Universidade Federal do Ceará
Departamento de Computação
Mestrado em Ciência da Computação

Dissertação de Mestrado

Uma Proposta para a Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas

Misael da Silva Santos

Fortaleza - CE, 2004

MISAEL DA SILVA SANTOS

Uma Proposta para a Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas

Dissertação apresentada ao Mestrado de Ciência da Computação da Universidade Federal do Ceará (UFC), como requisito parcial para a obtenção do título de Mestre em Ciência da Computação

Orientador:
Rossana Maria de Castro Andrade

Fortaleza - CE, 2004

Agradecimentos

À Deus, por estar sempre presente em cada passo dessa caminhada.

Aos meus pais, Zedequias Ferreira dos Santos e Maria Eliana da Silva Santos por todo o amor, carinho, orgulho e confiança depositada em mim, o que fazia tudo parecer fácil, mesmo quando parecia impossível.

À Simony Fauth, que me acompanhou neste último ano de mestrado com enorme paciência, e incentivo demonstrado, por me inspirar nos momentos difíceis e por sempre me mostrar um grande motivo pelo qual o meu trabalho valeria a pena. “Te amo”.

À professora Rossana Andrade, pela competência demonstrada na orientação deste trabalho e pela confiança depositada no nosso trabalho.

Ao amigo Joaquim, intitulado meu co-có-orientador, por todo o apoio técnico e científico prestado e por sempre saber a hora certa de encher o meu copo. Equipe *brother*!

Aos amigos do Samba Integral, que me proporcionaram grandes momentos de descontração.

Aos amigos do GREaT, em especial à Rute Nogueira, pela colaboração científica nas fases de pesquisa bibliográfica, projeto e testes de implementação.

Aos amigos da Turma 2002 do Mestrado em Ciência da Computação da UFC, que sempre demonstraram companheirismo durante a nossa caminhada.

Ao Instituto Atlântico, pelo apoio oferecido durante o projeto de colaboração e em particular ao Thiago Ferraz, pela importante contribuição na implementação do repositório de padrões do Instituto.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo apoio financeiro.

Resumo

A aplicação de padrões de *software* visa promover a reutilização, de maneira eficiente, de boas práticas da engenharia de *software*. O reuso desses padrões vem se tornando cada vez mais comum no desenvolvimento de sistemas e, com isso, surge a necessidade de utilização de ferramentas visando a aplicação dos mesmos. Entretanto, as ferramentas de desenvolvimento existentes limitam a aplicação dos padrões de *software* disponíveis na literatura. Existe também uma carência de mecanismos que auxiliem no armazenamento e busca desses padrões.

Esta dissertação apresenta uma proposta para a reutilização de padrões de *software* em ferramentas de desenvolvimento, de forma integrada e por demanda, a partir de um formato aberto, o XMI (*XML Metadata Interchange*). Para isso, foi especificado inicialmente um processo para a integração de modelos de padrões de *software* com ferramentas de apoio ao desenvolvimento de sistemas. Em seguida, o Rational Rose, ferramenta de modelagem de dados na linguagem UML (*Unified Modeling Language*), foi escolhido como estudo de caso para a implementação do processo proposto. Além disso, um repositório de padrões de *software* foi desenvolvido propondo uma nova abordagem para o armazenamento e busca de diferentes tipos e formatos de padrões de *software* existentes.

Palavras-chave: Padrões de *Software*, Reutilização de *Software*, Modelos de Padrões, Ferramentas de Apoio ao Desenvolvimento de Sistemas, UML, XMI.

Abstract

The application of software patterns provides reuse of the best practices of software engineering in an efficient way. The reuse of software patterns has increasingly become popular among system developers and, as a consequence, the use of tools for their application has emerged. However, the existing software development tools have limitations for the application of the software patterns available in the literature. There is also lack of mechanisms to help storing and searching for these patterns.

This master thesis presents a proposal for the reuse of software patterns in software development tools in an integrated way and on demand. An open standard format called XMI (*XML Metadata Interchange*) is applied in the proposal. Initially, a process for the integration of software patterns models and software development tools was specified. Then, Rational Rose, a UML (*Unified Modeling Language*) data modeling tool, was chosen as a case study for the implementation of the proposed process. Besides this, a repository for software patterns was developed with a new approach for storing and searching different software pattern categories and templates available.

Keywords: Software Patterns, Software Reuse, Patterns Models, Software Development Tools, UML, XMI.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CASE	Computer Aided Software Engineering
DAO	Data Access Object
DTD	Document Type Definition
GOV	Gang of Five
GOF	Gang of Four
GUI	Graphic User Interface
HTML	Hypertext Markup Language
IIMPAR	Interface de Integração de Modelos de Padrões de Software para o Rose
J2EE	Java 2 Enterprise Edition
JSP	Java Server Pages
MOF	Meta Object Facility
MORAR	Mobility Radio Resources
MVC	Model View Controller
OMG	Object Management Group
OO	Orientado a Objeto
PHP	*Acrônimo recursivo para "PHP: Hypertext Preprocessor"
PLOP	Patterns Languages of Programming
POSA	Pattern-Oriented Software Architecture
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	Structured Query Language
UML	Unified Modeling Language
VO	Value Object
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations
W3C	World Wide Web Consortium

Capítulo 1
Introdução

Este capítulo apresenta na Seção 1.1 uma visão geral sobre reutilização de *software* e a caracterização do problema que conduz à motivação do trabalho. A Seção 1.2 apresenta os objetivos principais da dissertação, e, por fim, a Seção 1.3 apresenta a estrutura na qual está organizado o restante do trabalho.

1.1 Reutilização de Software

Na tentativa de minimizar o tempo gasto e o custo de manutenção, bem como aumentar a qualidade, a confiabilidade e a produtividade no desenvolvimento de sistemas, práticas envolvendo reutilização vêm acompanhando a evolução da engenharia de *software*. Atualmente existem diferentes linhas de estudo que envolvem reutilização de *software*, como: padrões de *software*, construção de *frameworks* [5][44], desenvolvimento baseado em componentes (DBC) [94] e engenharia de domínio [95][96].

O paradigma da orientação a objetos proporcionou muitos avanços para a engenharia de *software*. Os analistas passaram a pensar em objetos, seus relacionamentos e responsabilidades, dando uma importância maior à representação do conhecimento. Dessa forma, muitas idéias e conceitos da análise orientada a objetos contribuíram para a evolução do processo de reutilização. Porém, para tornar efetivo na prática o uso destes conceitos é necessário saber onde, como e quando aplicá-los no desenvolvimento de um sistema.

A necessidade de compartilhar o conhecimento sobre as boas práticas da análise e programação orientadas a objetos impulsionou os estudos sobre padrões de *software* no início da década de 90 [21][22][66][33]. O conceito e o reuso de padrões de *software* vêm cada vez mais sendo difundidos entre analistas e desenvolvedores de sistemas, reafirmando a importância desta área de pesquisa dentro da engenharia de *software*. O Capítulo 2 explora os principais conceitos envolvendo padrões de *software*.

Atualmente, padrões podem ser utilizados individualmente (para resolver problemas isolados ou específicos) ou em conjunto (através do uso de coleções de padrões, para resolver problemas complexos ou para documentar e gerar arquiteturas de *software*). Além disso, padrões podem ser aplicados:

- manualmente – onde o desenvolvedor é responsável por modelar ou codificar o padrão por conta própria, a partir de sua documentação ou através do uso de *frameworks*;
- automaticamente – através do uso de ferramentas de modelagem, *refactoring* ou geração de código com suporte a padrões.

Um ***framework*** pode ser descrito como um elemento para o reuso de componentes de arquiteturas orientadas a objetos. Uma definição mais geral é apresentada em [44]: "*um framework é um projeto abstrato para um tipo de aplicação*". Outras definições de *frameworks* são apresentadas em [5]. Para tentar facilitar o processo de busca e reutilização de padrões de *software* algumas equipes de desenvolvimento constroem *frameworks* que implementam os padrões de *software* mais usados por seus desenvolvedores, padrões que atendam ao seu campo de pesquisa e desenvolvimento específicos, proporcionando assim um fácil acesso às informações e uma referência local do reuso dos padrões.

No **desenvolvimento baseado em componentes (DBC)** [94], a boa utilização de um componente está diretamente relacionada a uma estruturação adequada da arquitetura sobre a qual ele irá atuar. Portanto, existe a necessidade de se utilizar boas práticas para o desenvolvimento dos componentes da arquitetura e seus relacionamentos, o que pode ser feito utilizando padrões de *software*. Padrões podem ser usados também para documentar componentes.

Uma outra linha de estudo na área de reutilização de *software* é a **engenharia de domínio**, definida em [95] como o processo de identificar e organizar o conhecimento sobre uma classe de problemas para suportar sua descrição e solução. A engenharia de domínio tem por finalidade formalizar o conhecimento existente sobre um domínio específico, construir arquiteturas de *software* para atender os requisitos deste domínio e fornecer artefatos reutilizáveis para compor uma arquitetura. Padrões de software podem ser utilizados tanto na construção dessas arquiteturas de *software* quanto no fornecimento de artefatos.

Desta forma, o conceito de padrões de *software* está intimamente ligado às práticas de reutilização de *software* citadas no decorrer desta seção, sendo aplicado em diversos níveis de abstração.

Atualmente, existem ferramentas de apoio ao desenvolvimento de sistemas que automatizam o processo de aplicação de padrões de *software*, auxiliando na modelagem de sistemas, geração e reestruturação de código (exemplos destas ferramentas são apresentados na Seção 4.4). No entanto, os padrões aplicados automaticamente através da utilização dessas ferramentas são geralmente restritos a um conjunto de padrões fornecido pelos fabricantes das

ferramentas, que na maioria das vezes não é extensível, não suportando a inclusão de novos padrões de acordo com as necessidades dos desenvolvedores.

