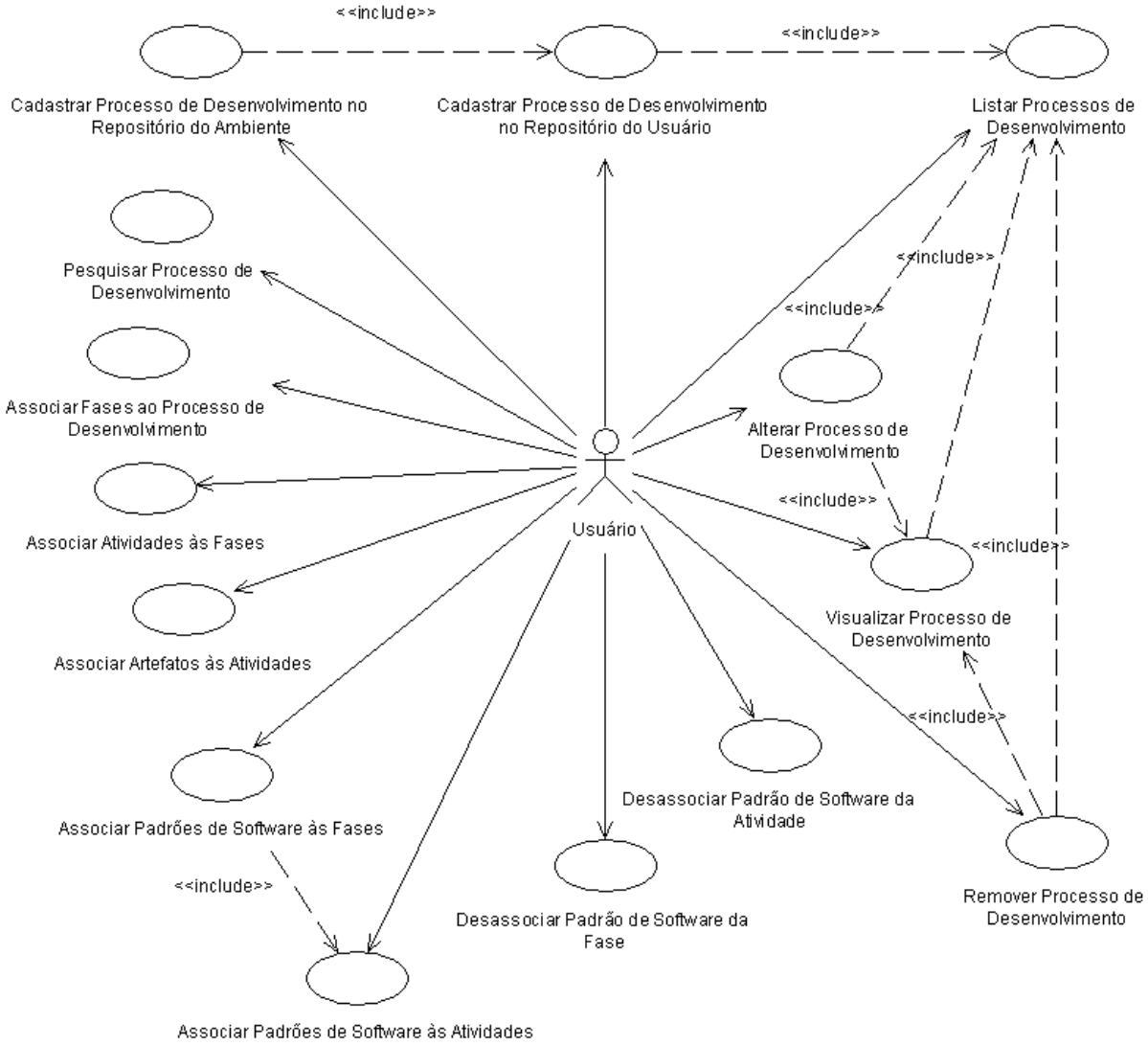

Figura 7.11: Diagrama de Casos de Uso Para Gerenciar Processo de Desenvolvimento.

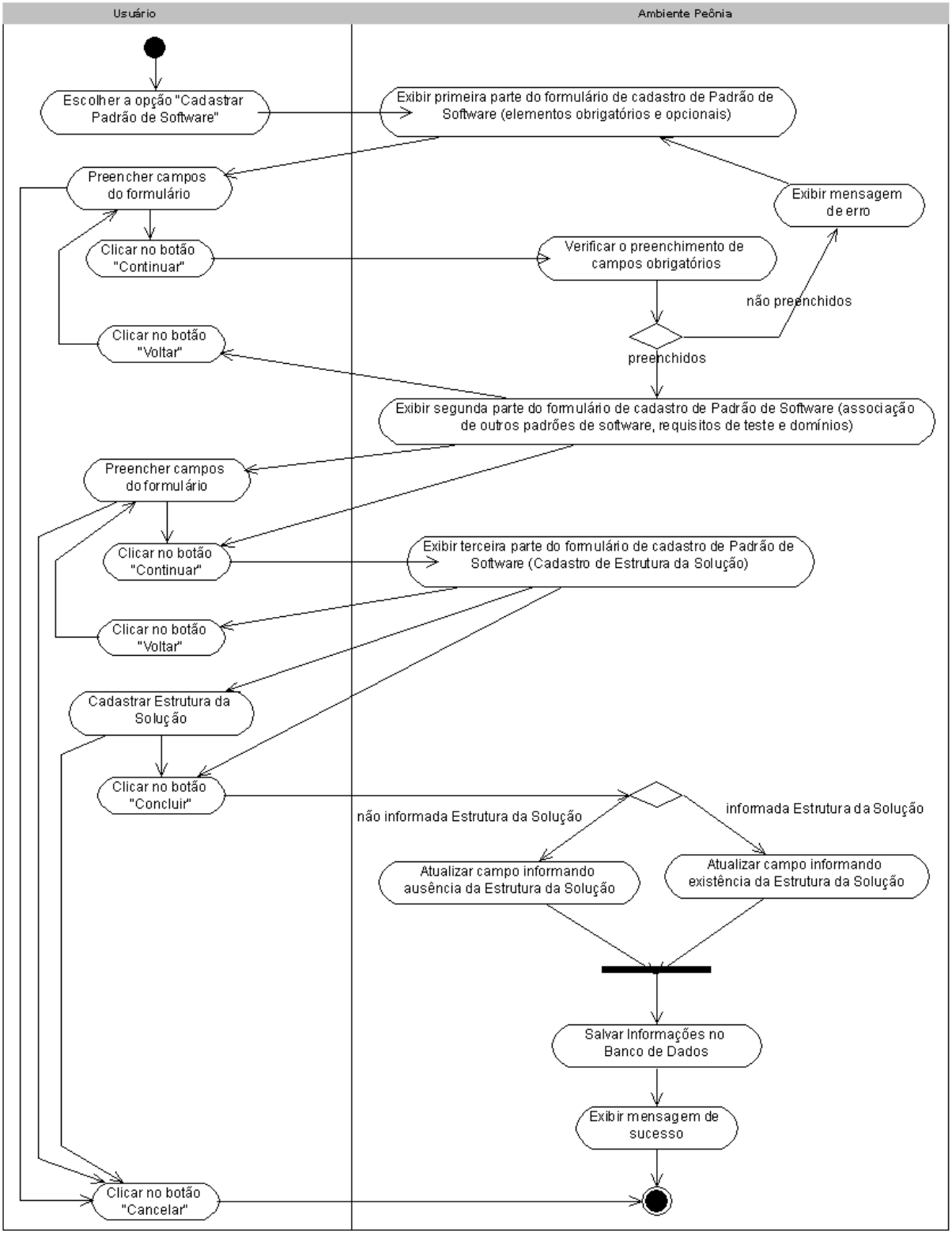

Figura 7.12: Diagrama de Atividades de *Cadastrar Padrão de Software*.

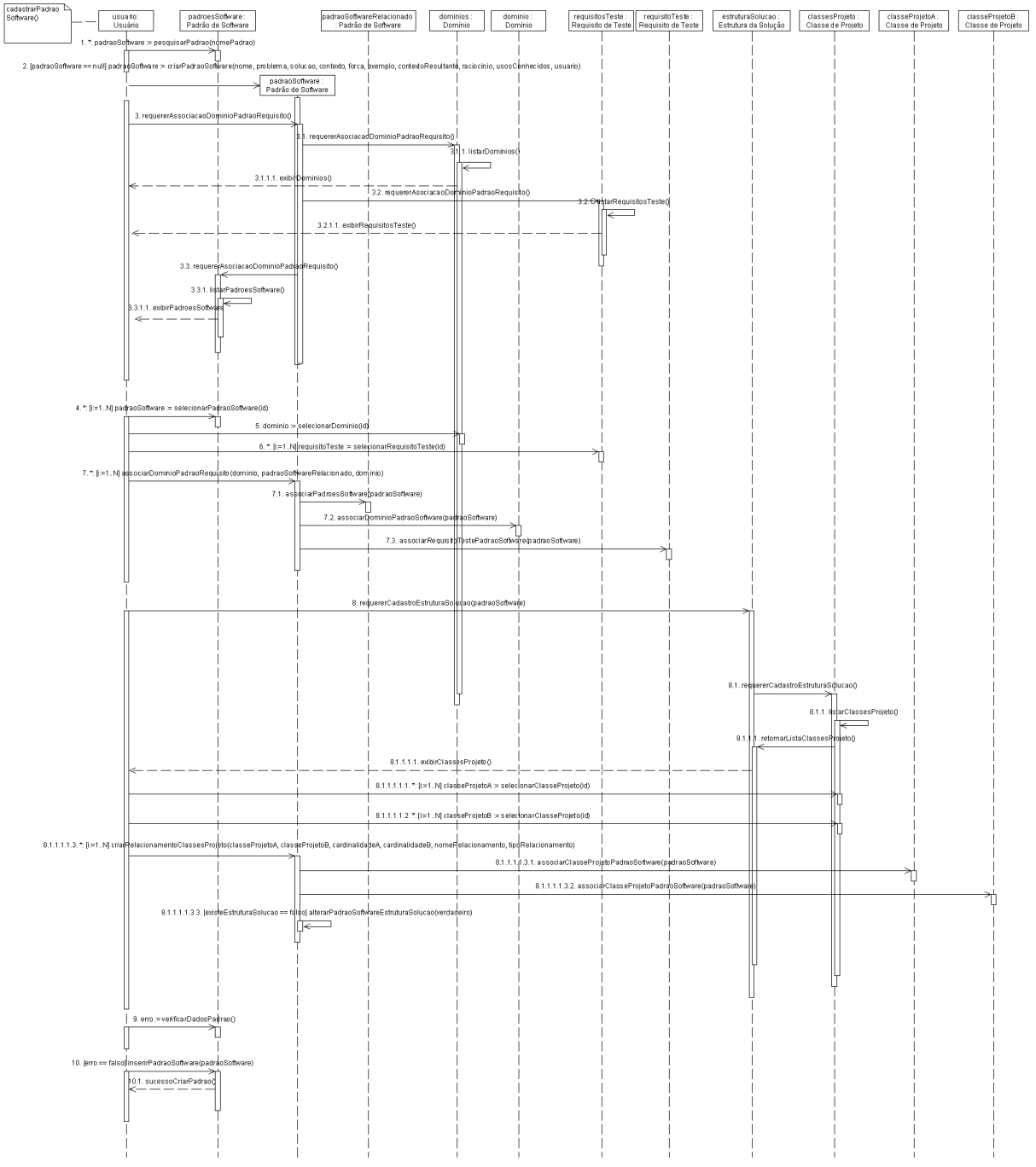

Figura 7.13: Diagrama de Seqüência de *Cadastrar Padrão de Software*.

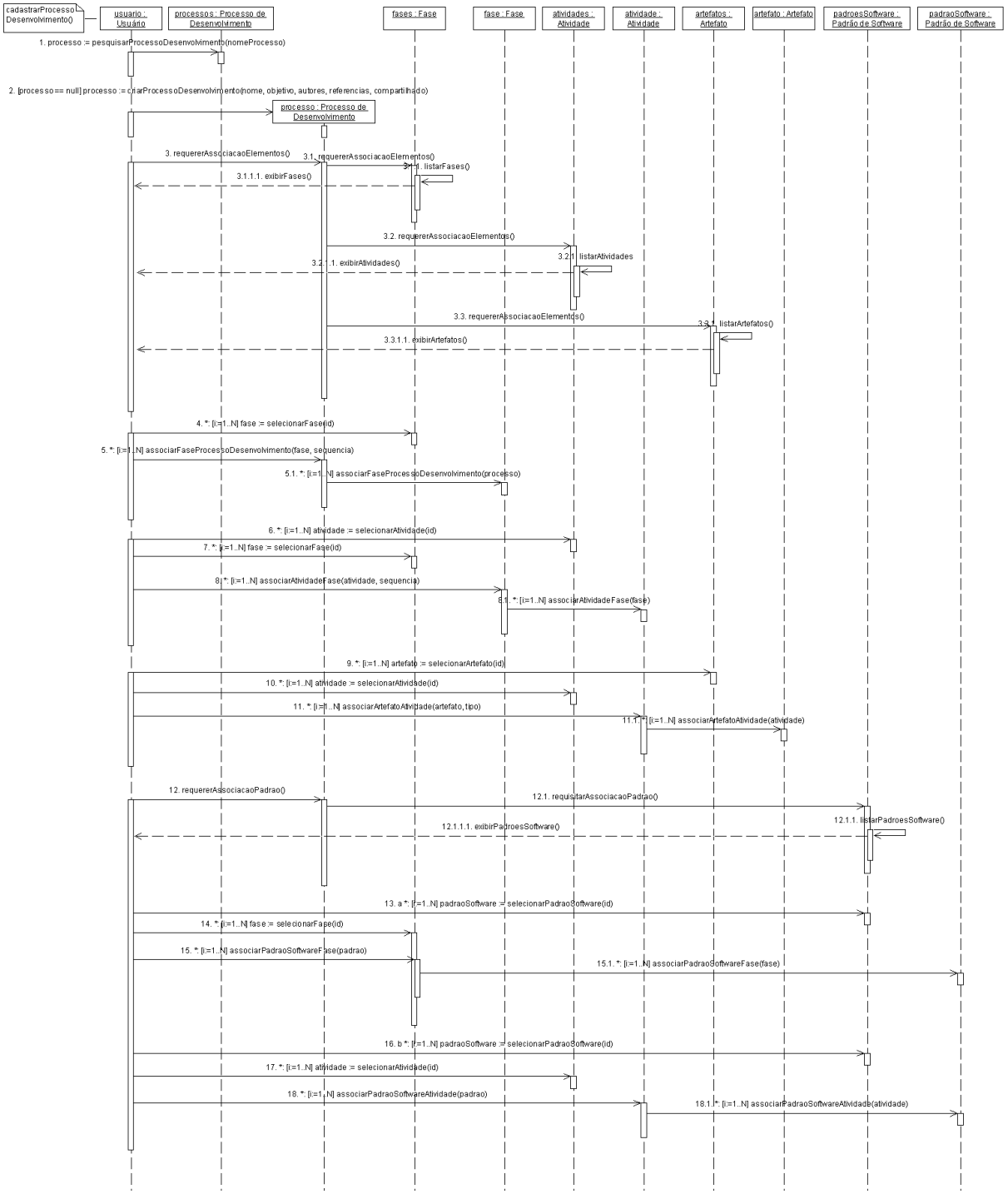

Figura 7.14: Diagrama de Seqüência de *Cadastrar Processo de Desenvolvimento*.