Um outro problema comumente encontrado é o fato dos modelos de padrões serem disponibilizados em formatos proprietários ou formatos específicos para uma determinada ferramenta, o que também limita as opções de escolha e uso dos padrões pelos desenvolvedores.

É importante ressaltar ainda que os mecanismos para armazenamento e busca de padrões de *software* não suportam os diversos tipos (categorias) e formatos de padrões que existem atualmente, o que pode dificultar o processo de busca e tornar inapropriado o armazenamento do padrão, quando o seu formato original não é preservado.

1.2 Objetivos

O objetivo principal desta dissertação é criar uma forma de reutilizar modelos de padrões de *software* em ferramentas de apoio ao desenvolvimento de sistemas de forma integrada e por demanda. Para isso, utilizamos modelos de padrões armazenados no formato XMI (*XML Metadata Interchange*) [60], que é uma especificação XML (*eXtensible Markup Language*) [97] criada como uma tentativa de padronizar a representação de modelos UML (*Unified Modeling Language*) [10][59]. A Figura 1 ilustra a proposta para a reutilização de modelos de padrões de *software.*

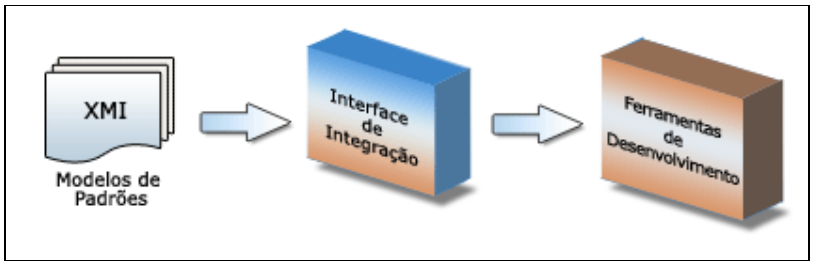

Figura 1 - Proposta para a reutilização de modelos de padrões

Conforme apresentado na Figura 1, uma *Interface de Integração* é responsável por receber modelos de padrões no formato XMI e gerar os componentes de extensão necessários para a inclusão desses modelos em uma ferramenta de apoio ao desenvolvimento de sistemas de forma que possam ser reutilizados sempre que necessário. Para a implementação desta *Interface de Integração* foi definido um ***Processo para a Integração de Modelos de Padrões***

de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas. Como estudo de caso, foi escolhida uma ferramenta de modelagem de dados baseada na linguagem UML, o Rational Rose [68], que tem sido usado com sucesso tanto comercialmente quanto academicamente.

Para atingir o objetivo principal citado anteriormente e suprir a carência de mecanismos que auxiliem no armazenamento e busca dos diferentes tipos de padrões de *software* existentes na literatura, propomos neste trabalho o desenvolvimento de um *Repositório de Padrões de Software* capaz de armazenar e disponibilizar padrões documentados em diferentes formatos e seus modelos na linguagem UML.

A Figura 2 mostra os elementos associados à implementação do processo de integração para o Rose, a IIMPaR (*Interface de Integração de Modelos de Padrões de Software para o Rose*) que é responsável por receber os modelos de padrões de *software* por demanda, gerar e instalar os componentes de extensão do Rose para a aplicação dos padrões de forma automática. O repositório de padrões de *software* citado anteriormente pode fornecer modelos de padrões no formato XMI a fim de integrá-lo ao Rose. Entretanto, a interface de integração é capaz de receber modelos XMI de padrões de qualquer origem.

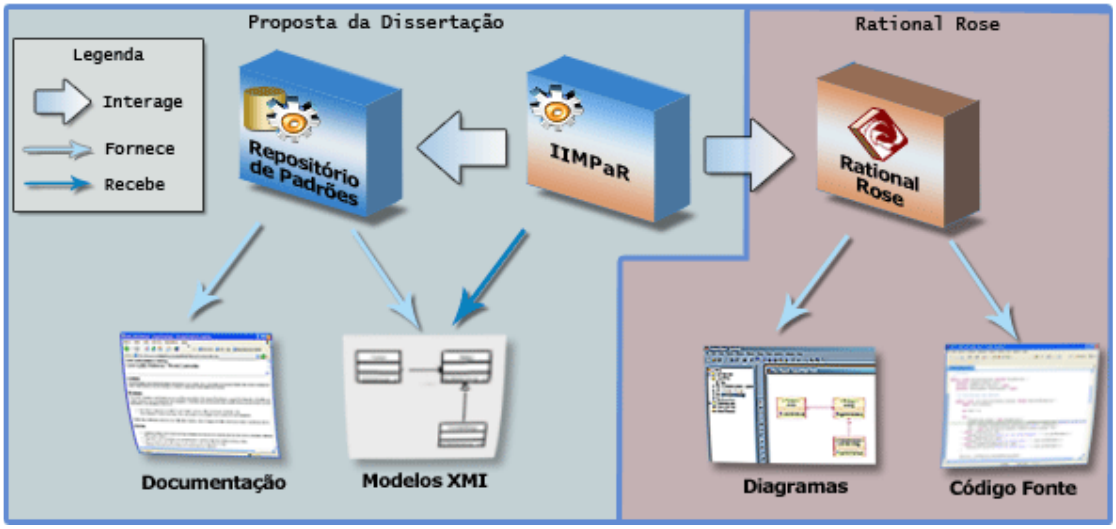

Figura 2 - Proposta de Integração do Repositório de padrões com o Rose

1.3 Organização da Dissertação

Esta dissertação está dividida em mais cinco capítulos e três anexos descritos a seguir:

- *Capítulo 2: Padrões de Software* – Este capítulo apresenta conceitos que envolvem padrões de *software,* aspectos importantes sobre critérios de classificação de

padrões e sobre relacionamentos entre padrões, apresentando ainda uma breve discussão sobre a dificuldade existente na busca e aplicação de padrões de *software,* uma motivação deste trabalho*;*

- *Capítulo 3: Um Repositório de Padrões de Software* – Este capítulo apresenta trabalhos relacionados à abordagem utilizada para a construção do repositório de padrões de *software* proposto, as principais funcionalidades do repositório, aspectos sobre a sua utilização, os recursos usados para o armazenamento e busca de padrões e os componentes de arquitetura do sistema;
- *Capítulo 4: Um Processo Para a Integração de Modelos de Padrões de Software a Ferramentas de Apoio ao Desenvolvimento de Sistemas* – Este capítulo apresenta o processo de integração de modelos de padrões com ferramentas de desenvolvimento, descrevendo as etapas e atividades envolvidas. Além disso, são apresentados os trabalhos relacionados e os conceitos relevantes à abordagem utilizada como: UML, Modelos de Padrões de *Software* e XMI;
- *Capítulo 5: IIMPaR – Uma Interface de Integração de Modelos de Padrões de Software para o Rational Rose* – Este capítulo apresenta o estudo de caso desta dissertação, a interface de integração implementada seguindo o processo proposto, aplicado ao Rational Rose. O Capítulo apresenta também a interface de extensão do Rose (*Rose Extensibility Interface*), que é utilizada para a criação dos recursos gerados pela interface de integração para a aplicação dos padrões no Rose;
- *Capítulo 6: Conclusão* – Este capítulo apresenta os resultados alcançados ao longo do desenvolvimento desta dissertação e as possibilidades para trabalhos futuros;
- *Apêndice A: Esquema XML para o repositório de padrões* – Este apêndice apresenta o esquema XML gerado para representar um padrão no repositório de padrões de *software* proposto;
- *Apêndice B: Arquivo XSL referente à transformação realizada na fase de Simplificação da etapa de Conversão do Processo de Integração* – Este apêndice apresenta o arquivo XSL que é responsável pela transformação realizada durante a fase de *Simplificação* da etapa de *Conversão* do processo de integração proposto no Capítulo 4, que converte os arquivos XMI em arquivos XML simplificados;
- *Apêndice C: Processo de conversão XMI / RoseScripts* – Este apêndice ilustra os passos da etapa de *Conversão* do processo de integração através de um exemplo

onde o modelo XMI do padrão *Strategy* [33] é convertido em um *script* do Rose que é utilizado no Rose para a inserção automática do modelo.

É importante ressaltar algumas particularidades da notação utilizada na escrita desta dissertação que visam facilitar a leitura.

- Os termos em *itálico* são utilizados para:
  - representar palavras ou expressões em língua estrangeira, com exceção de nomes de ferramentas ou produtos;
  - diferenciar um termo definido nesta dissertação do termo correspondente em outro escopo (i.e., "a solução do padrão é descrita no item *solução*", "as atividades da etapa de *Conversão...*");
  - citar definições de outros autores (também delimitadas por "*aspas*");
- Os termos em **negrito** são utilizados para representar palavras ou expressões que estão sendo definidas ou que merecem algum destaque especial no texto. Podem vir acompanhados também de ***itálico*** para um maior destaque;
- Os termos utilizando a fonte `Courier New` são utilizados para representar trechos de códigos.
- Notas de rodapé são utilizadas para fazer algum comentário relevante ou alguma explicação adicional sobre termos utilizados no texto.

Além disso, alguns termos que aparecem na forma de itens ou dotados de marcadores podem receber alguma forma de destaque com **negrito** e/ou *itálico*;

Capítulo 2
Padrões de Software

Neste capítulo são apresentados os conceitos de padrões de *software* mais relevantes para o desenvolvimento desta dissertação. Na Seção 2.1 são apresentadas algumas definições e considerações importantes sobre padrões de *software*. Na Seção 2.2 são identificados os componentes que definem os formatos de padrões. A Seção 2.3 traz uma visão geral sobre linguagens de padrões. Na Seção 2.5 seguem os critérios de classificação de padrões estudados. Em 2.4 são apresentados conceitos sobre relacionamentos entre padrões de *software*, e finalmente a Seção 2.6 traz uma visão geral de como os mecanismos para armazenamento e busca de padrões vêm sendo apresentados na literatura.