7.8 Modelos UX

Como mencionado na Seção 2.3.1, Conallen (2002) propõe o desenvolvimento de um documento de diretrizes de UX descrevendo a aparência global da aplicação e regras para definição de novas telas.

Assim, foi criado um documento contendo o Modelo UX, permitindo visualizar o ambiente Peônia por meio de suas telas. Por causa das restrições do cronograma e por terem sido criados com detalhes os casos de uso, não foi necessário criar os roteiros propostos por Conallen (2002).

As telas das funcionalidades foram criadas, em seguida, foram definidos os mapas de navegação a partir das interações descritas nos casos de uso detalhados. À medida que os diagramas de atividade e de seqüência eram criados, os modelos UX eram atualizados para adequar-se ao projeto. Exemplos dos modelos UX do ambiente Peônia são ilustrados nas Figuras 7.15 e 7.16.

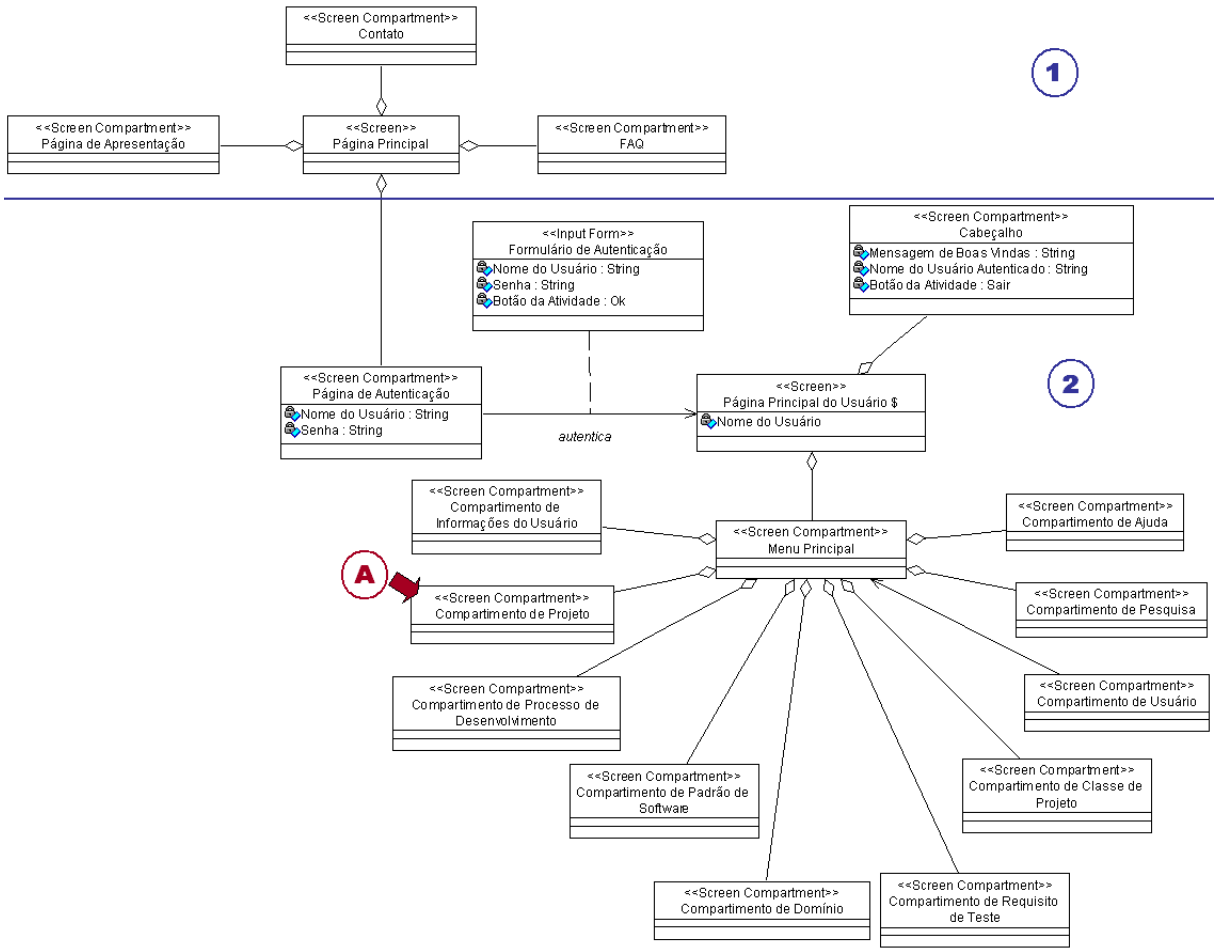

Figura 7.15: Modelo UX das telas envolvidas na atividade *Acessar Sessão do Usuário*.

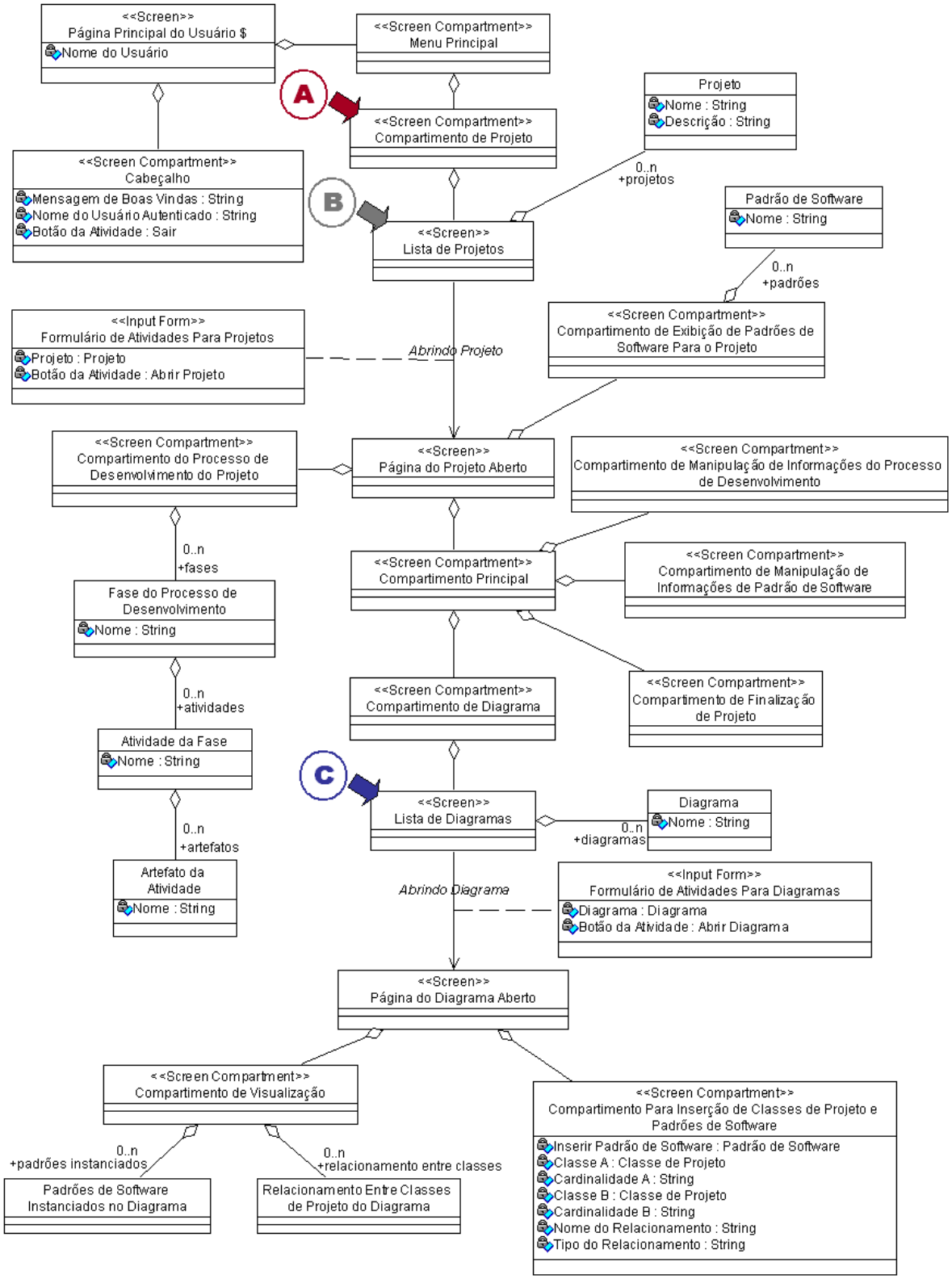

Figura 7.16: Modelo UX das telas envolvidas nas atividades de *Abrir Projeto* e *Abrir Diagrama*.

O modelo UX apresentado na Figura 7.15 representa a estrutura das telas e o fluxo de navegação de um visitante do ambiente Peônia que possui um cadastro e decide acessar

a sua sessão. Ele está dividido em duas partes. Na parte 1, são mostrados os compartimentos que formam a página inicial de visitantes do ambiente Peônia, cujo resultado da implementação é ilustrado no começo deste capítulo na Figura 6.1. Após inserir as informações necessárias para acessar o ambiente Peônia, é exibida a tela da página inicial da sessão do usuário autenticado, apresentado na parte 2 da Figura 7.15 e cujo resultado da implementação é exibido na Figura 6.7, contida na Seção 6.2.1.2. Observa-se pelo modelo que, para que o ambiente pudesse ter diversos compartimentos sem comprometer a visualização dos elementos, foi necessário dividir as informações em abas (*tabs*). Explicações mais detalhadas a respeito de como foi realizada a divisão dos elementos em abas são apresentadas na Seção 7.9.

Como a quantidade de telas e mapas de navegação é muito grande, os modelos UX foram divididos em partes. Por exemplo, um dos compartimentos do menu principal contém as informações dos projetos do usuário, destacado pela letra "A" vinho na Figura 7.15. A mesma letra "A" vinho na Figura 7.16 indica a continuação do fluxo de navegação, executando uma atividade no compartimento de projeto, sendo que, no caso ilustrado, trata-se da atividade de abertura de um projeto a partir da lista de todos os projetos do usuário, sinalizada pela letra "B" cinza. Além dessa atividade, também é ilustrada a atividade de abertura de diagrama a partir de uma lista de diagramas, cujo início é indicado pela letra "C" azul marinho, que é realizada em um dos compartimentos da tela do projeto aberto. Outros detalhes a respeito das atividades de abertura de projeto e diagrama são fornecidos nas Seções 6.2.3.2 e 6.2.3.6, respectivamente.

7.9 Distribuição dos Elementos

Para compreender como interagir e localizar as funcionalidades do ambiente Peônia é necessário entender como estão distribuídos os elementos que o compõem. Na Figura 7.2 da Seção 7.2, é apresentado o modelo conceitual do ambiente Peônia, sendo que os 22 conceitos apresentados podem ser separados em dois tipos: conceitos principais e sub-conceitos.

O desenvolvimento e divisão da apresentação do ambiente Peônia foram realizados em torno dos conceitos principais, sendo eles: *Artefato*, *Atividade*, *Classe de Projeto*, *Diagrama*, *Domínio*, *Fase*, *Padrão de Software*, *Papel*, *Processo de Desenvolvimento*, *Projeto*, *Requisito de Teste* e *Usuário*. Esses conceitos principais são denominados *Elementos* do ambiente Peônia.

Os sub-conceitos foram assim classificados por comporem, ou definirem, algum dos elementos do ambiente Peônia, além de não serem apresentados separadamente ao usuário. São sub-conceitos: *Administrador*, *Atributo*, *Classe de Equivalência*, *Diagrama de Classe*,

Email Adicional, *Engenheiro de Software*, *Estrutura da Solução*, *Modelo Conceitual*, *Operação* e *Valor Limite*

Os elementos podem ser compostos de outros elementos, ou podem conter ou ser classificados por sub-conceitos. Esses sub-conceitos também podem ser compostos de outros sub-conceitos. A relação de dependência entre esses conceitos é detalhada abaixo:

- *Processo de Desenvolvimento* é composto por *Fase*;
- *Fase* é composta por *Atividade*;
- *Atividade* pode receber ou produzir *Artefato*;
- *Atividade* pode ser atribuída a um *Papel* executor;
- *Padrão de Software* pode pertencer a um *Domínio*;
- *Padrão de Software* pode conter uma *Estrutura da Solução*;
- *Diagrama* pode ser um *Modelo Conceitual* ou um *Diagrama de Classes*;
- *Classe de Projeto* pode conter *Atributo* e *Operação*;
- *Estrutura da Solução*, *Diagrama de Classes* e *Modelo Conceitual* são compostos por *Classe de Projeto*;
- *Requisito de Teste* pode não ser classificado, ou ser uma *Classe de Equivalência* ou *Valor Limite*;
- *Classe de Equivalência* pode ser composta por *Valor Limite*;
- *Projeto* utiliza *Padrão de Software*, *Processo de Desenvolvimento* e *Requisito de Teste* durante a sua execução.
- *Usuário* pode ser do tipo *Engenheiro de Software* ou *Administrador*.

Com relação à interface visual do ambiente Peônia, os elementos são ditribuídos em abas, seguindo o relacionamento entre conceitos listado. Dentro dessas abas estão todas as informações e funcionalidades para manipular os elementos e seus sub-conceitos. A opção de utilizar abas foi realizada por permitir que diversos conceitos sejam manipulados e visualizados ao mesmo tempo, facilitando e agilizando a consulta e gerência.

As abas do ambiente Peôna são divididas em níveis. Na Figura 7.17, é ilustrado o primeiro nível de abas apresentado a um usuário do tipo *Administrador*. Nessa figura, a seta "A" vinho indica a aba contendo informações sobre o elemento *Projeto*; a seta "B" cinza aponta para aba de *Processo de Desenvolvimento*; a seta "C" azul marinho indica a

aba que deve ser selecionada para que as funcionalidades envolvendo *Padrão de Software* possam ser executadas; a seta "D" preta mostra a aba de *Domínio*; a seta "E" verde sinaliza a aba de *Requisito de Teste*; a seta "F" rosa aponta para o elemento *Classe de Projeto*; a seta "G" lilás indica a aba de controle de *Usuário*, que só aparece para *Administradores*; a seta "H" amarela sinaliza a aba de pesquisa; e a seta "I" marrom abre a aba com dicas para ajudar o usuário na utilização do ambiente Peônia.

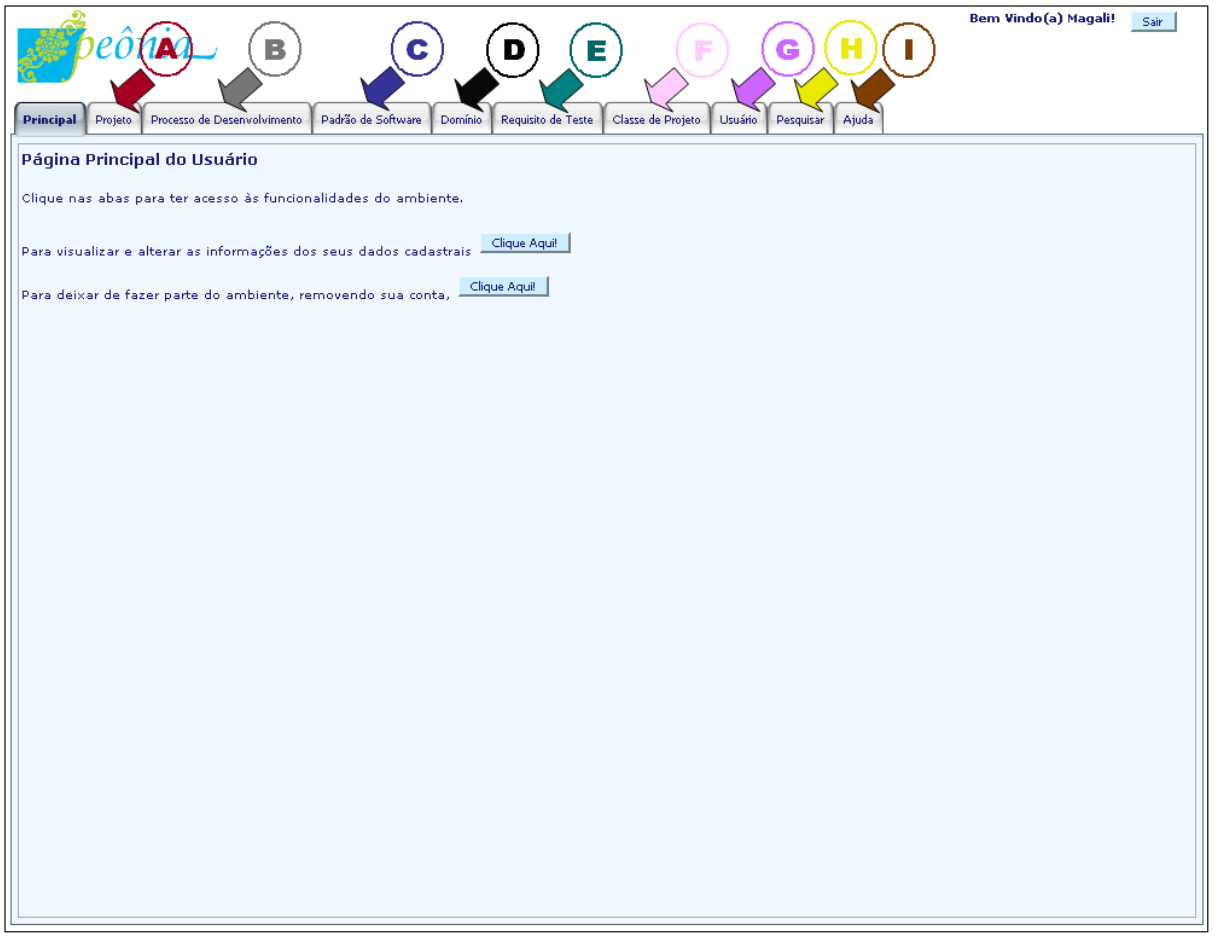

Figura 7.17: Tela do primeiro nível de abas do ambiente Peônia.

Para ilustrar a diferença entre o primeiro nível das abas de usuários do tipo *Administrador* e *Engenheiro de Software*, a seta "J" laranja destaca na Figura 7.18 a ausência da aba de controle de *Usuário*.

Ao abrir um projeto na aba de gerência de *Projeto*, indicada pela seta de cor "A" vinho na Figura 7.17, o segundo nível de abas de um *Projeto* é exibido com os seus conceitos e funcionalidades. Na Figura 7.19, esse segundo nível de abas do elemento *Projeto* é ilustrado, sendo que a seta "L" vinho mostra a aba para controle de *Diagrama*. As outras aba são utilizadas para auxiliar a visualização e gerenciamento de informações da execução do projeto. Como o objetivo do ambiente Peônia é estimular o emprego de padrões de

software durante cada uma das etapas de um processo de desenvolvimento, duas janelas na parte esquerda da tela são apresentadas na execução do projeto. Como mencionado na Seção 6.2.2.7, a janela superior, apontada pela seta "M" cinza na Figura 7.19, contém todos as fases, atividades e artefatos obrigatórios e opcionais do processo em desenvolvimento utilizado no projeto. A janela inferior, sinalizada pela seta "N" azul marinho na Figura 7.19, contém a lista dos padrões de software cadastrados no ambiente Peônia e, de acordo com a interação do usuário, apresenta os padrões associados a cada uma das etapas do processo de desenvolvimento. As funcionalidades de abertura de projeto e gerenciamento de projeto são detalhadas na Seção 6.2.3.2.

Figura 7.18: Tela da página inicial da sessão de um usuário do tipo *Engenheiro de Software*.

Outro elemento que possui um segundo nível de abas é o *Processo de Desenvolvimento*, acessado ao selecionar a aba indicada pela seta "B" cinza na Figura 7.17. É por meio desse segundo nível que são criados e gerenciados os elementos que compõem um processo de desenvolvimento, além de permitir a visualização dos repositórios de processos. Na Figura 7.20, que ilustra o segundo nível de abas do *Processo de Desenvolvimento*, a seta "A" vinho indica a aba com o *Repositório do Usuário* e a seta "B" cinza aponta para o *Repositório do Ambiente*, ambos detalhados na Sessão 6.2.2.7; a seta "C" azul marinho mostra a aba do elemento *Fase*; a seta "D" preta sinaliza a aba com as funcionalidades de *Atividade*; a seta "E" verde indica a aba de *Artefato*; e a seta "F' rosa aponta para a aba de *Papel*.

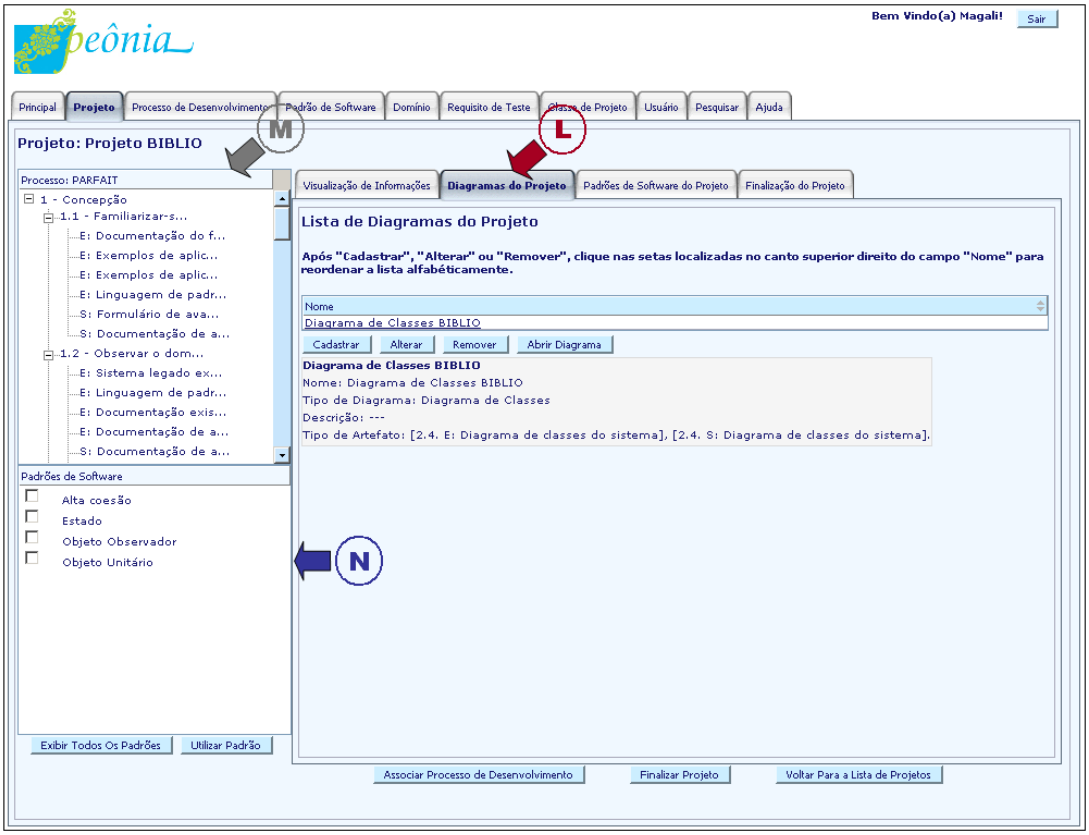

Figura 7.19: Tela do segundo nível de abas de um *Projeto* aberto.

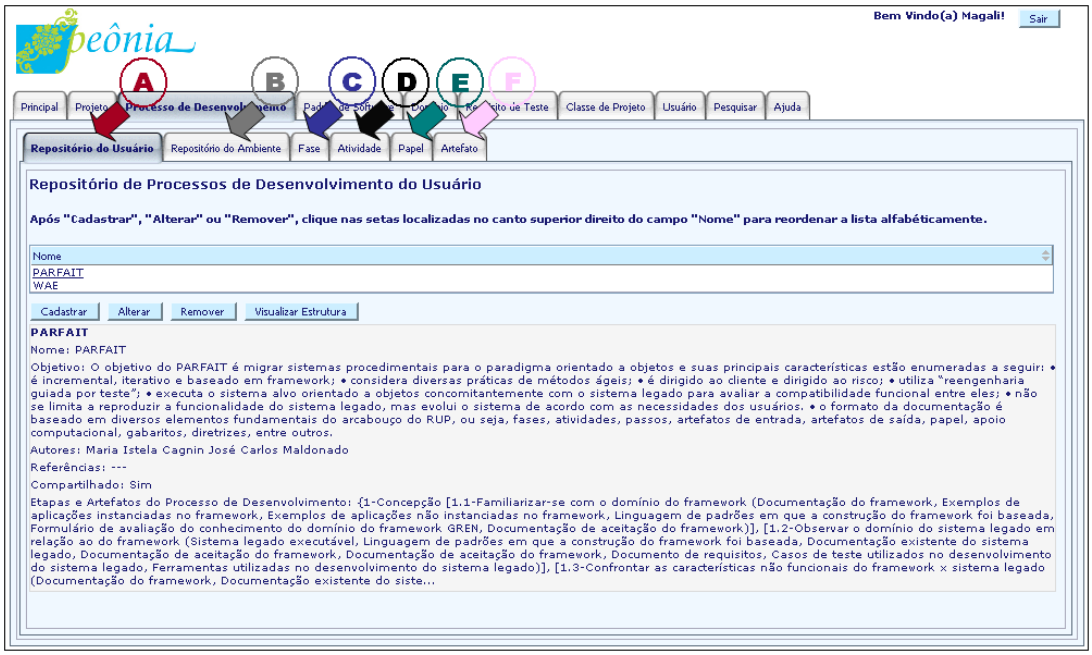

Figura 7.20: Tela do segundo nível de abas de um *Processo de Desenvolvimento*

7.10 Implementação e Teste

Como mencionado anteriormente, optou-se pelo desenvolvimento de uma aplicação para a *Web*. Assim, foram utilizados os *frameworks* Hibernate e ZK para apoiar a implementação do ambiente Peônia. Optou-se por utilizar Java (Sun Microsystems, Inc., 1999), por ser uma linguagem de programação orientada a objetos, simples, robusta, interpretada, portável, distribuída, de arquitetura neutra, segura, de alto desempenho, *multithreaded* e dinâmica (Sun Microsystems, Inc., 1999).

O *framework* Hibernate utiliza padrões de projeto como o *Session Façade* (Sun Microsystems, Inc., 2003a), que evita que os componentes da camada de negócio sejam acessados diretamente fornecendo uma interface para acesso. Empregando o *framework* Hibernate não foi necessário se preocupar com aspectos de comunicação com o SBDG. Assim, foi possível focar as atenções na Camada de Aplicação do ambiente Peônia.

Com relação aos testes, foram realizados testes unitários, funcionais e de integração. Por causa do pouco tempo disponível, não foi possível realizar testes estruturais e de estresse, no entanto, planeja-se utilizar em breve alguma ferramenta de automatização de teste para realizar uma avaliação mais robusta da implementação do ambiente Peônia.