2.1 Introdução

Os primeiros conceitos de padrões surgiram na área de arquitetura, com os trabalhos de Christopher Alexander, nas décadas de 60 e 70 [2] [3]. Alexander criou as primeiras definições para os termos *padrão* e *linguagens de padrões,* que são comumente referenciadas em trabalhos de padrões de *software* por serem aplicáveis ao domínio da engenharia de *software*. Vale mencionar as seguintes definições de padrão apresentadas por Alexander:

> *"Cada padrão é uma regra de três partes que expressa uma relação entre um certo contexto, um problema e uma solução"* (Alexander, 1979) [2].

> *"Um padrão é uma entidade que descreve um problema que ocorre repetidamente em um ambiente e então descreve a essência de uma solução para este problema, de tal forma que você possa usar essa solução milhões de vezes, sem nunca utilizá-la do mesmo modo"* (Alexander et al., 1977) [3].

> *"Como um elemento no mundo, cada padrão é uma relação entre um certo contexto, um certo sistema de forças que ocorrem repetidamente naquele contexto, e uma certa configuração espacial que permite que estas forças se solucionem por si só"* (Alexander, 1979) [2].

Assim como na arquitetura e em outras áreas de domínio, a engenharia de *software* possui boas soluções adotadas para a resolução de problemas de análise, projeto e

implementação de *software*, e é onde se aplicam os conceitos de padrões. Por exemplo, James Coplien descreveu um padrão de projeto de *software* da seguinte forma:

> *"Um padrão é um pedaço de literatura que descreve um problema de projeto e uma solução geral para o problema num contexto particular"* (Coplien, 1996) [21].

Considerando as definições apresentadas anteriormente, um padrão é uma entidade que documenta uma solução, um problema recorrente e o contexto em que se deve aplicar tal solução. Para esta dissertação, consideraremos esse conceito de padrões.

É importante salientar que o objetivo da comunidade de padrões de *software* é documentar e compartilhar soluções comprovadas de engenheiros de *software* experientes para problemas recorrentes em um determinado contexto, criando assim uma literatura que auxilie na adoção das boas práticas para o desenvolvimento de sistemas. Neste contexto, Vlissides [90] cita quatro benefícios importantes no reuso de padrões de *software*:

- Capturar experiências tornando-as acessíveis aos não experientes;
- Formar um vocabulário a fim de ajudar desenvolvedores a se comunicarem melhor;
- Ajudar engenheiros de *software* a entender um sistema mais rapidamente quando ele está documentado com os padrões reutilizados;
- Facilitar a reestruturação de um sistema, tendo ele sido ou não projetado com padrões em mente.

Além dos benefícios citados, uma pesquisa realizada em [7] e publicada em [6] apresenta outras vantagens no reuso de padrões de projeto para o desenvolvimento de sistemas, como listada a seguir:

- Evolução de código;
- Modularidade;
- Desacoplamento entre áreas de responsabilidades de forma que as mudanças em uma área não ocasionem mudanças nas outras;
- Diminuição da complexidade do projeto e do código final;
- Facilidade na criação de *frameworks* contendo padrões de projeto já implementados a fim de facilitar o reuso dos padrões posteriormente;
- Estabilidade do código;
- Confiabilidade no reuso de padrões cujas soluções são comprovadas;
- Ganho de produtividade;
- Facilidade de repassar conhecimento entre os desenvolvedores experientes; e

- Facilidade de aprendizado de novas áreas de conhecimento para uma equipe sem experiência na aplicação a ser desenvolvida.

Entretanto, desvantagens no uso de padrões de *software* também podem ser constatadas. Por exemplo, em [6] as seguintes desvantagens foram apontadas em casos particulares do reuso de padrões de projeto:

- *Perda de Eficiência* – O uso de alguns padrões pode tornar um programa mais lento, uma vez que às vezes é preciso adicionar novas classes ou novas camadas de aplicação ao modelo original do sistema, o que pode causar um retardo na sua execução;
- *Diminuição da Legibilidade/Manutenibilidade* – Um padrão pode diminuir a compreensão de um projeto ou de uma implementação. A aplicação de um padrão pode ocasionar em um aumento do número de classes, mensagens, linhas de código, níveis hierárquicos de classes, tornando o código do programa menos legível em virtude do aumento da complexidade do código, levando a um custo maior de manutenção, principalmente para programadores que não tenham algum conhecimento técnico sobre os padrões aplicados;
- *Falta de Motivação da Equipe* – Quando uma equipe de desenvolvimento não tem um entendimento prévio da solução de um padrão, das consequências da sua aplicação e do relacionamento com outros padrões, os membros desta equipe podem se tornar desmotivados quanto à aplicação deste padrão.

2.2 Componentes de um Padrão

Padrões são documentados textualmente e geralmente organizados em seções ou componentes que definem o que chamamos de ***template*** ou **formato** do padrão. Alguns destes componentes são considerados essenciais para um padrão de *software* de acordo com [5] e [54]:

- ***Nome***: descreve um nome que geralmente referencia o problema ou a solução. O nome torna-se rapidamente parte do vocabulário do desenvolvedor;
- ***Contexto***: descreve em que circunstâncias o problema surge. O contexto pode trazer um histórico de que padrões foram aplicados anteriormente;
- ***Problema***: descreve o problema a ser resolvido, expresso por uma simples pergunta ou por uma formulação resumida do problema;

- ***Solução***: descreve claramente o que é necessário para resolver o problema;
- ***Forças***: descreve as considerações positivas e negativas a serem avaliadas a fim de definir por quê (e se) a solução proposta deve ser empregada;
- ***Usos Conhecidos***: descreve exemplos da aplicação do padrão em sistemas reais. Devem constar pelo menos três usos conhecidos (regra de 3) (referenciado em [65]);
- ***Contexto Resultante***: descreve a conclusão do padrão. Define o estado ou configuração do sistema depois da aplicação do padrão, incluindo as consequências boas ou ruins. Em alguns formatos de padrões vem descrito como ***Consequências***;
- ***Padrões Relacionados***: descreve como o padrão está relacionado com outros padrões que se referem ao mesmo problema. Pode referenciar outros padrões usados em conjunto ou outras soluções para o problema, ou ainda, variações do padrão. Em alguns formatos é apresentado como ***Veja Também***.

A seguir são descritos outros componentes de padrões que formam os *templates* de padrões apresentados nesta Seção:

- ***Intenção***: descreve o que o padrão faz, caracterizando o problema e a solução;
- ***Também Conhecido Como***: expõe outros nomes conhecidos para o padrão, se existirem. Em alguns formatos é apresentado como ***Analogias***;
- ***Motivação***: descreve um cenário que ilustra um problema de projeto e como estruturas de classes e objetos no padrão resolvem o problema. O cenário ajudará você a entender a descrição mais abstrata do padrão que segue;
- ***Aplicabilidade***: descreve em que situações o padrão pode ser aplicado. Sugere exemplos de projetos pobres para os quais o padrão pode estar endereçado e indica como você pode reconhecer essas situações.
- ***Estrutura***: descreve a estrutura básica da solução, geralmente através de diagramas de classes e de sequência. Nela também podem ser detalhados os ***Participantes*** e ***Colaborações*****;**
- ***Participantes***: descreve as classes, objetos ou componentes participantes do padrão e suas responsabilidades;
- ***Colaborações***: descreve como os participantes colaboram entre si para cuidarem de suas responsabilidades;
- ***Argumentação (Rationale):*** descreve o porquê de esta solução ser a mais apropriada para o problema dentro deste contexto;

- ***Implementação***: descreve técnicas, dicas ou questões específicas de linguagem que você precisa saber para implementar o padrão;
- ***Exemplos***: descreve exemplos concretos de como aplicar o padrão. Pode apresentar códigos-fonte, diagramas, figuras, etc;
- ***Estratégias***: descreve as diferentes maneiras com que um padrão pode ser implementado;
- ***Sketch***: descreve, através de desenhos ou diagramas, como o padrão funciona ou como as partes de relacionam;
- ***Dinâmica***: descreve os cenários típicos do comportamento do padrão em tempo de execução;
- ***Exemplo Resolvido***: descreve aspectos importantes para a resolução do exemplo que ainda não foram cobertos pelas seções *solução*, *estrutura*, *dinâmica* ou *implementação.*
- ***Resumo*****:** descreve o padrão de forma breve**.** Em alguns formatos é apresentado como ***Sinopse.***

Estes componentes podem aparecer com nomes diferentes (i.e., *Analogias* e *Também Conhecido Como;* e *Padrões Relacionados* e *Veja Também*), ou ordem de apresentação diferente, em formatos diferentes de padrões.

Na Tabela 1 apresentamos os formatos de padrões da *Gang of Four* [33], do catálogo Core J2EE [24], Coplien [21] e POSA [14].

GoF	J2EE	Coplien	POSA
❑ *Nome*	❑ *Nome*	❑ *Nome*	❑ *Nome*
❑ *Classificação*	❑ *Problema*	❑ *Contexto*	❑ *Também Conhecido Como*
❑ *Intenção*	❑ *Forças*	❑ *Problema*	❑ *Exemplo*
❑ *Também Conhecido Como*	❑ *Solução*	❑ *Forças*	❑ *Contexto*
❑ *Motivação*	○ *Estrutura*	❑ *Solução*	❑ *Problema*
❑ *Aplicabilidade*	○ *Estratégias*	❑ *Sketch*	❑ *Solução*
❑ *Estrutura*	❑ *Consequências*	❑ *Contexto Resultante*	❑ *Estrutura*
❑ *Participantes*	❑ *Código Exemplo*	❑ *Argumentação (Rationale)*	❑ *Dinâmica*
❑ *Colaborações*	❑ *Padrões Relacionados*		❑ *Implementação*
❑ *Implementação*			❑ *Exemplo Resolvido*
❑ *Código Exemplo*			❑ *Variantes*
❑ *Consequências*			❑ *Usos Conhecidos*
❑ *Usos Conhecidos*			❑ *Consequências*

❑ *Padrões Relacionados*			❑ *Veja Também*

Tabela 1 - Formatos de Padrões

Os formatos de padrões apresentados na Tabela 1 foram estudados durante o desenvolvimento deste trabalho, dos quais foram retirados os componentes para a elaboração da estrutura que utilizamos para representar os padrões de *software* para esta dissertação (veja Capítulo 3).