Além disso, o ambiente Peônia foi utilizado por alunos de uma disciplina do curso de pós-graduação do ICMC da USP. Esses alunos testaram o funcionamento da aplicação e os erros encontrados foram corrigidos. Adicionalmente, forneceram sugestões que serão listadas na Seção de 8.4.

7.11 Considerações Finais

Neste capítulo foi apresentado o projeto executado para desenvolver o ambiente Peônia. Optou-se por criá-lo como uma aplicação *Web* para explorar vantagens como facilidade de divulgação, acesso e compartilhamento de informações, permitindo que usuários troque informações sobre padrões de software.

A partir do problema encontrado por usuários no emprego de padrões de software, foram levantadas 168 funcionalidades que foram analisadas por meio dos diagramas de casos de uso, seqüência, atividades e classes. Levando em consideração a interface e experiência do usuário, foram criados modelos UX para definir o conteúdo das telas e mapa de navegação do ambiente Peônia.

Apesar de Conallen (2002) não ter recomendado em seu processo a criação de modelos das tabelas de um banco de dados, por causa da complexidade da aplicação, foi necessário criar o MER e MRel para poder entender o relacionamento entre as entidades do banco de dados do ambiente Peônia.

Tomando por base as informações dos modelos e diagramas criados, foi realizada a implementação do sistema seguido de teste. O ambiente Peônia também foi testado e avaliado por alunos de uma disciplina de pós-graduação do ICMC da USP, permitindo capturar erros e opiniões para contribuições futuras.

Capítulo

8

Conclusão

8.1 Resumo do Trabalho Realizado

Como descrito na Seção 1.3, este trabalho tem por meta fornecer um método e um ambiente para apoiar o emprego de padrões de software durante a execução de um processo de desenvolvimento.

Assim, foi desenvolvido o ambiente Peônia que auxilia usuários na aplicação de padrões de software durante projetos orientados por processos de desenvolvimento. Observando os projetos durante o teste do ambiente Peônia, notou-se a repetição da seqüência de atividades desenvolvidas e de como ocorre o estímulo ao emprego de padrões de software, sendo proposto o Método Para Desenvolvimento Utilizando Padrões de Software.

Para cumprir esses objetivos foi feita uma pesquisa na literatura a respeito de padrões que auxiliem o desenvolvimento de aplicações voltadas para a *Web*, como por exemplo, os mencionados em Bonura et al. (2002); Danculovic et al. (2001); Garzotto et al. (1999); Koch e Rossi (2002); Weiss (2003), além de estudos sobre processos e métodos de desenvolvimento e arquiteturas desse tipo de aplicações, para compreender o seu comportamento. Também foram feitos estudos sobre a linguagem Java, utilizada para o desenvolvimento do ambiente Peônia, além de *frameworks* que apóiam a implementação de aplicações *Web*.

Para avaliar as funcionalidades comumente oferecidas por ferramentas de automatização, foram realizados estudos sobre ferramentas que apoiam a utilização de padrões de software (Bolchini et al., 2002; Braga, 2003; Braga e Masiero, 2000; Garzotto et al., 1999;

Hakala et al., 2001b; Marinho et al., 2003), auxiliam a aplicação de processos de desenvolvimento (Bertollo et al., 2006; Lima et al., 2006); ou forneçam suporte ao emprego de métodos de desenvolvimento *Web* (Brito, 2003; Knapp et al., 2003, 2004a, 2005, 2004b; Schwabe e Pontes, 1998; Schwabe et al., 1999; Turine et al., 1998, 1999; Web Models, 2006).

De acordo com esse levantamento, foram realizados estudos sobre as funcionalidades que já são oferecidas atualmente e as que não são implementadas, permitindo averiguar quais devem ser incluídas no ambiente proposto neste trabalho. Ferramentas que auxiliam o teste de software também foram pesquisadas, mas sem a intenção de realizar uma análise profunda sobre as funcionalidades de automatização de teste, pois a prioridade era o apoio dado ao emprego de processos de desenvolvimento e padrões de software.

Foram feitos estudos de estratégias de teste da técnica funcional ou caixa-preta (Myers, 2004; Pressman, 2002), que focaliza os requisitos funcionais do software, tomando por base a especificação e não a implementação, e da técnica estrutural ou caixa-branca (Myers, 2004; Pressman, 2002), que utiliza a estrutura de controle do projeto procedimental para derivar casos de teste, verificando se atendem aos detalhes do código e solicitando a execução de partes ou componentes elementares do programa (Pressman, 2002; Rocha et al., 2001).

Pesquisas sobre critérios de testes específicos de aplicações *Web*, como os citados em Ricca e Tonella (2001); Xu et al. (2005), foram realizadas para levantar os requisitos necessários para criar um conjunto de testes que validasse adequadamente o ambiente Peônia. No entanto, não foi possível concluir o levantamento para realizar o planejamento do teste das características específicas de aplicações *Web*.

Para compreender e desenvolver um mecanismo de associação de requisitos de teste a padrões de software, como proposto por (Cagnin et al., 2005), fez-se a análise das técnicas funcionais de Particionamento de Equivalência, Análise do Valor Limite e Teste Funcional Sistemático. Nesse projeto foi considerado apenas o teste funcional, pois padrões de software possuem obrigatoriamente ao menos uma descrição do problema e solução, fornecendo as condições para a criação dos requisitos de teste que validem a parte funcional dos projetos desenvolvidos utilizando o padrão.

Como o ambiente Peônia foi desenvolvido como uma aplicação *Web*, optou-se por explorar as vantagens oferecidas pela linguagem de programação Java. Assim, foi feita uma busca por *frameworks* que apoiassem a implementação do ambiente Peônia utilizando Java.

Com relação à parte de persistência, os *frameworks* Hibernate e iBATIS foram analisados quanto às suas vantagens e desvantagens. Ao final do estudo, o *framework* Hibernate foi eleito para realizar o gerenciamento da comunicação com o SGBD.

Diversos *frameworks* podem ser encontrados na literatura para auxiliar a implementação da interface visual de aplicações *Web*, no entanto, optou-se por utilizar o *framework* ZK principalmente, pois permite que informações sejam adicionadas dinamicamente em uma página já renderizada, além da facilidade de aprendizado e a rica documentação disponibilizada sobre os seus componentes.

Por causa da integração com outras ferramentas foi iniciado o estudo do padrão XMI (*Extensible Markup Language Metadata Interchange*) (World Wide Web Consortium, 2007a), permitindo realizar a exportação e importação de diagramas.

8.2 Contribuições

A principal contribuição deste trabalho é o ambiente Peônia que fornece flexibilidade para usuários cadastrarem processos de desenvolvimento e acompanharem a sua execução, sugerindo automaticamente padrões de software para serem empregados nas fases e atividades do processo de desenvolvimento escolhido pelo usuário, além de também apoiar as atividades de VV&T por oferecer requisitos de teste para validar os padrões cadastrados no ambiente.

Outra contribuição é o Método Para Desenvolvimento Utilizando Padrões de Software, que formaliza o método empregado no ambiente Peônia para o projeto de aplicações, incentivando a utilização de padrões de software durante as etapas de um processo de desenvolvimento.

Com a associação prévia de padrões de software às fases e atividades de um processo, usuários são estimulados a pelo menos considerar a possibilidade de utilizar esses padrões associados durante a execução da etapa no desenvolvimento, além de tentar reduzir as chances de usuários deixarem de utilizá-los por não recordarem a existência das soluções.

Acredita-se que a definição forçada de um formato comum, realizada por meio dos campos obrigatórios para cadastro de padrões de software no ambiente Peônia, possa facilitar a busca e visualização. A utilização opcional da classificação por domínio também é oferecida pelo ambiente Peônia com o intuito de facilitar também a localização de padrões de software de acordo com a natureza do projeto.

Com a associação de requisitos de teste a padrões de software, é dado auxílio à validação dos padrões de software utilizados em um projeto, podendo minimizar o tempo despendido nas atividades de VV&T. Também é oferecida uma funcionalidade de geração automática de requisitos de teste do tipo classe de equivalência a partir de valores limites cadastrados, agilizando e auxiliando na elaboração dos testes.

Outra colaboração são os repositórios de padrões de software e processos de desenvolvimento que são disponibilizados no ambiente Peônia e, por ser uma aplicação disponibilizada na *Web*, amplia a divulgação e troca de informações entre usuários.

Por ter sido feita de maneira direcionada às necessidades do ambiente Peônia, uma contribuição indireta e de menor relevância foi o estudo realizado sobre os *framework* de persistência durante o desenvolvimento do ambiente Peônia. Os *frameworks* Hibernate e iBATIS são as soluções de código aberto mais utilizadas atualmente, sendo que o Hibernate possui uma documentação rica e organizada, garantias de controle de concorrência, geração automática de tabelas e oferece uma linguagem orientada a objetos de manipulação de banco de dados. Espera-se que a comparação realizada seja um estímulo para que sejam feitos estudos experimentais sobre cada uma das características dos *frameworks* de persistência existente, pois é crescente a difusão, interesse e utilização desses *frameworks* no desenvolvimento de aplicações.

8.3 Limitações

Primeiramente, o objetivo era desenvolver um ambiente que apoiasse o desenvolvimento *Web* utilizando métodos e padrões de software específicos para esse tipo de aplicação. No entanto, foi possível observar que o apoio poderia ser estendido para outros contextos, onde a meta deveria ser auxiliar o desenvolvimento de aplicações utilizando padrões de software e requisitos de teste. Assim, o conceito de acompanhamento de processo foi adicionado ao ambiente Peônia.

Assim, com a mudança do foco da pesquisa, não foi possível analisar todos os pontos desejáveis sobre processos de desenvolvimento, sendo fornecida a estrutura básica de acordo com o RUP para permitir o cadastro.

Atentou-se em observar qual a melhor maneira de aplicar padrões de software, chegando a conclusão que ligando a utilização a processos de desenvolvimento facilitaria na distinção, de acordo com a fase ou atividade executada, de um conjunto de padrões que devem ao menos ser analisados. Também foi dada atenção ao projeto do ambiente Peônia para que a implementação utilizasse recursos simples, porém portáveis, e fosse facilmente estendida.

No entanto, por causa do pouco tempo restante disponível não foi possível desenvolver as funcionalidades que permitissem a integração com outras ferramentas, como para exportar e importar diagramas; a manipulação gráfica ou estudo de ferramentas e *plugins* que fornecessem apoio a interface visual dos componentes UML cadastrados no ambiente Peônia; e permitir o cadastro e criação de diversos critérios de teste, pois foi possível

implementar apoio à Análise do Valor Limite, Particionamento de Equivalência e Teste Funcional Sistemático.

Também por causa da restrição de cronograma, não foi possível realizar o estudo de caso controlado experimentalmente, sendo realizada avaliação apenas das funcionalidades por meio de uma disciplina de pós-graduação ministrada no ICMC da USP.

Apesar do ambiente Peônia restringir o formato com o intuito de auxiliar usuários na visualização e pesquisa de padrões de software, isso pode dificultar a atividade de cadastro, pois muitos padrões de software não possuem os campos exigidos, como por exemplo, os padrões propostos por Gamma et al. (1995). Não há um consenso sobre um formato adequado e os campos obrigatórios propostos não têm a intenção de ser a ideal. No entanto, a padronização de informações cadastradas foi realizada como uma maneira de garantir a qualidade dos padrões inseridos, além de tentar facilitar a pesquisa, visualização e assimilação das informações.

Com relação à qualidade dos padrões inseridos, não há um sistema para filtrar os padrões de software cadastrados. Assim, usuários podem atualmente inserir informações que podem não compor um padrão. Há apenas uma tentativa de evitar que faltem informações essenciais com a definição de campos obrigatórios.

8.4 Trabalhos Futuros

O ambiente Peônia está sendo desenvolvido de maneira incremental, adicionando funcionalidades à medida que são implementadas.

Durante o levantamento de requisitos houve a preocupação de considerar maneiras que permitissem integrar as informações desenvolvidas no ambiente Peônia com outras ferramentas. Assim, foram incluídas no documento de requisitos as funcionalidades de exportar e importar diagramas (*Atividades: Exportar Diagrama* ou *Importar Diagrama*) seguindo a versão 2.1 do padrão XMI (World Wide Web Consortium, 2005b), que contempla a exportação completa de diagramas. No entanto, não foi possível implementar tais funcionalidades por causa do tempo disponível, apesar de ter sido iniciado o estudo das *tags* necessárias para representar cada elemento do diagrama.

Existe a possibilidade de realizar a exportação e importação de padrões de software, sendo necessário estudar a viabilidade e pesquisar ferramentas que implementem essas funcionalidades para avaliar como realizar a troca de informações.

Também considerou-se a necessidade de uma interface visual na criação de classes de projeto, estruturas da solução e diagramas, no entanto, o planejamento já estimava que não seria possível implementar esse requisito dentro deste projeto de mestrado. A parte gráfica é importante principalmente no formato de interação difundido atualmente, que segue o

padrão UML da OMG (OMG's, 2007), pois o usuário já está familiarizado, dificultando a aceitação de envio textual. Essa dificuldade de manipulação dos dados via formulário textual foi citada na avaliação realizada pelos alunos da disciplina de pós-graduação, onde o ambiente Peônia foi estudado, e foi sugerido, como esperado, o desenvolvimento da interface gráfica.

Assim, planeja-se adicionar essa parte gráfica, sendo que devem ser feitos estudos sobre qual a melhor forma para fazê-lo, podendo ser incluído diretamente no ambiente Peônia, ou permitindo o uso embutido de uma ferramenta que trate especificamente dessa parte.

Além disso, foi levantado pelos alunos que seria interessante a possibilidade de criar sub-atividades; a atribuição de múltiplos papéis a uma atividade; e a necessidade de alimentar com mais elementos o repositório de padrões de software e de processos de desenvolvimento. Todos esses pontos devem ser futuramente considerados.

Nesse primeiro momento, apenas é fornecido apoio à construção de diagramas de classes e modelo conceitual e ao cadastro de requisitos de teste utilizando os critérios Análise do Valor Limite, Particionamento de Equivalência e Teste Funcional Sistemático. Assim, para as fases e atividades que não envolvem a construção de diagramas de classes ou modelo conceitual, o usuário pode utilizar o ambiente para acompanhar e controlar o emprego de padrões. No entanto, em trabalhos futuros planeja-se adicionar apoio à construção de outros artefatos para que o suporte ao emprego de padrões torne-se mais efetivo gradativamente. Também se espera que, além de apoio ao cadastro a outros critérios de teste, o ambiente também forneça um suporte maior à validação de padrões de software utilizados no desenvolvimento de aplicações, como por exemplo, a criação de casos de teste.