As seções utilizadas no formato de Alexander não são fortemente delimitadas sintaticamente. Por exemplo, a única estrutura sintática utilizada é o termo "*Therefore"* que precede a *Solução* [5][21]. Existem elementos e particularidades que são características fortes dos padrões de Alexander, como o fato de sempre existir uma figura no início e um parágrafo que introduz o contexto. Os demais formatos de padrões vêm geralmente com as seções estruturadas em itens.

2.3 Linguagens de Padrões

Como dito anteriormente, o conceito de linguagens de padrões também foi introduzido por Alexander. Em uma adaptação para a engenharia de *software,* Coplien descreveu uma linguagem de padrões como:

> *"Uma coleção de padrões que trabalham em conjunto para construir um sistema"*, (Coplien, 1996) [21].

Padrões isolados são úteis, mas eles são mais poderosos quando combinados em uma Linguagem de Padrões.

As linguagens de padrões, além dos padrões que as compõem, possuem um título, geralmente possuem uma descrição ou resumo e um mapa, que consiste de um grafo que ilustra como seus padrões estão relacionados. Algumas considerações importantes sobre estes relacionamentos entre padrões são apresentadas na Seção 2.4.

A Figura 3 mostra um exemplo de um mapa da linguagem de padrões MoRaR [5], que consiste em uma Linguagem de Padrões para o Gerenciamento de Mobilidade e de Recursos de Rádio. Cada padrão da linguagem está associado a uma categoria que representa uma função de Gerenciamento de Mobilidade ou uma função de Gerenciamento de Recursos de Rádio, usadas em diferentes cenários relacionados a sistemas móveis. Além disso, os padrões da linguagem são divididos em Estruturais (retângulos) e Comportamentais (ovais).

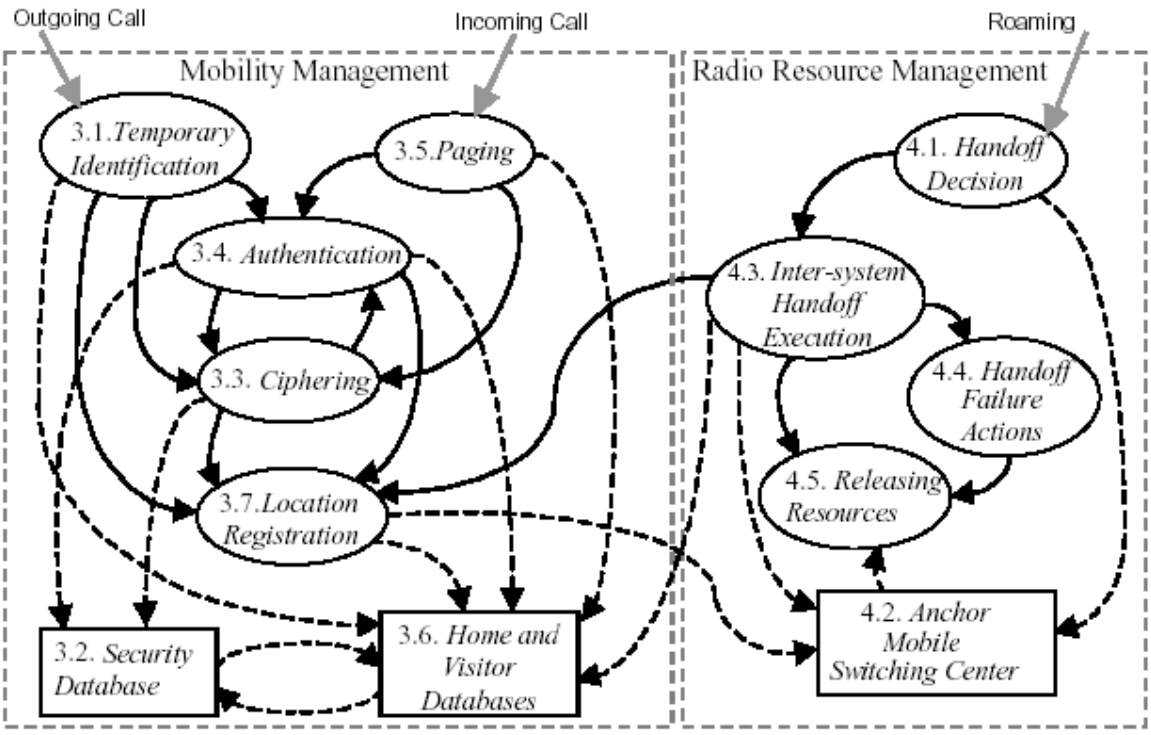

Figura 3 - MoRaR - Uma Linguagem de Padrões para o Gerenciamento de Mobilidade e de Recursos de Rádio [5]

Os padrões em uma *linguagem de padrões* trabalham em conjunto para solucionar um problema em comum. Cada padrão cria o contexto para o próximo padrão a ser aplicado, e dessa forma os padrões de uma linguagem se completam. Neste sentido, segundo [35], uma *linguagem de padrões* deve ser **completa morfologicamente** (seus padrões se combinam para formar uma estrutura sem lacunas) e **completa funcionalmente** (quaisquer novas forças introduzidas pelos padrões são resolvidas por eles mesmos). Se os padrões não são completos dessas duas maneiras, então eles são considerados uma simples ***coleção de padrões*** ou ***catálogo de padrões***, como [33][14][24].

Outro ponto também discutido em [35] é o fato de um mapa de uma linguagem de padrões sempre representar, ou não, um DAG (Grafo Acíclico Direcionado), onde a direção das arestas do grafo indicam a ordem na qual os padrões devem ser aplicados.

O termo ***sistema de padrões*** é usado para coleções de padrões de características semelhantes às *linguagens de padrões*. Buschmann et al definem em [14] que um *sistema de padrões* não precisa ser computacionalmente completo (morfologicamente e funcionalmente) como uma *linguagem de padrões*.

2.4 Relacionamentos entre Padrões

Padrões de *software* podem estar relacionados uns com os outros, sejam eles padrões pertencentes a uma mesma coleção, ou padrões de origens distintas. Diferentes soluções para um mesmo problema podem gerar padrões diferentes. Padrões podem ainda representar uma evolução de outros padrões já existentes. É comum também encontrarmos padrões que especializam ou generalizam soluções de outros padrões, ou ainda, padrões podem trabalhar em conjunto, como visto na Seção 2.3, em linguagens ou catálogos de padrões.

Estes são exemplos que descrevem alguns dos tipos de relacionamentos existentes entre padrões de *software*. Muitos *templates* de padrões (ver Seção 2.2) trazem como um de seus componentes a seção de *Padrões Relacionados,* onde os autores descrevem de forma textual os padrões relacionados e o tipo de relacionamento entre eles.

Algumas linguagens e catálogos de padrões possuem representações gráficas dos relacionamentos entre seus padrões, são os chamados mapas, como mencionado na seção anterior. Um exemplo destas representações gráficas pode ser visto na Figura 3, que traz o mapa da linguagem de padrões MoRaR [5]. A Figura 4, traz os relacionamentos entre os padrões de projeto da *Gang of Four* [33]. Em particular, este último diagrama não traz claramente os tipos de relacionamentos, mas sim uma descrição de como os padrões podem ser usados em conjunto.

Zimmer [101] organiza os relacionamentos entre os padrões de projeto da *Gang of Four* em categorias a fim de facilitar o entendimento da estrutura completa do catálogo. Os relacionamentos entre um par (X,Y) de padrões foram divididos em três tipos: *X **usa** Y na sua solução*, *X **é semelhante a** Y* (is similar to) e *X **pode ser combinado com** Y.*

Em [20], são formalizados quatro tipos de relacionamentos possíveis entre padrões.

- Se um padrão P_1 ***usa*** um padrão P_2, então a solução do padrão P_1 deve ser expressa usando P_2.
- Se um padrão P_1 ***refina*** um padrão P_2, então o problema do padrão P_1 deve ser uma especialização do padrão P_2.
- Se um padrão P_1 ***requer*** um padrão P_2, então aplicação do padrão P_2 é exigida na aplicação de P_1.
- Se um padrão P_1 é ***alternativo*** a um padrão P_2, então os padrões P_1 e P_2. fornecem soluções diferentes para o mesmo problema (o relacionamento ***alternativo*** é apresentado em [34] como o relacionamento ***É Conflitante (Conflicts)***).