As funcionalidades de pesquisas atualmente oferecidas são restritas a busca de acordo com o tipo de elemento desejado e é realizada apenas pelo campo "Nome". Já está sendo desenvolvida uma busca avançada onde o usuário poderá realizar a consulta por outros campos do padrão de software, como o executado pelo repositório proposto por Marinho et al. (2003), ilustrado na Figura B.2, Apêndice B.

Um ponto a ser considerado é a qualidade dos padrões armazenados no ambiente. Futuramente planeja-se estabelecer um filtro no cadastro ou um critério de remoção para que apenas padrões de software válidos sejam mantidos e repassados entre os usuários, evitando que soluções inválidas comprometam os projetos.

Planeja-se, também, implementar um sistema de controle, ou liberação, automática de usuários que requisitam um cadastro no ambiente Peônia.

Estudos de caso também devem ser realizados para avaliar o emprego do ambiente Peônia no desenvolvimento de aplicações e o Método Para Desenvolvimento Utilizando

Padrões de Software, além de ser útil para que as pessoas submetidas ao teste forneçam opiniões sobre as funcionalidades disponibilizadas e encontrem possíveis inconsistências.

Referências

ACM SIGWEB Officers and Executive Committee ACM Special Interest Group on Hypertext, Hypermedia and the Web. Online, 1998.
Disponível em `http://www.sigweb.org` (Acessado em 17/02/2008)

Adobe Systems Incorporated What is flash cs3 professional? Online, 2008.
Disponível em `http://www.adobe.com/uk/products/flash/ssi/iframe/product_overview.html` (Acessado em 17/02/2008)

Alain Zarli and Yacine Rezgui A survey of internet-oriented technologies for document-driven applications in construction open dynamic virtual environments. 2000.
Disponível em `http://cic.cstb.fr/Ilc/publicat/zarli-rezgui-cibw78.pdf` (Acessado em 22/01/2006)

Alexander, C. *A Pattern Language.* Oxford University Press, 1977.

Alexander, C. *The Timeless Way of Building.* Oxford University Press, 1979.

Ambler, S. W. The Process Patterns Resource Page. Online, 2006.
Disponível em `http://www.ambysoft.com/processPatternsPage.html` (Acessado em 17/02/2008)

Andrade, R. M. C. *Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems.* Tese de Doutoramento, SITE/University of Ottawa, Ottawa/Ontario - Canada, 2001.

Appleton, B. Patterns and software: Essential concepts and terminology. Online, 2000.
Disponível em `http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html` (Acessado em 17/02/2008)

Araújo, A. C. M. *Framework de análise e projeto baseado no RUP para o desenvolvimento de aplicações para a web.* Dissertação de Mestrado, Cin/UFPE, Recife/PE - Brasil, 2001.

Baresi, L.; Garzotto, F.; Paolini, P. Extending uml for modeling web applications. In: *34th Annual Hawaii International Conference on System Sciences*, 2001, p. 1–10.

Bauer, C.; King, G. *Hibernate in Action.* 1th. ed. Manning Publications Co., 2004.

Ben Goodger and Ian Hickson and David Hyatt and Chris Waterson XML User Interface Language (XUL) 1.0. Online, 2001.
Disponível em `http://www.mozilla.org/projects/xul/xul.html` (Acessado em 17/02/2008)

Berners-Lee, T. WWW: Past, Present, and Future. *Computer*, v. 29, n. 10, p. 69–77, 1996.

Bertollo, G.; Segrini, B. M.; Falbo, R. A. Evoluindo a Definição de Processos de Software em ODE. In: *SBES'06 - XIII Sessão de Ferramentas do SBES*, Florianópolis/SC - Brasil, 2006, p. 109–114.

Bianchini, S. L. *Avaliação de Metodologias de Desenvolvimento de Sistemas Web.* Dissertação de Mestrado, ICMC/USP, São Carlos/SP - Brasil, em andamento, 2007.

Bolchini, D.; Garzotto, F.; Paolini, P.; Lowe, D.; Cantoni, L.; Nanard, J.; Rossi, G.; Schwabe, D.; Ruggeri, R. HPR - Hypermedia Design Patterns Repository. Online, 2002.
Disponível em `http://www.designpattern.lu.unisi.ch/index.htm` (Acessado em 17/02/2008)

Bonura, D.; Culmone, R.; Merelli, E. Patterns for Web Applications. In: *SEKE '02 - Proceedings of the 14th international conference on Software engineering and knowledge engineering*, New York/NY - USA: ACM Press, 2002, p. 739–746.
Disponível em `http://portal.acm.org/citation.cfm?doid=568760.568887` (Acessado em 17/02/2008)

Braga, R. T. V. GREN-Wizard: A Wizard to Instantiate the GREN Framework. Online, 2002.
Disponível em `http://www.icmc.usp.br/~rtvb/GRENWizard.htm` (Acessado em 17/02/2008)

Braga, R. T. V. *Um Processo para Construção e Instanciação de Frameworks Baseados em uma Linguagem de Padrões para um Domínio Específico.* Tese de Doutoramento, ICMC/USP, São Carlos/SP - Brasil, 2003.

Braga, R. T. V.; Germano, F. S. R.; Masiero, P. C. A Pattern Language for Business Resource Management. In: *PLoP'1999, Conference on Pattern Languages of Programs*, 1999, p. 1–33.
Disponível em `http://www.icmc.sc.usp.br/~rtvb/pat_lang_v3.doc` (Acessado em 17/02/2008)

Braga, R. T. V.; Germano, F. S. R.; Masiero, P. C. Uma Linguagem de Padrões para Gestão de Recursos de Negócios. Documento de Trabalho, 2000.
Disponível em `http://www.icmc.sc.usp.br/~rtvb/linguagem_padroes_grn.zip` (Acessado em 17/02/2008)

Braga, R. T. V.; Germano, F. S. R.; Masiero, P. C.; Maldonado, J. C. Introdução aos Padrões de Software. Nota Didática, 2001.
Disponível em `http://sugarloafplop2005.icmc.usp.br/NotasDidaticasPadroes.pdf` (Acessado em 17/02/2008)

Braga, R. T. V.; Masiero, P. C. A Framework based on a Pattern Language for Business Resource Management. In: *SBES'2000, Anais do V WTES - Workshop de Teses e Dissertacoes em Engenharia de Software*, 2000, p. 385–390.
Disponível em `http://www.icmc.sc.usp.br/~rtvb/wtsbes2000.pdf` (Acessado em 17/02/2008)

Braga, R. T. V.; Masiero, P. C. The role of pattern languages in the instantiation of object-oriented frameworks. In: *OOIS '02 - Proceedings of the Workshops on Advances in Object-Oriented Information Systems*, Montpellier - France: Springer-Verlag, 2002, p. 122–131.

Brambilla, M.; Comai, S.; Fraternali, P. Hypertext Semantics For Web Applications. In: *SEBD Italian National Conference on DataBase Systems*, Portoferraio - Italy, 2002.

Brito, L. S. F. *WebSCharts: Uma Ferramenta de Desenvolvimento de Aplicações Web Baseada no HMBS/M.* Dissertação de Mestrado, UFMS, Campo Grande/MS - Brasil, 2003.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *Pattern-oriented software architecture: a system of patterns.* New York/NY - USA: John Wiley & Sons, Inc., 1996.

Buschmann, F.; et al. Pattern-Oriented Software Architeture. In: *European Conference on Object-Oriented Programming*, 1997.

Cagnin, M. I. *PARFAIT: Uma Contribuição Para a Reengenharia de Software Baseada em Linguagem de Padrões e Frameworks*. Tese de Doutoramento, ICMC/USP, São Carlos/SP - Brasil, 2005.

Cagnin, M. I.; Braga, R. T. V.; Germano, F.; Chan, A.; Maldonado, J. C. Extending Patterns with Testing Implementation. In: *SugarLoafPlop'2005, V Conferência Latino-Americana em Linguagens de Padrões para Programação*, Campos do Jordão/SP - Brasil, 2005.

Cagnin, M. I.; Maldonado, J. C.; Chan, A.; Penteado, R. D.; Germano, F. S. Reuso na Atividade de Teste para Reduzir Custo e Esforço de VV&T no Desenvolvimento e na Reengenharia de Software. In: *XVIII Simpósio Brasileiro de Engenharia de Software*, Brasília/DF - Brasil, 2004, p. 71–85.

Carvalho, M. R. *HMBS/M - um método orientado a objetos para o projeto e o desenvolvimento de aplicações hipermídia*. Dissertação de Mestrado, ICMC/USP, São Carlos/SP - Brasil, 1998.

Carvalho, M. R.; Oliveira, M. C. F.; Masiero, P. C. HMBS/M - an object oriented method for hypermedia design. In: *V Simpósio Brasileiro de Multimídia e Sistemas Hipermídia (SBMIDIA'99)*, Goiânia/GO - Brasil, 1999, p. 43–62.

Ceri, S.; Fraternali, P.; Bongio, A. Web modeling language (WebML): a modeling language for designing web sites. In: *9th International World Wide Web Conference*, Amsterdam, 2000, p. 1–22.

Chaim, M. L. *Poke-tool: Uma Ferramenta Para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados*. Dissertação de Mestrado, DCA/FEEC/UNICAMP, Campinas/SP - Brasil, 1991.

Chan, A. *Um Ambiente de Apoio ao Uso de Padrões de Software e Requisitos de Teste no Desenvolvimento de Aplicações Web*. Dissertação de Mestrado, ICMC/USP, São Carlos/SP - Brasil, qualificação, 2005.

Cho, E. S.; Kim, S. D.; Rhew, S. Y.; Lee, S. D.; Kim, C. G. Object-Oriented Web Application Architectures and Development Strategies. In: *APSEC '97/ICSC '97 - 4th Asia-Pacific Software Engineering and International Computer Science Conference*, Clear Water Bay - Hong Kong, 1997, p. 322–331.

Conallen, J. *Buildind Web applications with UML*. 2nd. ed. Addison-Wesley, 2002.

Coplien, J. O. A Development Process Generative Pattern Language. In: Coplien, J. O.; Schmidt, D., eds. *Pattern Languages of Program Design*, Addison-Wesley, p. 183–237, 1995.

Corriveau, J.; P.; Nel, D. N. Introduction to Object-Oriented Software Engineering Version 1.0. Course Notes, 1997.

Crane, D.; Pascarello, E.; James, D. *Ajax in action*. Greenwich/CT - USA: Manning Publications Co., 2006.

Danculovic, J.; Rossi, G.; Schwabe, D.; Miaton, L. Patterns for Personalized Web Applications. In: *EuroPLoP'2001 - Proceedings of the 6th. European Conference on Pattern Languages of Programs*, Irsee - Germany, 2001.

David Gardner What is DHTML? Online, 2000.
Disponível em `http://www.irt.org/articles/js201/index.htm` (Acessado em 22/01/2006)

Delamaro, M. E. *Proteum - Um ambiente de teste baseado na analise de mutantes*. Dissertação de Mestrado, ICMC/USP, São Carlos/SP - Brasil, 1993.

Domingues, A. L. S. *Aplicações Web: Definição e Análise de Recursos de Teste e Validação*. Tese de Doutoramento, ICMC/USP, São Carlos/SP - Brasil, em andamento, 2005.

Falbo, R. A.; Ruy, F. B.; Pezzin, J.; Moro, R. D. Ontologias e Ambientes de Desenvolvimento de Software Semânticos. In: *JIISIC'04 - IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento*, Madri - Espanha, 2004.

Fayad, M. E.; Johnson, R. E. *Domain-specific application frameworks: framework experience by industry*. New York/NY - USA: John Wiley & Sons, Inc., 2000.

Figueiredo, A. *ECO - um ecossistema para o desenvolvimento ágil de sistemas web*. Dissertação de Mestrado, ICMC/USP, São Carlos/SP - Brasil, 2005.

Fons, J.; Pelechano, V.; Pastor, O.; Albert, M.; Valderas, P. Extending an OO Method de Develop Web Applications. In: *The Twelfth International World Wide Web Conference*, 2003.

Fuggetta, A. Software Process: A Roadmap. In: *ICSE'00 - Future of Software Engineering Track*, Limerick - Ireland, 2000, p. 25–34.

Gall, H. C.; Klösch, R. R.; Mittermeir, R. T. Application patterns in re-engineering: Identifying and using reusable concepts. In: Bouchon-Meunier, B.; Delgado, M.; Verdegay, J. L.; Vila, M. A.; Yager, R. R., eds. *Proceedings of the 6th. International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Springer-Verlag, 1996, p. 1099–1106.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable of Object-Oriented Software*. 2th. ed. Addison-Wesley, 1995.

Garzotto, F.; Paolini, P.; Bolchini, D.; Valenti, S. "Modeling-by-Patterns" of Web Applications. In: *ER '99, Proceedings of the Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling*, London, UK: Springer-Verlag, 1999, p. 293–306.

Garzotto, F.; Paolini, P.; Schwab, D. HDM - a model for the design of hypertext applications. In: *Third Annual ACM Conference on Hypertext (Hypertext'91)*, New York/NY - USA, 1991, p. 313–328.

Garzotto, F.; Paolini, P.; Schwabe, D. HDM - a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, v. 11, n. 1, p. 1–26, 1993.

Ginige, A.; Murugesan, S. The essence of web engineering. *IEEE MultiMedia*, v. 8, n. 2, p. 22–25, 2001a.

Ginige, A.; Murugesan, S. Web engineering: An introduction. *IEEE MultiMedia*, v. 8, n. 1, p. 14–18, 2001b.

Gómez, J.; Cachero, C.; Pastor, O. Conceptual Modeling of Device-Independent Web Applications. *IEEE MultiMedia*, v. 8, n. 2, p. 26–39, 2001.

Griffiths, G.; Hebbron, B. D.; Lockyer, M. A.; Oates, B. J. A simple method & tool for web engineering. In: *SEKE'02 - Proceedings of the 14th international conference on Software engineering and knowledge engineering*, New York/NY - USA: ACM Press, 2002, p. 755–762.

Hakala, M.; Hautamäki, J.; Koskimies, K.; Paakki, J.; Viljamaa, A.; Viljamaa, J. Architecture-Oriented Programming Using FRED. In: *ICSE'01 - Proceedings of the 23rd International Conference on Software Engineering*, Washington/DC - USA: IEEE Computer Society, 2001a, p. 823–824.

Hakala, M.; Hautamäki, J.; Koskimies, K.; Paakki, J.; Viljamaa, A.; Viljamaa, J. Generating application development environments for Java frameworks. In: *GCSE'01 - Proceedings of the 3rd International Conference on Generative and Component-Based Software Engineering*, Erfurt - Germany, 2001b, p. 163–176.
Disponível em `http://practise.cs.tut.fi/files/publications/Fred/gcse01.pdf` (Acessado em 17/02/2008)

Harrison, W.; Ossher, H.; Tarr, P. Software engineering tools and environments: a roadmap. In: *ICSE'00 - Proceedings of the Conference on The Future of Software Engineering*, New York/NY - USA, 2000, p. 261–277.

Henninger, S.; Corrêa, V. *Software pattern communities: Current practices and challenges*. Relatório Técnico TR-UNL-CSE-2007-0015, University of Nebraska-Lincoln, 2007.

Horgan, J. R.; Mathur, A. P. Assessing Testing Tools in Research and Education. *IEEE Software*, v. 9, n. 3, p. 61–69, 1992.

Houaiss, A. Dicionário Houaiss da Língua Portuguesa. Online, 2006.
Disponível em `http://houaiss.uol.com.br` (Acessado em 17/02/2008)

iBATIS Team iBATIS Data Mapper Version 2.0: Developer Guide. Online, 2006.
Disponível em `http://ibatis.apache.org/docs/java/pdf/iBATIS-SqlMaps-2_en.pdf` (Acessado em 17/02/2008)

iBATIS Team iBATIS. Online, 2007.
Disponível em `http://ibatis.apache.org/index.html` (Acessado em 17/02/2008)

International Organization for Standardization ISO/IEC 9075:1992: Database Languages SQL. 1992.
Disponível em `http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt` (Acessado em 17/02/2008)

Isakowitz, T.; Stohr, E. A.; Balasubramanian, P. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, v. 18, n. 8, p. 34–44, 1995.

Jacobson, I. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.

Jacyntho, M. D.; Schwabe, D.; Rossi, G. A Software Architecture for Structuring Complex Web Applications. *Journal of Web Engineering*, v. 1, n. 1, p. 37–60, 2002.

Java-Source.Net Open Source Persistence Frameworks in Java. Online, 2007. Disponível em `http://java-source.net/open-source/persistence` (Acessado em 17/02/2008)

Jia, X.; Liu, H. Rigorous and Automatic Testing of Web Applications. In: *SEA 2002 - Proceedings of the 6th IASTED International Conference on Software Engineering and Applications*, Cambridge/MA - USA, 2002, p. 280–285.

Knapp, A.; Koch, N.; Moser, F.; Zhang, G. ArgoUWE: A CASE tool for Web applications. In: *EMSISE´03 - First International Workshop on Engineering Methods to Support Information Systems Evolution*, 2003, p. 1–14.

Knapp, A.; Koch, N.; Zhang, G. Modeling the structure of web applications with ArgoUWE. In: *ICWE 2004 - 4th. International Conference*, Munich - Germany: Springer, 2004a, p. 615–616 (*LNCS*, v.3140).

Knapp, A.; Koch, N.; Zhang, G. Modelling the behaviour of web applications with ArgoUWE. In: *ICWE'05 - 5th International Conference Web Engineering*, Springer, 2005, p. 624–626 (*LNCS*, v.3579).

Knapp, A.; Koch, N.; Zhang, G.; Hassler, H.-M. Modeling business processes in web applications with ArgoUWE. In: *UML'04 - Proceedings of the 7th International Conference on the Unified Modeling Language - the Language and its Applications*, Lisbon - Portugal: Springer, 2004b, p. 69–83 (*LNCS*, v.3273).

Koch, N. *Software engineering for adaptive hypermedia systems: Reference model, modeling techniques and development process.* Tese de Doutoramento, Ludwig-Maximilians University, Munich - Germany, 2000.

Koch, N. OpenUWE - Tool Suite for Web Applications. Online, 2004. Disponível em `http://www.pst.informatik.uni-muenchen.de/projekte/uwe/openuwe.shtml` (Acessado em 17/02/2008)

Koch, N.; Rossi, G. Patterns for Adaptive Web Applications. In: *EuroPLoP'2002 - Proceedings of the 7th. European Conference on Pattern Languages of Programs*, Irsee - Germany, 2002.

Krasner, G. E.; Pope, S. T. A cookbook for using the model view controller user interface paradigm in smalltalk-80. In: *Journal of Object-Orientated Programming*, 1988, p. 26–49.

Kruchten, P. *The Rational Unified Process: An Introduction.* 2th. ed. Addison-Wesley, 298 p., 2000.

Lafon, Y. HTTP - Hypertext Transfer Protocol. Online, 2005. Disponível em `http://www.w3.org/Protocols/` (Acessado em 17/02/2008)

Leavitt, N. Will wap deliver the wireless internet? *Computer*, v. 33, n. 5, p. 16–20, 2000.

Lima, A.; Costa, A.; França, B.; Reis, C. A. L.; Reis, R. Q. Gerência Flexível de Processos de Software com o Ambiente WebAPSEE. In: *SBES'06 - XIII Sessão de Ferramentas do SBES*, Florianópolis/SC - Brasil, 2006, p. 97–102.

Linkman, S.; Vincenzi, A. M.; Maldonado, J. C. Systematic Functional Testing versus Mutation Testing. In: *VII International Conference on Empirical Assessment in Software Engineering*, Staffordshire - UK, 2003, p. 1–15.

Maldonado, J. C. *Critérios potenciais usos: Uma contribuição ao teste estrutural de software*. Tese de Doutoramento, DCA/FEE/UNICAMP, Campinas/SP - Brasil, 1991.

Maldonado, J. C.; Barbosa, E. F.; Vincenzi, A. M. R.; Márcio Eduardo Delamaro, Simone Rocio Senger Souza, M. J. Introdução ao Teste de Software. Nota Didática, 2004. Disponível em `ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/not_did/ND_65.pdf.zip` (Acessado em 17/02/2008)

Marinho, F.; Santos, M.; Pinto, R. N.; Andrade, R. Uma Proposta de um Repositório de Padrões de Software Integrado ao RUP. In: *SugarLoafPlop Proceeding 2003, The Third Latin American Conference on Pattern Languages of Programming*, Porto de Galinhas/PE - Brasil, 2003, p. 277–290.

Meszaros, G.; Doble, J. MetaPatterns: A Pattern Language for Pattern Writing. In: *PLoP'1996 - Proceedings of the 8th Pattern Languages of Programs Conference*, Monticello/Illinois - USA, 1996.

Microsoft Corporation XAML Overview. Online, 2008. Disponível em `http://msdn2.microsoft.com/en-us/library/ms752059.aspx` (Acessado em 17/02/2008)

Mozilla Foundation JavaScript. Online, 2007. Disponível em `http://developer.mozilla.org/en/docs/JavaScript` (Acessado em 17/02/2008)

Murugesan, S.; Deshpande, Y.; Hansen, S.; Ginige, A. Web Engineering: A new discipline for Web-Based System Development. In: *ICSE'99 - Workshop on Web Engineering, Fisrt International Conference on Software Engineering*, University of Western Sidney - Australia, 1999, p. 1–9.

Myers, G. J. *The art of software testing.* 2th. ed. John Wiley & Sons, Inc., 2004.

MySQL AB MySQL AB - The world's most popular open source database. Online, 2006.
Disponível em `http://www.smalltalk.org/main/` (Acessado em 17/02/2008)

OMG's OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. Online, 2007.
Disponível em `http://www.omg.org/docs/formal/07-11-01.pdf` (Acessado em 17/02/2008)

OMG's CORBA CORBA FAQ. Online, 1997.
Disponível em `http://www.omg.org/gettingstarted/corbafaq.htm` (Acessado em 17/02/2008)

Potix Corporation Simple Rich ZK - The Developer's Guide - Version 3.0.0. Online, 2007a.
Disponível em `http://www.zkoss.org/doc/devguide/` (Acessado em 17/02/2008)

Potix Corporation ZK - Simply Ajax And Mobile. Online, 2007b.
Disponível em `http://www.zkoss.org/` (Acessado em 17/02/2008)

Powell, T.; Jones, D.; Cutts, D. *Web site engineering: Beyond web page design.* Prentice Hall, 1998.

Pree, W. *Design patterns for object-oriented software development.* New York/NY - USA: ACM Press/Addison-Wesley Publishing Co., 1995.

Pressman, R. S. *Engenharia de Software.* 5th. ed. McGraw-Hill, 2002.

Pressman, R. S. *Engenharia de Software.* 6th. ed. McGraw-Hill, 2005.

Price, A. M.; Zorzo, A. Visualizando o Fluxo de Controle de Programas. In: *SBES'1990 - IV Simpósio Brasileiro de Engenharia de Software*, Águas de São Pedro/SP - Brasil, 1990.

Ramos, R.; Araújo, J.; Moreira, A.; Castro, J.; Alencar, F.; Penteado, R. Um Padrão para Requisitos Duplicados. In: *SugarLoafPlop'2007 Proceedings, VI Conferência Latino-Americana em Linguagens de Padrões para Programação*, Porto de Galinhas/PE - Brasil, 2007.

Red Hat Middleware, LLC Hibernate: Relational Persistence for Java and .NET. Online, 2006.
Disponível em `http://www.hibernate.org/` (Acessado em 17/02/2008)

Red Hat Middleware, LLC Hibernate API Documentation (3.2.2.ga). Online, 2007a. Disponível em `http://www.hibernate.org/hib_docs/v3/api/index.html` (Acessado em 17/02/2008)

Red Hat Middleware, LLC Hibernate Reference Documentation - Version: 3.2.2. Online, 2007b.
Disponível em `http://www.hibernate.org/` (Acessado em 17/02/2008)

Ricca, F.; Tonella, P. Analysis and testing of Web applications. In: *ICSE '01, Proceedings of the 23rd International Conference on Software Engineering*, Washington/DC - USA: IEEE Computer Society, 2001, p. 25–34.

Rising, L. *The Pattern Almanac 2000*. 1th. ed. Addison-Wesley Publishing Company, 2000.

Rocha, A. R. C.; Maldonado, J. C.; K, C. W. *Qualidade de Software: Teoria e Prática*. 1th. ed. Prentice Hall, 2001.

Rosenberg, D.; Scott, K. *Use case driven object modeling with uml: a practical approach*. Boston/MA - USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

Rossi, G. *Um método orientado a objetos para o projeto de aplicações hipermídia*. Tese de Doutoramento, PUC-Rio, Rio de Janeiro/RJ - Brasil, 1996.

Santos, M. S. *Uma Proposta para a Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas*. Dissertação de Mestrado, UFC, Fortaleza/CE - Brasil, 2004.

Schwabe, D. Resumo do OOHDM. Online, 2005.
Disponível em `http://www.oohdm.inf.puc-rio.br:8668/space/resumo+do+OOHDM` (Acessado em 27/04/2005)

Schwabe, D.; Pontes, R. A. *OOHDM-Web: Rapid prototyping of hypermedia applications in the WWW*. Relatório Técnico MCC 08, PUC-Rio, Rio de Janeiro/RJ - Brasil, 1998.

Schwabe, D.; Pontes, R. A.; Moura, I. OOHDM-Web: an environment for implementation of hypermedia applications in the WWW. *ACM SIGWEB Newsletter*, v. 8, n. 2, p. 18–34, 1999.

Smalltalk.org Welcome to Smalltalk.org. Online, 2004.
Disponível em `http://www.smalltalk.org/main/` (Acessado em 17/02/2008)

Sommerville, I. *Engenharia de Software*. 6th. ed. Addison-Wesley, 2003.

Sun Microsystems and OMG CORBA Technology and the Java 2 Platform, Standard Edition. Online, 2002.
Disponível em http://java.sun.com/j2se/1.4.2/docs/guide/corba (Acessado em 17/02/2008)

Sun Microsystems, Inc. JavaBeans TM. Online, 1997.
Disponível em http://java.sun.com/products/javabeans/docs/spec.html (Acessado em 17/02/2008)

Sun Microsystems, Inc. The Java Language: An Overview. Online, 1999.
Disponível em http://java.sun.com/docs/overviews/java/java-overview-1.html (Acessado em 17/02/2008)

Sun Microsystems, Inc. JDBC. Online, 2002.
Disponível em http://java.sun.com/javase/6/docs/technotes/guides/jdbc/ (Acessado em 17/02/2008)

Sun Microsystems, Inc. Core J2EE Patterns Catalog. Online, 2003a.
Disponível em http://www.corej2eepatterns.com/Patterns2ndEd/index.htm (Acessado em 17/02/2008)

Sun Microsystems, Inc. Java TM 2 Platform, Standard Edition, v 1.4.2 - API Specification. Online, 2003b.
Disponível em http://java.sun.com/j2se/1.4.2/docs/api/ (Acessado em 17/02/2008)

Sun Microsystems, Inc. J2EE FAQ. Online, 2006.
Disponível em http://java.sun.com/j2ee/faq.html (Acessado em 17/02/2008)

Sun Microsystems, Inc. Expression Language. Online, 2007.
Disponível em http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPIntro7.html (Acessado em 17/02/2008)

Sun Microsystems, Inc. Applets. Online, 2008a.
Disponível em http://java.sun.com/applets/ (Acessado em 17/02/2008)

Sun Microsystems, Inc. Enterprise JavaBeans Technology. Online, 2008b.
Disponível em http://java.sun.com/products/ejb/ (Acessado em 17/02/2008)

Sun Microsystems, Inc. Java ME Technology. Online, 2008c.
Disponível em http://java.sun.com/javame/technology/index.jsp (Acessado em 17/02/2008)

Sun Microsystems, Inc. Java Message Service (JMS). Online, 2008d.
Disponível em `http://java.sun.com/products/jms/` (Acessado em 17/02/2008)

Taligent Inc. Building object-oriented frameworks. Online, 1994.
Disponível em `http://www.ide.hk-r.se/~michaelm/fwpages/files/BuildingFrameworks.ps` (Acessado em 17/02/2008)

Tazzoli, R.; Castagna, P.; Campanini, S. E. Towards a Semantic WikiWikiWeb. In: *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima - Japan, 2004.