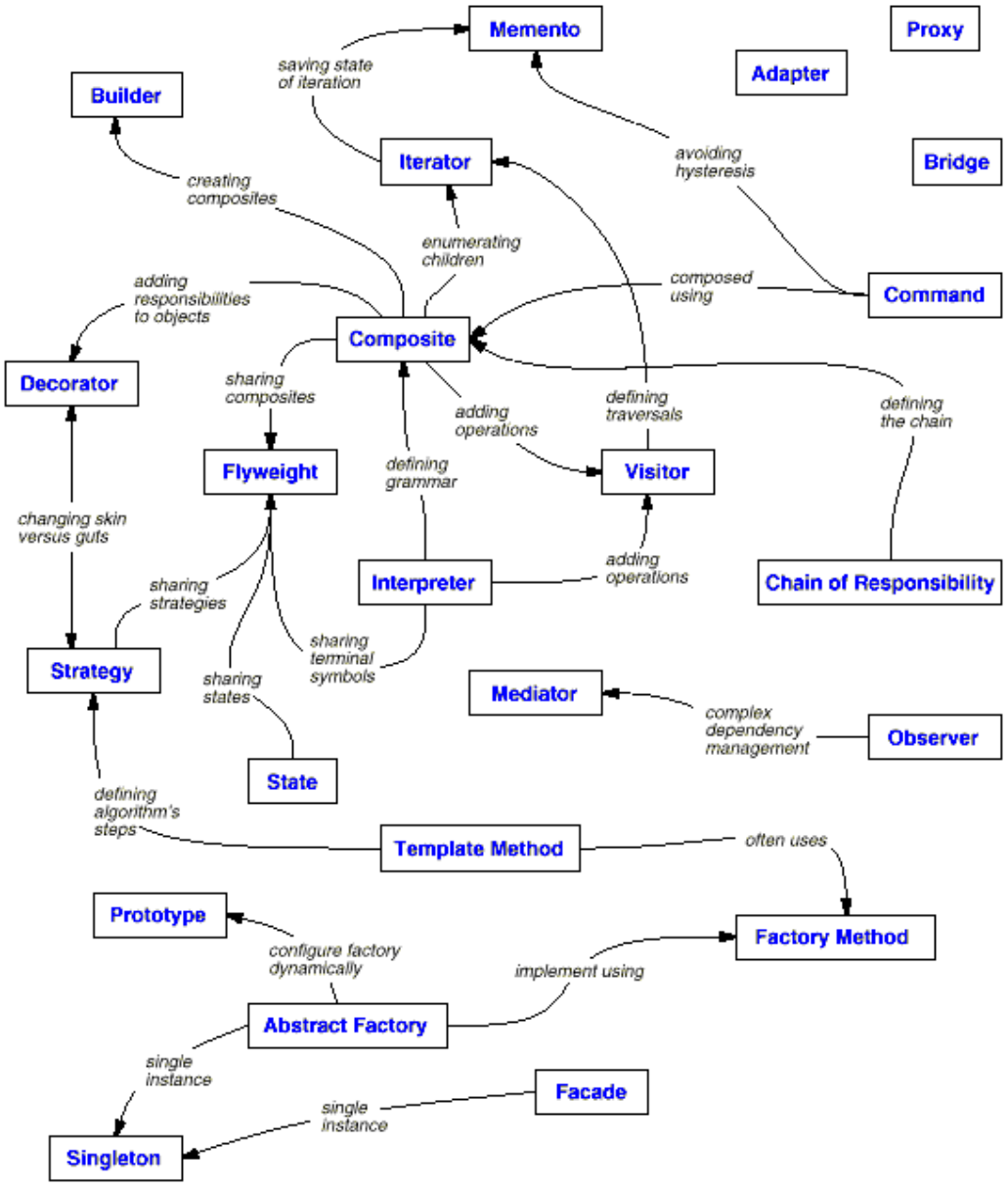

Figura 4 - Relacionamentos entre os padrões da Gang of Four [33]

Estas formalizações de relacionamentos de padrões são consideradas atualmente para o nosso trabalho. Mesmo assim, o repositório de padrões (Capítulo 3) suporta a inclusão de novos tipos de relacionamentos, pois o *tipo de relacionamento*, assim como o *padrão*, é uma entidade de dados que é mantida pelo repositório.

2.5 Classificação de Padrões

Nesta Seção iremos apresentar uma visão geral sobre classificação de padrões de *software*. Não pretendemos aqui citar exaustivamente todos os critérios de classificação existentes, mas sim os critérios mais utilizados na literatura.

Antes de falarmos de classificação é importante citar aqui o uso dos termos ***classificação*** e ***categoria***. Para esta dissertação, o termo *classificação* corresponde ao "ato ou processo de classificar" ou "a distribuição por classes". O termo *categoria* é empregado quando falamos de uma classe particular, cujo conjunto define uma *classificação*. Em resumo, o conjunto de *categorias* às quais um padrão pertence define sua *classificação*. Desta forma, para estudarmos uma determinada *classificação*, precisamos estudar as *categorias* que a definem. A mesma terminologia é utilizada em POSA [14], onde as *categorias de padrões* e *categorias de problemas* são os critérios que definem o seu esquema de classificação.

2.5.1 Classificação segundo o estágio de desenvolvimento do software

Segundo Andrade [5], uma classificação bastante usada para padrões de *software* é a que classifica os padrões de acordo com o estágio de desenvolvimento de *software* em que o padrão é aplicado. Desta forma, os padrões podem pertencer às seguintes categorias: padrões de **requisitos**, padrões de **análise**, padrões de **projeto** e padrões de **implementação** (também chamados de **Idiomas**) e padrões de **testes**, como ilustrado na Figura 5. Padrões *arquitetônicos*[1] e *meta-padrões* também são considerados padrões de *projeto*, apesar de constituírem por si só duas categorias de padrões.

Padrões **arquitetônicos** são os padrões que ajudam a especificar elementos de arquiteturas de sistemas e seus relacionamentos. Uma definição mais completa é apresentada na Seção 2.5.5.

Meta-padrões [66] são padrões de *software* utilizados para descrever como construir *frameworks* independente de um domínio específico. Os meta-padrões trabalham em um nível mais alto de abstração que os padrões de projeto, e podem especificar como as soluções adotadas por padrões de projeto se adequam a situações mais gerais através da observação dos seus aspectos comuns.

[1] O termo "Arquitetônico" é usado nesta dissertação como uma tradução para "*Architectural*", apesar de muitos usarem o termo "Arquitetural", o termo correspondente para a língua portuguesa é "Arquitetônico".

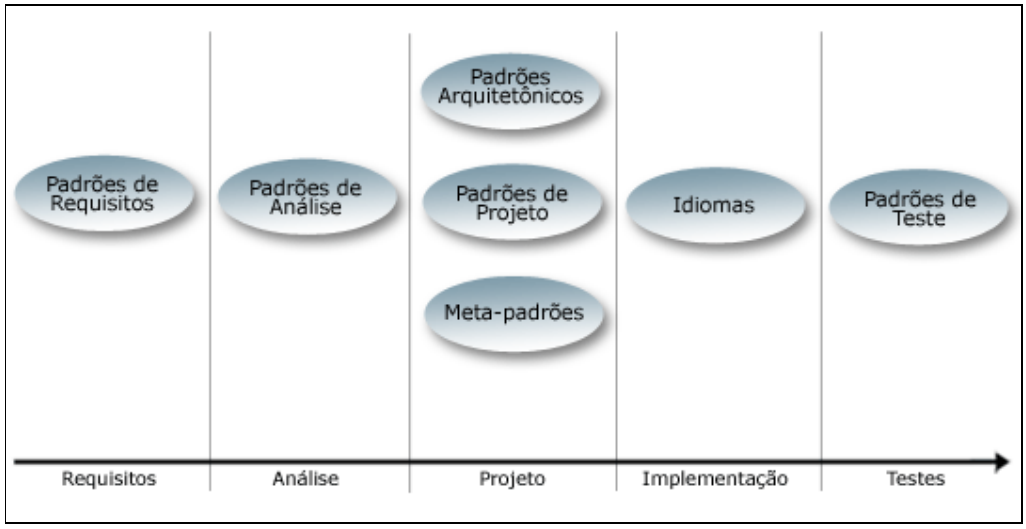

Figura 5 - Classificação pelo estágio de desenvolvimento (traduzido e adaptado de [5])

Os primeiros tipos de padrões de *software* publicados foram *padrões de projeto (Design Patterns)* [14][33] e *Idiomas* [22][9], com o tempo foram surgindo padrões voltados para as fases inicias de desenvolvimento, padrões de níveis mais altos de abstração, como os *padrões de análise*, com os trabalhos de Fowler [30][32], *padrões de requisitos* [73][47] e *padrões de testes* [71].

2.5.2 Classificação segundo o domínio da aplicação

Outro critério muito utilizado é a classificação dos padrões pelo seu domínio de aplicação específico. Atualmente, desenvolvedores com pouca experiência em padrões ou que não estão familiarizados com os tipos de padrões existentes, buscam padrões pela natureza da aplicação. Dessa forma, é comum encontramos padrões na literatura referenciados em categorias como: padrões para *Web*, *GUI*, *XML*, Segurança de Sistemas, Redes, Banco de Dados, Sistemas Distribuídos, entre outros.

Alguns sites se especializaram em disponibilizar padrões por áreas específicas de pesquisa. É o caso do *SecurityPatterns.org* [82], que traz vários padrões sobre de segurança de sistemas, o *XMLPatterns.com* [91], que traz um catálogo de padrões para XML, o *phpPatterns* [64] que possui referências de trabalhos sobre padrões para desenvolvimento em PHP, e o *Amsterdan Patterns Collection* [93], que traz uma coleção de padrões classificados como padrões de projeto para *Web*, *GUI* e *MobileUI*.

O Almanaque de Padrões 2000 [71] possui um índice de padrões por categorias, que consistem tanto de categorias que classificam os padrões segundo o domínio da aplicação, segundo o estágio de desenvolvimento da aplicação do padrão, quanto às categorias utilizadas pela *Gang of Four* [33], que são apresentadas na próxima Seção 2.5.4.

Existem ainda algumas categorias de padrões que estão relacionados a áreas que dão suporte ao desenvolvimento de sistemas. São exemplos dessas categorias: *Gerenciamento de Configuração*, *Organizacional*, *Processo, Modelagem de Sistemas, Treinamento* ([23][4][71]), entre outras. Coplien apresentou em [23] o primeiro trabalho sobre padrões de processo, *"A Development Process Generative Pattern Language"*. Tornou-se claro que a idéia por trás de padrões de *software* poderia trazer benefícios não só às áreas ligadas diretamente a projeto e implementação, mas também às áreas correspondentes a outras disciplinas que dão suporte ao desenvolvimento de sistemas.

2.5.3 Classificação segundo a camada de aplicação do padrão

Um outro critério de classificação utiliza como base a camada de aplicação onde o padrão deve ser aplicado. O catálogo de padrões, *Core J2EE* [24] da *Sun Microsystems* traz um conjunto de padrões dividido em três categorias referentes às camadas de aplicação: *Apresentação*, *Negócios* e *Integração*.