Tech-FAQ What is a PDA? Online, 2007.
Disponível em `http://www.tech-faq.com/pda.shtml` (Acessado em 17/02/2008)

Turine, M. A. S.; Oliveira, M. C. F.; Masiero, P. C. HySCharts: Um ambiente de autoria e navegação baseado no modelo HMBS. In: *IV Simpósio Brasileiro de Multimídia e Sistemas Hipermídia (SBMIDIA'98)*, Rio de Janeiro/RJ - Brasil, 1998.

Turine, M. A. S.; Oliveira, M. C. F. D.; Masiero, P. C. Hyscharts: A statechart-based environment for hyperdocument authoring and browsing. *Multimedia Tools Application*, v. 8, n. 3, p. 309–324, 1999.

Vincenzi, A. M. R.; Wong, W. E.; Delamaro, M. E.; Maldonado, J. C. JaBUTi: A Coverage Analysis Tool for Java Programs. In: *Sessão de Ferramentas do 17º Simpósio Brasileiro de Engenharia de Software*, Manaus, AM, Brasil, 2003.

Web Models Webratio. 2006.
Disponível em `http://www.webratio.com` (Acessado em 17/02/2008)

Weiss, M. Patterns for Web Applications. In: *PLoP'2003 - The 10th Conference on Pattern Languages of Programs 2003*, Monticello/IL - USA, 2003.
Disponível em `http://jerry.cs.uiuc.edu/~plop/plop2003/Papers/weiss-web.pdf` (Acessado em 17/02/2008)

World Wide Web Consortium CGI: Common Gateway Interface. Online, 1999.
Disponível em `http://www.w3.org/CGI/` (Acessado em 17/02/2008)

World Wide Web Consortium XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). Online, 2002.
Disponível em `http://www.w3.org/TR/xhtml1/` (Acessado em 17/02/2008)

World Wide Web Consortium HyperText Markup Language (HTML) Home Page. Online, 2005a.
Disponível em `http://www.w3.org/MarkUp` (Acessado em 17/02/2008)

World Wide Web Consortium MOF 2.0/XMI Mapping Specification, v2.1. Online, 2005b.
Disponível em `http://www.omg.org/docs/formal/05-09-01.pdf` (Acessado em 17/02/2008)

World Wide Web Consortium MOF 2.0/XMI Mapping Specification, v2.1.1. Online, 2007a.
Disponível em `http://www.omg.org/technology/documents/formal/xmi.htm` (Acessado em 17/02/2008)

World Wide Web Consortium XQuery 1.0: An XML Query Language. Online, 2007b.
Disponível em `http://www.w3.org/TR/xquery/` (Acessado em 17/02/2008)

Xu, L.; Xu, B.; Jiang, J. Testing Web Applications Focusing on Their Specialties. *SIGSOFT Softw. Eng. Notes*, v. 30, n. 1, p. 10, 2005.

Apêndice

A

Processos e Métodos de Desenvolvimento

Na Tabela A.1 é exibida a comparação entre atributos gerais e classificação dos métodos de desenvolvimento *Web* citados no Capítulo 2.

Tabela A.1: Comparação entre Métodos de Desenvolvimento: Atributos Gerais (Bianchini, 2007)

Método	**Processo**	**Técnica**	**Representação Gráfica**	**Notação**	**Ferramenta**
HDM	1. Esquema HDM 2. Instanciação do esquema HDM 3. Definição da semântica de navegação	E-R	1. Diagrama E-R 2. Diagrama E-R 3. Diagrama E-R	1. E-R 2. E-R 3. E-R	

Tabela A.1: Comparação entre Métodos de Desenvolvimento: Atributos Gerais

Método	**Processo**	**Técnica**	**Representação Gráfica**	**Notação**	**Ferramenta**
RMM	1. Projeto E-R 2. Projeto de fatias 3. Projeto navegacional 4. Projeto de interface com o usuário 5. Projeto de protocolos de conversão 6. Projeto de comportamento em tempo real 7. Construção e teste	E-R	1. Diagrama E-R 2. Diagrama de fatias 3. Diagrama RMDM	1. E-R 2. própria 3. própria	
OOHDM	1. Projeto conceitual 2. Projeto navegacional 3. Projeto de interface abstrata 4. Implementação	OO	1. Diagrama de classes 2. Classes navegacionais e esquema navegacional 3. ADVs e ADV-Charts	1. UML 2. própria 3. própria	OOHDM-Web
HMBS	1. Modelagem conceitual 2. Modelagem navegacional 3. Modelagem de interface 4. Implementação e teste	OO e estados	1. Modelo de objetos e modelo de fatias 2. Contextos de navegação; modelo navegacional de tipos; e de instâncias 3. Canais de apresentação	1. Fusion 2. Statecharts 3. própria	HySCharts e WebSCharts (HMBS/M Estendido)
W2000	1. Análise de requisitos 2. Projeto de evolução de estados 3. Projeto de hipermídia 4. Projeto funcional 5. Projeto de visibilidade	E-R e OO	1. Casos de uso 2. Casos de uso estendido 3. Esquemas e diagrama de classes 4. Cenários e diagramas de interação 5. Visões do sistema	1. UML 2. UML 3. HDM; e UML 4. UML 5. própria	

Tabela A.1: Comparação entre Métodos de Desenvolvimento: Atributos Gerais

Método	**Processo**	**Técnica**	**Representação Gráfica**	**Notação**	**Ferramenta**
UWE	1. Análise de requisitos 2. Modelagem conceitual 3. Modelagem navegacional 4. Modelagem de apresentação	OO	1. Casos de uso 2. Modelo de classes 3. Modelo navegacional de espaço; e de estrutura 4. Diagrama de classes	1. UML 2. UML 3. própria 4. própria	ArgoUWE
WebML	1. Modelagem estrutural 2. Modelagem de hipertexto 3. Modelagem de apresentação 4. Modelagem de personalização	OO	1. Organização conceitual dos dados; e diagrama de classes 2. Modelo de composição; e de navegação 3. Modelo navegacional de espaço; e de estrutura 4. Modelo de entidade pré-definida	1. E-R; e UML 2. própria 3. própria 4. própria	WebRatio
WAE	1. Gerenciamento de projeto 2. Obtenção de requisitos 3. Análise 4. Projeto 5. Implementação 6. Teste 7. Implantação	OO	1. Plano de projeto; de iteração; e de gerenciamento de alteração 2. Documento de declaração de visão; e casos de uso 3. Modelo conceitual; diagramas de seqüência; e de estado 4. Diagramas de classe; de seqüência; e de estado	1. própria 2. própria; e UML 3. UML 4. UML	

Tabela A.1: Comparação entre Métodos de Desenvolvimento: Atributos Gerais

Método	**Processo**	**Técnica**	**Representação Gráfica**	**Notação**	**Ferramenta**
OO-H	1. Modelagem Conceitual 2. Projeto de Navegação 3. Projeto de Interface	OO	1. Diagrama de Classes 2. Diagrama de Acesso Navegacional 3. Diagrama Abstrato de Apresentação	1. UML 2. própria 3. própria	CASE Tool
OOWS	1. Modelagem Conceitual 2. Desenvolvimento da Solução	OO	1. Diagrama de Caso de Uso Cenários 2. Diagrama de Classes 3. Diagrama de Estado Diagrama de Seqüência 4. Diagrama Navagacional (Navigational Map)	1. UML 2. UML 3. UML 4. UML	
SWM	1. Planejamento 2. Análise 3. Projeto 4. Implementação 5. Manutenção	DFD	1. Especificação de projeto conceitual 2. Especificação de projeto do site	1. DFD 2. DFD	ASCENT
ECO	1. Aquisição 2. Entrega 3. Encerramento	OO	1. Mapa de Navegação 2. Modelos de Grade 3. Diagrama de Componentes 4. Modelo de Domínio	1. própria 2. UML 3. UML 4. UML /Domain Neutral Component	

Na Tabela A.2 é apresentada uma comparação considerando etapas de processo de desenvolvimento, propostas por (Pressman, 2002), e o apoio oferecido pelos métodos de desenvolvimento *Web* citados no Capítulo 2.

Tabela A.2: Comparação entre Métodos de Desenvolvimento: Etapas de Processo (Bianchini, 2007)

	Formulação	**Plan.**	**Análise**	**Projeto**			**Geração**	**Teste**	**Avaliação**
				Arquit.	**Naveg.**	**Interf.**			
HDM				✓	✓				
RMM				✓	✓	✓	✓		
OOHDM				✓	✓	✓	✓		
HMBS				✓	✓	✓	✓	✓	
W2000			✓	✓	✓	✓			
UWE			✓	✓	✓	✓			
WebML					✓	✓	✓		
WAE		✓	✓	✓	✓	✓	✓	✓	
OO-H			✓	✓	✓	✓	✓		
OOWS			✓	✓	✓	✓	✓		
SWM	✓	✓	✓	✓	✓	✓	✓	✓	
ECO		✓	✓	✓	✓	✓	✓		

Apêndice

B

Ferramentas de Apoio ao Desenvolvimento

Existem duas possibilidades de busca no repositório de padrões integrado ao RUP, a simples e a avançada. Na Figura B.1 é ilustrada a tela de busca simples, permitindo uma busca geral por palavra-chave.

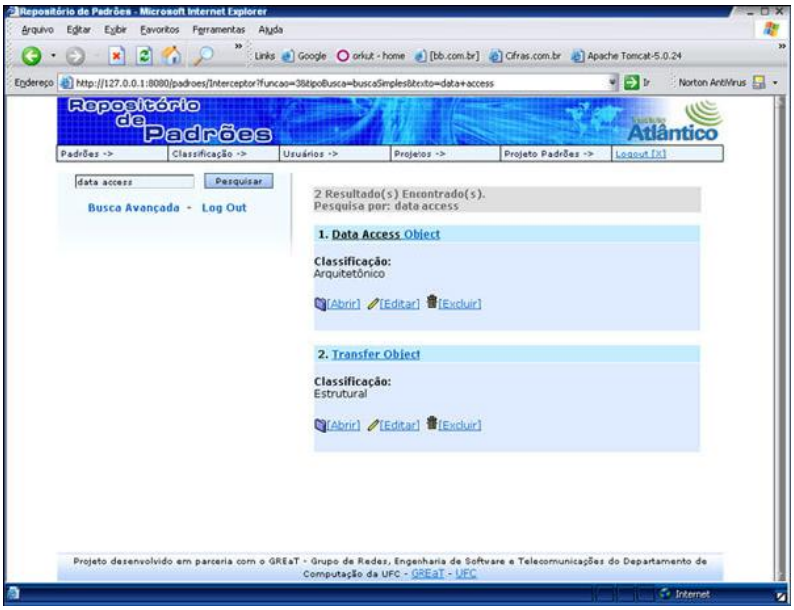

Figura B.1: Tela da busca simples do repositório de padrões integrado ao RUP (Santos, 2004).

Já na Figura B.2 é ilustrada a busca avançada do repositório, permitindo a seleção dos campos onde a palavra-chave deve ser procurada.

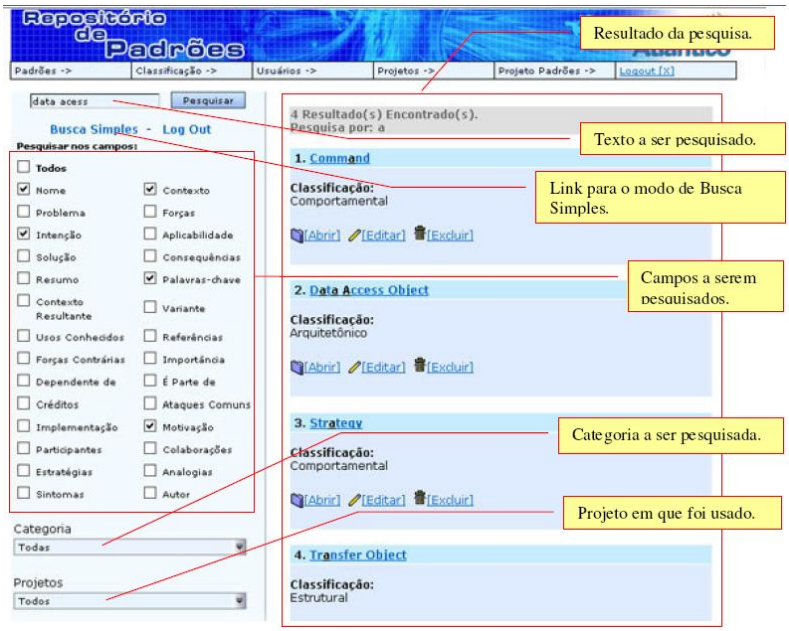

Figura B.2: Tela da busca avançada do repositório de padrões integrado ao RUP (Santos, 2004).

Na Figura B.3 é ilustrada a tela de interface apresentada pelo GREN-Wizard.

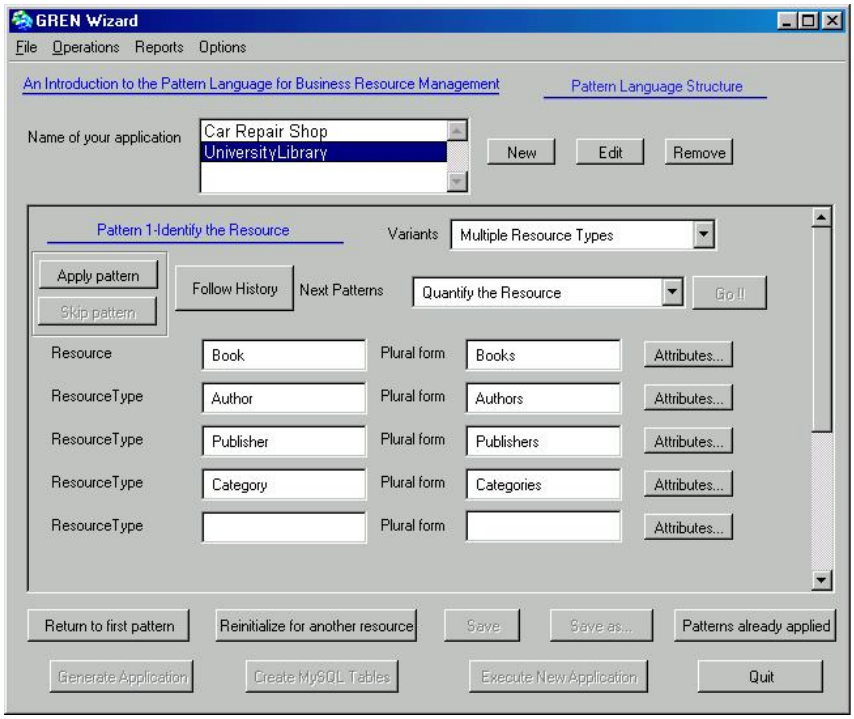

Figura B.3: Tela da interface do GREN-Wizard. (Braga, 2002)

Na Figura B.4 são ilustradas as telas apresentadas ao usuário, onde são executadas as tarefas permitidas pelo FRED. Na Figura B.5 é apresentada a tarefa que cria uma classe de teste para que os usuários possam executar e testar o funcionamento do *framework* especializado pela ferramenta.

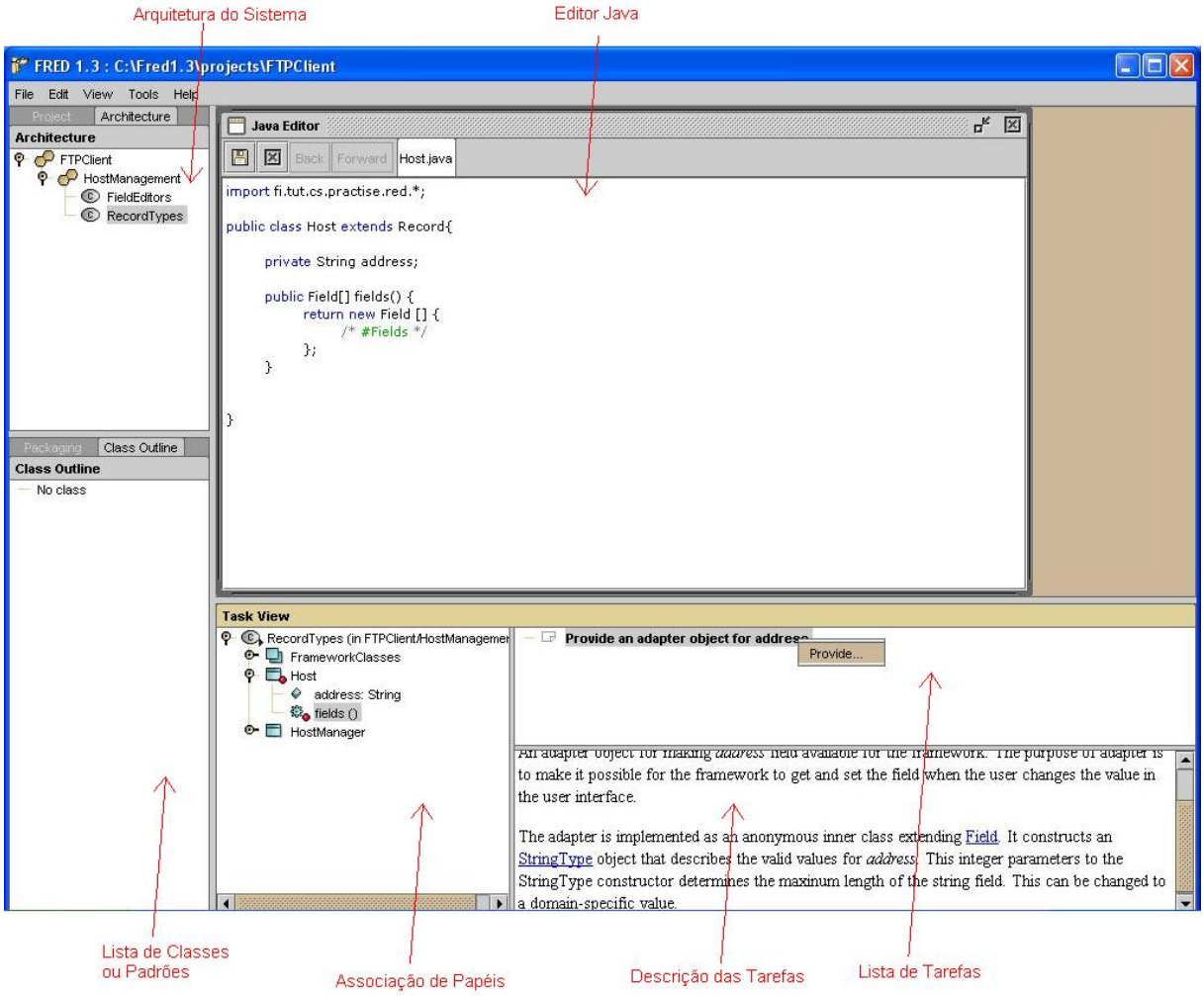

Figura B.4: Exemplo de janelas e funcionalidades do FRED.

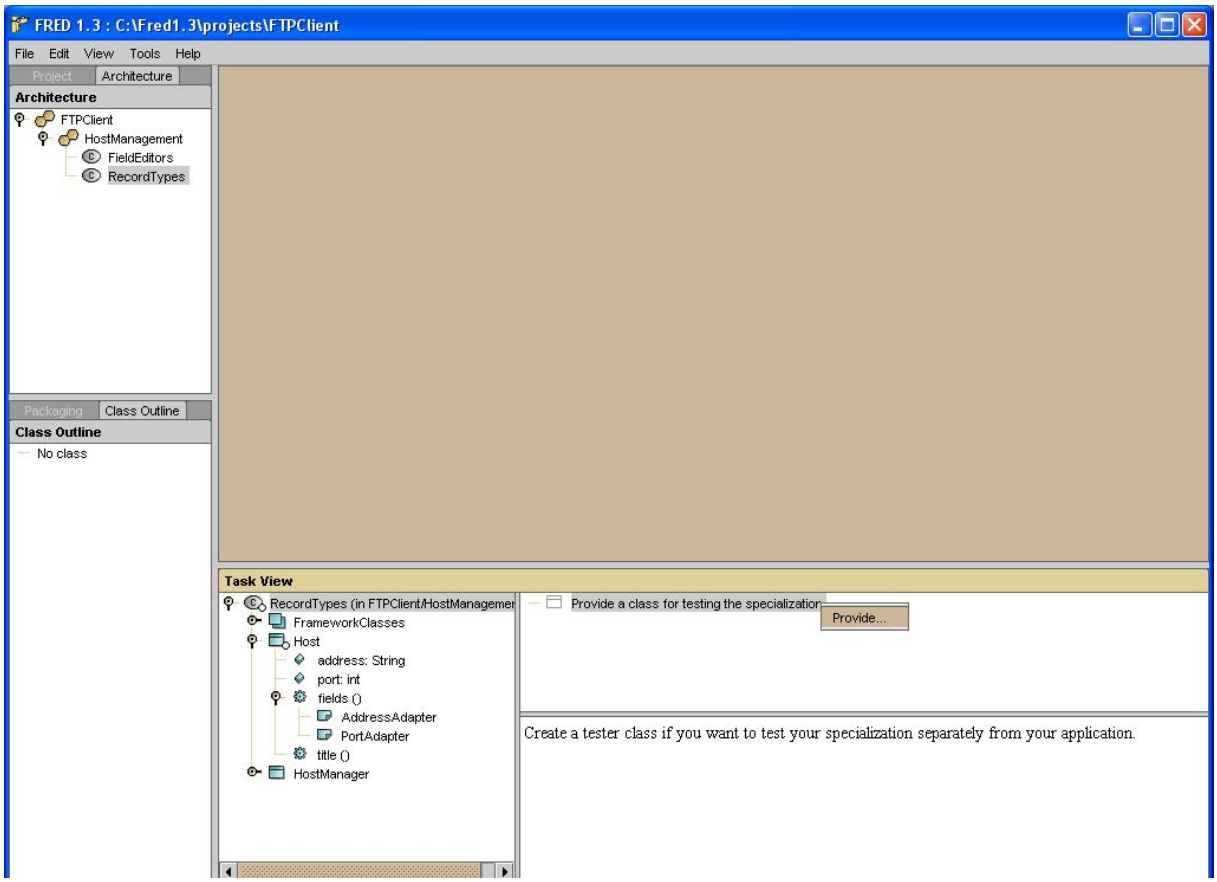

Figura B.5: Classe *Tester*: Seleção da tarefa de criação de uma classe de teste, para permitir a execução da aplicação especializada.

Apêndice

C

Frameworks de Apoio ao Desenvolvimento

Na Figura C.1 é ilustrado um arquivo de mapeamento O/R do Hibernate. Note que é informado para cada coluna da tabela *Usuário* o atributo correspondente na classe *Usuário*. No caso do código identificador, campo *codigo*, o valor é gerado automaticamente utilizando uma seqüência. Na Figura C.2 é ilustrado um arquivo de mapeamento O/R do Hibernate com um exemplo de consulta SQL definida utilizando propriedades XML fornecidas pelo *framework*.

Na Figura C.3 é apresentado um exemplo de descritor XML utilizado pelo iBATIS para realizar a configuração de conexão e mapeamento O/R. É possível observar que as vantagens de consultas complexas em SQL podem ser facilmente mapeadas, pois são inseridas nos arquivos XML igualmente como devem ser realizadas utilizando o SQL nativo do banco de dados, sendo necessário apenas mapear o retorno esperado.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

    <class name="usuario.Usuario" table="Usuario">
        <!-- Informa que a geração de código identificador do usuário é automática. -->
        <id name="codigo" column="codigo" type="java.lang.Long">
            <generator class="native"/>
        </id>

        <!-- Relaciona cada coluna do banco de dados com o atributo da classe. -->
        <property name="nome" column="nome" length="100" not-null="true" type="java.lang.String"/>
        <property name="sobrenome" column="sobrenome" length="100" not-null="true" type="java.lang.String"/>
        <property name="nomeUsuario" column="nomeUsuario" length="50" not-null="true" unique="true"
            type="java.lang.String"/>
        <property name="senha" column="senha" length="50" not-null="true" type="java.lang.String"/>
        <property name="email" column="email" length="100" not-null="true" type="java.lang.String"/>
        <property name="idade" column="idade" length="3" type="java.lang.Integer"/>
    </class>

</hibernate-mapping>
```

Figura C.1: Exemplo de documento de mapeamento XML do Hibernate.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

    <class name="usuario.Usuario" table="Usuario">
        <!-- Informa que a geração de código identificador do usuário é automática. -->
        <id name="codigo" column="codigo" type="java.lang.Long">
            <generator class="native"/>
        </id>

        <!-- Relaciona cada coluna do banco de dados com o atributo da classe. -->
        <property name="nome" column="nome" length="100" not-null="true" type="java.lang.String"/>
        <property name="sobrenome" column="sobrenome" length="100" not-null="true" type="java.lang.String"/>
        <property name="nomeUsuario" column="nomeUsuario" length="50" not-null="true" unique="true"
            type="java.lang.String"/>
        <property name="senha" column="senha" length="50" not-null="true" type="java.lang.String"/>
        <property name="email" column="email" length="100" not-null="true" type="java.lang.String"/>
        <property name="idade" column="idade" length="3" type="java.lang.Integer"/>
    </class>

    <!-- Seleciona um usuário do banco de dados. -->
    <sql-query name="buscarUsuarioPorCodigo">
        <return alias="Usuario" class="usuario.Usuario"/>

        SELECT codigo, nome, sobrenome, nomeUsuario, senha, email, idade
          FROM Usuario WHERE codigo = :codigo
    </sql-query>

</hibernate-mapping>
```

Figura C.2: Exemplo de consultas SQL definidas em documento de mapeamento XML do Hibernate.

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

<!DOCTYPE sqlMap
    PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Usuario">
  <typeAlias alias="Usuario" type="usuario.Usuario"/>

  <!-- Relaciona cada coluna retornada do banco de dados com o atributo da classe. -->
  <resultMap id="UsuarioResultado" class="Usuario">
    <result property="codigo" column="codigo"/>
    <result property="nome" column="nome"/>
    <result property="sobrenome" column="sobrenome"/>
    <result property="nomeUsuario" column="nomeUsuario"/>
    <result property="senha" column="senha"/>
    <result property="email" column="email"/>
    <result property="idade" column="idade"/>
  </resultMap>

  <!-- Seleciona todos os usuários cadastrados no banco de dados. -->
  <select id="buscarUsuario" resultMap="UsuarioResultado">
    SELECT * FROM Usuario
  </select>

  <!-- Seleciona um usuário do banco de dados. -->
  <select id="buscarUsuarioPorCodigo" parameterClass="int" resultClass="Usuario">
    SELECT codigo, nome, sobrenome, nomeUsuario, senha, email, idade
      FROM Usuario WHERE codigo = #codigo#
  </select>

  <!-- Cadastra um novo usuário no banco de dadas -->
  <insert id="inserirUsuario" parameterClass="Usuario">
    INSERT INTO Usuario (codigo, nome, sobrenome, nomeUsuario, senha, email, idade)
      VALUES (#codigo#, #nome#, #sobrenome#, #nomeUsuario#, #senha#, #email#, #idade#)
  </insert>

  <!-- Atualiza dados de um usuário -->
  <update id="alterarUsuario" parameterClass="Usuario">
    UPDATE Usuario SET
        codigo = #codigo#, nome = #nome#, sobrenome = #sobrenome#, nomeUsuario = #nomeUsuario#,
        senha = #senha#, email = #email#, idade = #idade#
      WHERE codigo = #codigo#
  </update>

  <!-- Remove o usuário do banco de dados -->
  <delete id="removerUsuario" parameterClass="int">
    DELETE FROM Usuario WHERE codigo = #codigo#
  </delete>

</sqlMap>
```

Figura C.3: Exemplo de Descritor XML do iBATIS.

Livros Grátis

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

Milhares de Livros para Download:

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