A camada de ***Apresentação*** encapsula toda a lógica relacionada com a apresentação dos dados. A camada de ***Integração*** encapsula a lógica relacionada com a integração do sistema com a camada de informação. A camada de ***Negócios*** encapsula a lógica central da aplicação. A organização dos padrões do catálogo nestas três categorias é apresentada na Tabela 2

Camadas de Aplicação		
Apresentação	**Negócios**	**Integração**
• **Decorating Filter** • **Front Controller** • **View Helper** • **Composite View** • **Service to Worker** • **Dispatcher View**	• **Business Delegate** • **Value Object** • **Session Facade** • **Aggregate Entity** • **Value Object Assembler** • **Value List Handler** • **Service Locator**	• **Data Access Object** • **Service Activator**

Tabela 2 - Catálogo de Padrões J2EE [24]

O catálogo de padrões *IBM patterns for e-business* [38] também utiliza esse critério para a classificação de padrões.

2.5.4 Classificação da *GoF*

Os autores da ***Gang of Four*** (como ficaram conhecidos os autores de [33]) , ou simplesmente *GoF,* propõem um sistema de classificação baseado em dois critérios: quanto ao ***escopo*** e quanto ao ***propósito*** do padrão. O critério *escopo* especifica se o padrão trabalha com foco em classes, subclasses e seus relacionamentos (escopo ***classe***), ou se o padrão trabalha com relacionamentos entre objetos (escopo ***objeto***). O critério *propósito* reflete o papel do padrão. Segundo este critério os padrões são classificados em:

- **De Criação (*Creational*)**: padrões que abrangem a configuração e o processo de criação de objetos ou classes;
- **Estruturais (*Structural*)**: padrões que trabalham com a composição de classes ou objetos;
- **Comportamentais (*Behavioral*)**: padrões que definem o modo com que grupos de classes ou objetos interagem entre si e distribuem responsabilidades.

A seguir é apresentada a organização do catálogo de padrões *GoF* na Tabela 3.

		Propósito		
		Criação	**Estrutural**	**Comportamental**
Escopo	***Classe***	• **Factory Method**	• **Adapter (classe)**	• **Interpreter** • **Template method**
	Objeto	• **Abstract Factory** • **Builder** • **Prototype** • **Singleton**	• **Adapter (objeto)** • **Bridge** • **Composite** • **Decorator** • **Façade** • **Flyweight** • **Proxy**	• **Chain of Responsibility** • **Command** • **Iterator** • **Mediator** • **Memento** • **Observer** • **State** • **Strategy** • **Visitor**

Tabela 3 - Classificação da *Gang of Four*

Estas categorias tornaram-se populares, juntamente com os padrões da *GoF*, e são frequentemente utilizadas na classificação de padrões de projeto por diversos autores, como Linda Rising em [71].

2.5.5 Classificação POSA

Em [14] foi publicado um catálogo de padrões para arquitetura de *software* (*POSA – Pattern-Oriented Software Architecture*). Assim como a *GoF*, os autores deste catálogo são comumente referenciados como *GoV* (***G**ang **o**f fi**V**e*). O livro agrupa os padrões em três **categorias de padrões** e dez **categorias de problemas**, utilizando estes dois critérios para o esquema de classificação. A seguir são apresentadas as definições das categorias utilizadas pelos autores:

Categorias de Padrões:

- **Padrões Arquitetônicos:** padrões que expressam um esquema de organização estrutural fundamental para sistemas de *software*. Fornecem um conjunto de subsistemas predefinidos, especifica suas responsabilidades e incluem regras e diretrizes para organizar os relacionamentos entre eles.
- **Padrões de Projeto:** padrões que fornecem um esquema para refinar subsistemas, componentes de um sistema de *software* ou os relacionamentos entre eles. Descrevem uma estrutura comumente recorrente de componentes comunicantes que resolvem um problema geral de projeto em um contexto particular.
- **Idiomas:** padrões de baixo nível, específicos para uma linguagem de programação. Um idioma descreve como implementar aspectos particulares de componentes ou os relacionamentos entre eles usando as características da linguagem.

Categorias de Problemas:

- **Da Lama à Estrutura:** padrões que suportam uma decomposição adequada de um sistema de tarefas em subtarefas cooperativas.
- **Sistemas Distribuídos:** padrões que fornecem infra-estrutura para sistemas que têm componentes localizados em diferentes processos ou em vários subsistemas e componentes.
- **Sistemas Interativos:** padrões que ajudam a estruturar sistemas com interação homem-computador.
- **Sistemas Adaptáveis:** padrões que fornecem infra-estruturas para a extensão e adaptação de aplicações em resposta a complicações e mudanças nos requisitos funcionais.

- **Decomposição Estrutural:** padrões que suportam uma decomposição adequada de subsistemas e componentes complexos em parte cooperativas.
- **Organização de Trabalho:** padrões que definem como componentes colaboram entre si para fornecer um serviço complexo.
- **Controle de Acesso:** padrões que guardam e controlam o acesso a serviços ou componentes.
- **Gerenciamento:** padrões para o tratamento de coleções de objetos, serviços e componentes homogêneos em sua plenitude.
- **Comunicação:** padrões que ajudam a organizar a comunicação entre componentes.
- **Tratamento de Recursos:** padrões que ajudam a gerenciar componentes e objetos compartilhados.

A organização do catálogo de padrões POSA é apresentada na Tabela 4.

		Categorias de Padrões		
		Padrões Arquitetônicos	**Padrões de Projeto**	**Idiomas**
Categorias de Problemas	***Da Lama à Estrutura***	• **Layers** • **Pipes and Filters*** • **Blackboard**		
	Sistemas Distribuídos	• **Broker** • **Pipes and Filters*** • **Microkernel***		
	Sistemas Interativos	• **Model-View-Controller (MVC)** • **Presentation-Abstraction-Control (PAC)**		
	Sistemas Adaptáveis	• **Microkernel*** • **Reflection**		
	Decomposição Estrutural		• **Whole-Part**	
	Organização de Trabalho		• **Master-Slave**	
	Controle de Acesso		• **Proxy**	
	Gerenciamento		• **Command Processor** • **View Handler**	
	Comunicação		• **Forwarder-Receiver** • **Client-Dispatcher-Server** • **Publisher-Subscriber**	
	Tratamento de Recursos			• **Counted Pointer**

Tabela 4 - Classificação POSA

*Padrões que pertencem a mais de uma categoria de problemas.

2.6 Dificuldade na busca e aplicação de padrões de software

Uma grande quantidade de padrões de *software* está disponível em *websites* e na literatura, prontos para serem reutilizados por engenheiros de *software* nas diferentes fases de desenvolvimento de sistemas.

Os padrões de projeto da *Gang of Four* (GoF) [33], o catálogo de padrões *Core J2EE* da Sun [24], os padrões de análise propostos por Fowler [31] e os padrões voltados para a arquitetura de software (POSA) [14] são exemplos de padrões disponíveis na literatura. Além disso, alguns sites já tradicionais nesta área trazem uma boa quantidade de padrões catalogados. O Grupo Hillside [88], que é uma organização sem fins lucrativos que busca disseminar a área de padrões de *software*, mantém o site oficial de padrões que disponibiliza um sistema de meta-busca para padrões. Esse sistema traz como resultados: endereços de *websites,* referências para livros, e muitas vezes os *sites* dos próprios autores dos padrões procurados.

Uma tentativa de catalogar os padrões publicados em conferências PLoP (*Pattern Languages of Programming*) [63] foi apresentada em [71], onde os padrões ficam organizados por diversas categorias (ver Seção 2.5).

Como visto na Seção 2.5.2, existem também sites especializados em catalogar padrões de domínios de aplicação específicos, o que é útil para pesquisadores e desenvolvedores que buscam soluções para estes domínios específicos em particular.

Com todas essas fontes de pesquisa fica a critério do desenvolvedor o modo como irá executar a busca, muitas vezes exaustiva, por padrões específicos que atendam às suas necessidades. Além disso, é difícil encontrar um modelo de processo que auxilie o desenvolvedor na busca e aplicação de padrões durante o desenvolvimento de sistemas.

2.7 Conclusão

Padrões de *software* têm sido objeto de estudo, aprendizado e de reutilização de conhecimento para pesquisadores e profissionais das mais diversas áreas da computação como apresentado neste capítulo. *Linguagens de padrões*, *catálogos de padrões*, *Design Patterns* são termos que ganharam espaço na literatura de *software* e já fazem parte do vocabulário de engenheiros de *software*.

Como visto nas seções anteriores, um dos problemas existentes devido a grande diversidade de padrões de *software* existentes é a ausência de um mecanismo ou metodologia que auxilie no armazenamento, busca e aplicação destes padrões. No Capítulo 3 é apresentada uma proposta de um *Repositório de Padrões de Software* desenvolvida ao longo desta dissertação a fim de solucionar alguns dos problemas apresentados anteriormente, como representação, busca e armazenamento de padrões. No Capítulo 4 é apresentada uma proposta para a reutilização de padrões de *software* através da definição de um processo de integração entre ferramentas de desenvolvimento e modelos de padrões de *software*.

Capítulo 3
Um Repositório de Padrões de Software

Neste capítulo, um repositório de padrões de *software* é proposto para catalogar os padrões que servem de insumo para as diferentes etapas do processo de desenvolvimento de sistemas. Na Seção 3.1 é introduzido o contexto no qual se aplica a proposta do repositório. Na Seção 3.2 são apresentados trabalhos relacionados com o repositório de padrões e considerações importantes sobre os fatores que diferenciam a implementação do repositório proposto de soluções existentes. As principais funcionalidades do repositório são descritas na Seção 3.3. A Seção 3.4 mostra a utilização do repositório através da descrição dos papéis dos usuários do sistema. Na Seção 3.5 são apresentados os mecanismos de busca implementados para o repositório. A estrutura e a abordagem utilizada para representar e armazenar os dados no repositório são apresentados na Seção 3.6. A Seção 3.7 traz uma visão geral das tecnologias utilizadas no projeto e implementação do repositório. E, finalmente, a Seção 3.8 apresenta um conclusão do capítulo.

3.1 Introdução

Conforme apresentado no Capítulo 1, reutilização de *software* consiste em algo mais do que armazenar e compartilhar código-fonte de programas. Escrever e documentar padrões de *software* é uma forma eficaz de reutilizar conhecimento sobre o desenvolvimento de sistemas. Para que a reutilização de padrões seja realizada de forma eficiente são necessários mecanismos ou ferramentas que auxiliem na automatização do processo de reuso.

Com o crescimento acelerado do número de padrões de *software* na literatura e do seu reuso no desenvolvimento de sistemas, cresce também a necessidade do aprimoramento destes mecanismos para o suporte ao armazenamento, busca e aplicação desses padrões.

Padrões de *software,* como artefatos de reutilização, precisam ser armazenados em um formato apropriado e estar disponíveis de forma prática para o reuso. É importante facilitar o processo de busca por padrões para que o desenvolvedor concentre esforços em aprender a como utilizar as soluções dos padrões e adequá-las aos seus problemas e não em como e onde procurar padrões. Conforme visto no Capítulo 2, existem tipos de padrões de *software* adequados para cada etapa do desenvolvimento de sistemas e para os mais diversos domínios

de aplicação, portanto, para que uma ferramenta seja eficaz no suporte à disponibilização de padrões de *software*, ela deve ser capaz de armazenar diferentes tipos e formatos de padrões. Para isso, foi feito o levantamento dos formatos de padrões e critérios de classificação mais utilizados.

Objetivando investigar o reuso de diferentes tipos de padrões de *software*, foi feito através de um projeto de cooperação entre professores, alunos de mestrado e graduação da UFC e analistas e desenvolvedores do Instituto Atlântico (IA) [7], um estudo sobre o impacto do reuso de padrões de *software* no desenvolvimento de sistemas a fim de identificar os problemas no processo de busca e aplicação de padrões. O IA desenvolve sistemas para diversas áreas tecnológicas, por exemplo, sistemas de suporte a negócios e operações, planejamento, Internet, inteligência de negócios e diversos outros setores de telecomunicações, de energia e do governo.

Ao longo do projeto foram realizadas entrevistas com integrantes de vários projetos em andamento do IA, a fim de:

- Identificar e classificar os padrões utilizados;
- Documentar os padrões usados e propor novos padrões;
- Avaliar o impacto do reuso de padrões de *software* no desenvolvimento de sistemas.

Os resultados desta pesquisa, obtidos a partir da experiência prática dos desenvolvedores, serviram para identificar e vantagens e desvantagens do reuso de padrões de *software* e serviram de base para a especificação dos requisitos do repositório de padrões proposto por esta dissertação.

3.2 Soluções Existentes para o Armazenamento e Busca de Padrões

Os trabalhos relacionados com a construção de um repositório de padrões podem ser divididos em: repositórios na forma de *websites* [65][87]; ferramentas que armazenam padrões de software [20][52]; repositórios de componentes e artefatos [48][67][95][96]; e sistemas de consulta a padrões armazenados em uma fonte de dados [20][48].

Alguns sites funcionam como referências para padrões publicados. Por exemplo, o *Portland Patterns Repository* [65] apresenta um conjunto de páginas descrevendo linguagens de padrões e traz referências a outras páginas. Outro repositório de padrões na forma de *website* é o *Diemen Repository of Interaction Design Patterns* [87] que apresenta uma coleção de padrões para o projeto de *websites* (*Webdesign*).

Existe uma grande quantidade de sites que apresentam coleções de diversos tipos de padrões, principalmente sites de domínios de aplicação específicos, como foi citado na Seção 2.5.2. Entretanto, nenhum destes sites funciona como um repositório que dê suporte ao armazenamento e busca de padrões de forma ampla, abrangendo os diferentes tipos de padrões que existem hoje na literatura.

Em [52] é apresentado o CASEGEO, uma ferramenta CASE que auxilia no projeto de sistemas que utilizam bancos de dados geográficos. Um dos recursos apresentados nesta ferramenta é o suporte à recuperação de diagramas de classes referentes a padrões disponíveis em um repositório de análise.

O protótipo de um ambiente de desenvolvimento para a definição e reutilização de padrões é proposto em [20]. No trabalho é criado um formalismo, denominado *P-Sigma*, para a representação de padrões e de sistemas de padrões. O *template* de padrões utilizado pelo sistema possui os seguintes elementos: identificador, classificação, problema, contexto, forças, solução, diagramas e conseqüências. Este trabalho define formalmente e utiliza os seguintes relacionamentos entre os padrões: usa, refina, requer e alternativo.

Vale ainda mencionar nesta seção o trabalho desenvolvido em [48], onde os autores introduzem uma ferramenta de auxílio ao desenvolvimento de sistemas baseado em componentes. O objetivo do trabalho é propiciar técnicas seguras para desenvolvimento de sistemas em grupos de trabalho, com políticas de manutenção e ferramentas de auxílio à reutilização de software. O repositório de componentes usado, além das informações técnicas dos componentes (endereço do arquivo, tamanho, criador, data de inclusão e atualização), armazena outros atributos dos componentes, tais como *contexto*, *problema* e *solução*. Estes atributos compõem parte de um *template* de padrões de *software* (ver Seção 2.2). O trabalho possui um mecanismo de busca semelhante ao mecanismo de busca utilizado nesta dissertação. Existem dois tipos de busca: uma busca simples, que utiliza a pesquisa entre as palavras-chave inseridas no registro de cada componente, e uma busca avançada, onde a pesquisa é realizada nos atributos dos componentes.

Werner et al apresentam em [67][95][96], resultados de um trabalho que fornece uma infra-estrutura de reutilização de artefatos (e.g., modelos conceituais, arquiteturas de software e modelos de implementação) baseada em modelos de domínio. Neste trabalho é feito um estudo sobre engenharia de domínio, definida por Prieto (1991) como o processo de identificar e organizar o conhecimento sobre uma classe de problemas, o domínio do problema, para suportar sua descrição e solução. Em [96] são descritos aspectos de persistência dos modelos de domínio e em [95] são apresentados aspectos de documentação

de componentes (*templates de componentes*), a navegação inteligente através dos modelos de domínio e o desenvolvimento de aplicações, no contexto da infra-estrutura.

Ao longo do desenvolvimento da pesquisa realizada para a construção do repositório de padrões de *software,* pôde-se observar que pouco tem sido feito no sentido de criar ferramentas que dêem suporte ao armazenamento de vários tipos de padrões de *software*. Existem sim, muitos trabalhos que visam disponibilizar pequenos grupos de padrões para áreas específicas.

O repositório de padrões, que é um dos subprodutos desta dissertação, propõe um suporte ao armazenamento de diversos tipos e formatos de padrões de *software* (ver Seções 2.2 e 2.5), visando preservar sua documentação original. O fato de alguns sistemas escolherem um *template* fixo de padrões limita a inclusão para aquele *template*, fazendo com que um padrão com *template* diferente daquele disponível tenha que ser adequado ao formato, passando a alterar o conteúdo do padrão, descaracterizando-o.

O suporte a inclusão de diversas categorias de padrões é um recurso de grande auxílio para os mecanismos de busca, tendo em vista a existência de vários critérios de classificação de padrões (ver Seção 2.5). Usuários que não têm familiaridade com algumas classificações podem ainda procurar por classificações de domínios de aplicação (i.e., Segurança, GUI, *Web*, XML, entre outros).

O repositório de padrões de *software* proposto, além das documentações dos padrões, visa ainda disponibilizar: código-fonte de padrões; modelos de padrões representados na linguagem UML; e quaisquer outros artefatos referentes aos padrões.

3.3 Funcionalidades do Repositório

Nesta Seção são apresentadas as principais funcionalidades do repositório de padrões de *software* desenvolvido nesta dissertação. A seguir, os requisitos funcionais do sistema que compreende o repositório de padrões são apresentados:

- ***Manter Padrões***: o repositório deve permitir a inclusão de padrões numa base de dados. O repositório não adota um formato específico para os padrões o que torna possível incluir um padrão usando os campos dos *templates* de padrões mais usados na literatura (ver Seção 2.2). Associado ao padrão deve ser cadastrado um conjunto de palavras-chave. O repositório pode associar arquivos relacionados a cada padrão, tais

como: códigos-fonte, modelo do padrão e imagens ilustrativas. Mais detalhes sobre a estrutura utilizada para armazenar os padrões são apresentados na Seção 3.6.

- ***Manter Linguagens de Padrões***: o repositório de padrões deve permitir a inclusão de *Linguagens de Padrões* que consiste na inclusão dos campos que descrevem a linguagem e dos padrões que a compõem.
- ***Manter Catálogos de Padrões***: o repositório de padrões deve permitir a inclusão de *Catálogos de Padrões;* semelhantes às linguagens de padrões, a inclusão de um *Catálogo de Padrões* consiste na inclusão dos campos que descrevem o catálogo e dos padrões que o compõem.
- ***Manter Categorias de Padrões***: o repositório deve permitir o cadastro de categorias dos padrões que são entidades do sistema que classificam os padrões. A Seção 2.5 apresenta uma visão geral sobre tipos de classificação de padrões de *software*.
- ***Associar Padrões***: o repositório deve permitir o registro dos relacionamentos entre os padrões cujos tipos são apresentados na Seção 2.4. Os *templates* dos padrões possuem o componente chamado *Padrões Relacionados* onde o administrador do sistema na inclusão do padrão pode associá-lo a outros padrões cadastrados ou não no sistema.
- ***Buscar Padrões***: o repositório deve possuir uma busca eficiente que forneça como resultado padrões relacionados aos elementos da pesquisa informados pelo usuário. Tornar os padrões disponíveis para busca e a posterior recuperação dos dados é o objetivo principal do repositório. Na Seção 3.5 serão apresentados mais detalhes dos mecanismos de busca adotados.
- ***Visualizar Dados***: o repositório deve disponibilizar aos usuários, todos os dados e arquivos relacionados ao padrão selecionado após uma pesquisa para a visualização em uma tela do próprio sistema ou via *URL.*
- ***Manter Projeto***: o repositório deve permitir manter um cadastro de projetos desenvolvidos ou em desenvolvimento que já utilizaram os padrões armazenados.
- ***Associar Padrão a Projetos***: o repositório deve permitir associar padrões a um ou mais projetos, o que pode vir a gerar referências futuras de quais projetos estão utilizando ou utilizaram um determinado padrão. Da mesma forma é possível saber que padrões foram utilizados por um determinado projeto.

- ***Manter Usuário***: o repositório deve permitir o cadastro dos usuários do sistema em uma categoria de usuário específica (i.e., usuário comum ou administrador), que designará seu papel e suas permissões dentro do sistema (ver Seção 3.4).
- ***Exportar Modelos***: o repositório de padrões deve fornecer os recursos necessários para a disponibilização dos modelos de padrões para a integração com ferramentas de desenvolvimento, através de uma *Interface de Integração*. Apesar de possuir esse nome, a *Interface de Integração* é uma ferramenta que tem por finalidade integrar modelos de padrões de *software* com ferramentas de desenvolvimento e não integrar o repositório de padrões diretamente com uma ferramenta de desenvolvimento. O processo de integração é apresentado no Capítulo 4.

3.4 Utilização do Repositório

A seção anterior apresentou as principais funcionalidades do repositório de padrões que podem ser sumarizadas através de diagramas de casos de uso da UML [10]. Os principais atores do sistema representando os usuários e entidades externas[2] são ilustrados na Figura 6 e descritos a seguir:

- ***Usuário Comum****:* são os usuários do repositório de padrões cadastrados no sistema para buscar e aplicar padrões.
- ***Administrador****:* são os usuários do sistema responsáveis principalmente por manter os dados cadastrais, inclusive o cadastro dos demais usuários. Além disso, eles podem desempenhar o papel de um *Usuário Comum*. Suas funcionalidades específicas são ilustradas na Figura 7.
- ***Base de Dados****:* consiste da fonte de dados utilizada para o armazenamento dos dados do sistema. Atualmente o repositório utiliza um SGBD (Sistema de Gerenciamento de Banco de Dados).
- ***Servidor de Arquivos****:* é o servidor onde residem os arquivos referentes aos modelos dos padrões e os arquivos associados aos campos dos padrões, como imagens ou diagramas adicionais. Esses arquivos são enviados durante o cadastro dos padrões e disponibilizados através da interface *Web* do sistema.

[2] "Entidades externas" é o termo utilizado em análise estruturada para representar entidades que não fazem parte do sistema como: outros sistemas, dispositivos ou periféricos.

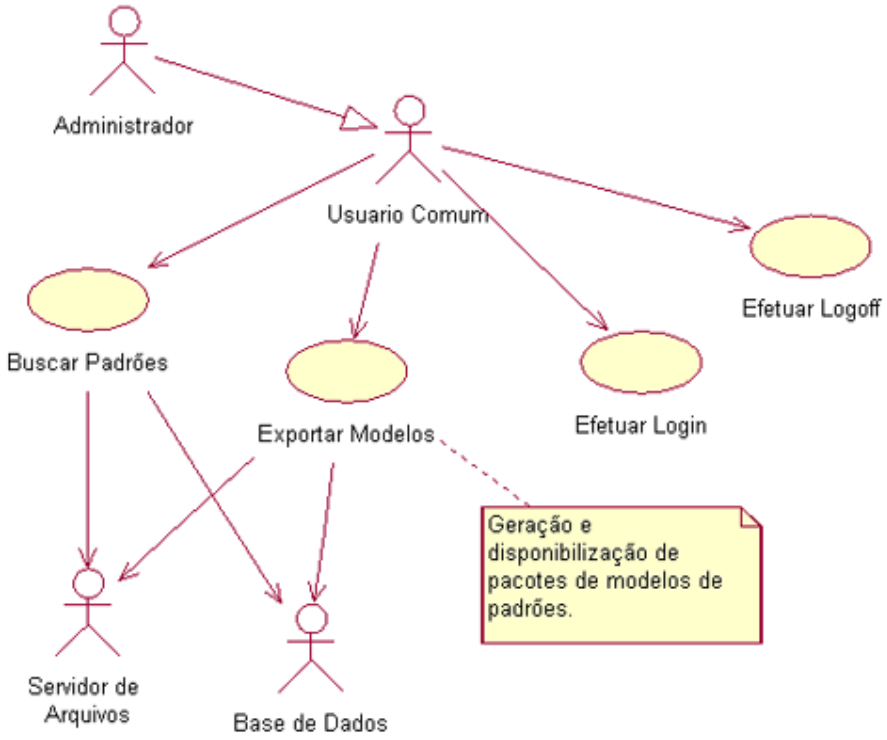

Figura 6 - Diagrama de Casos de Uso referentes à utilização do Repositório de Padrões por Usuários Comuns

Os casos de uso ilustrados nas Figuras 6 e 7 são referentes às funcionalidades do sistema apresentadas na seção 3.3. O principal objetivo do *Usuário Comum* é aplicar um ou mais padrões no desenvolvimento de um sistema. A primeira funcionalidade importante para o usuário é *Buscar Padrões,* que realiza a pesquisa e recuperação dos dados dos padrões armazenados na *Base de Dados* e a posterior recuperação de arquivos no *Servidor de Arquivos*. O caso de uso *Exportar Modelos* refere-se à função de disponibilização de dos modelos de padrões em pacotes no formato apropriado para a reutilização dos mesmos em ferramentas de modelagem através do processo de integração apresentado no Capítulo 4. O formato desses pacotes é apresentado com detalhes no Capítulo 5.

Os casos de uso *Efetuar Login* e *Efetuar Logoff* são responsáveis pelas operações de *Login* e *Logout* do sistema para a autenticação dos usuários.

O *Administrador* deve ser responsável pela manutenção das entidades de dados do sistema, ou seja, *Padrões, Linguagens de Padrões, Catálogos de Padrões, Categorias, Tipos de Relacionamentos, Projetos e Usuários*. O *Administrador* é uma especialização de um *Usuário Comum*, portanto, além de desempenhar o seu papel específico, pode desempenhar o

papel de um *Usuário Comum*. Os casos de uso referentes à manutenção dessas entidades são ilustrados na Figura 7.

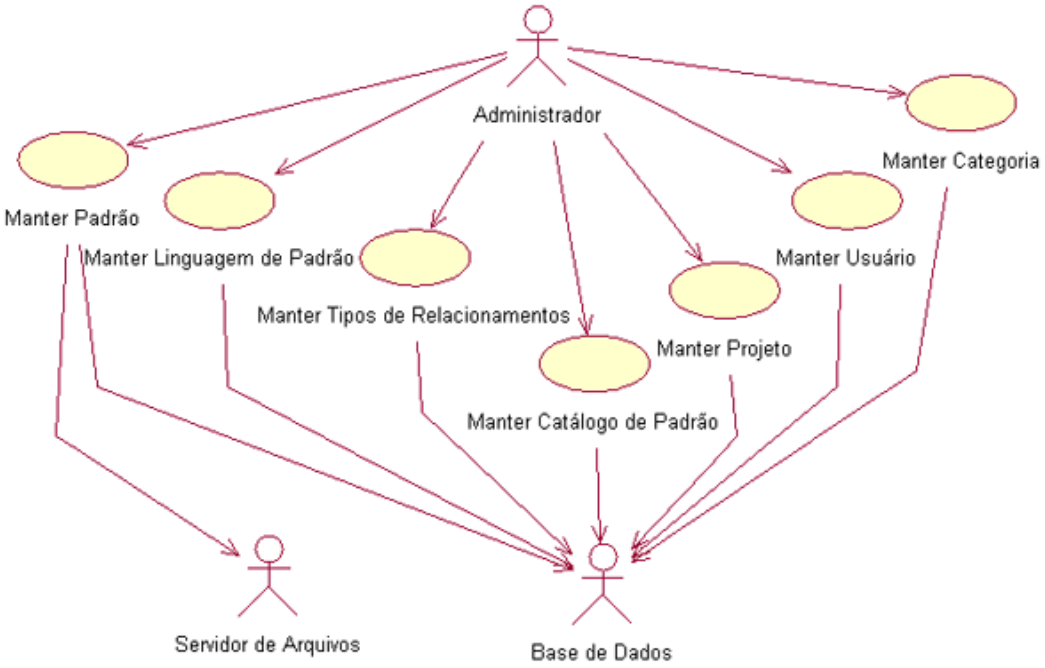

Figura 7 - Diagrama de casos de uso referente às responsabilidades do Administrador

3.5 Mecanismos de Busca do Repositório

Os mecanismos de busca do repositório consistem de pesquisas realizadas sobre os campos que compõem os padrões (ver Figura 8 e Figura 9). O formalismo para a entrada da pesquisa é semelhante a maioria dos sistemas de busca da *web* conhecidos como *search engines* [84]. Operadores lógicos "e" e "ou" são utilizados, bem como operadores de adição e exclusão (respectivamente "+" e "-"). O texto de consulta passa então por um processamento para que sejam definidos todos os parâmetros da consulta. Através de um *parser* são geradas uma pilha de operadores e uma pilha de *frases* (palavras ou frases delimitadas por aspas). O *parser* é responsável então pela decomposição do texto da pesquisa em elementos que irão formar a consulta sobre a fonte de dados. Para o ambiente que é utilizado atualmente, o *parser* devolve ao sistema a próprio consulta *SQL* que será usada na consulta ao Banco de Dados.

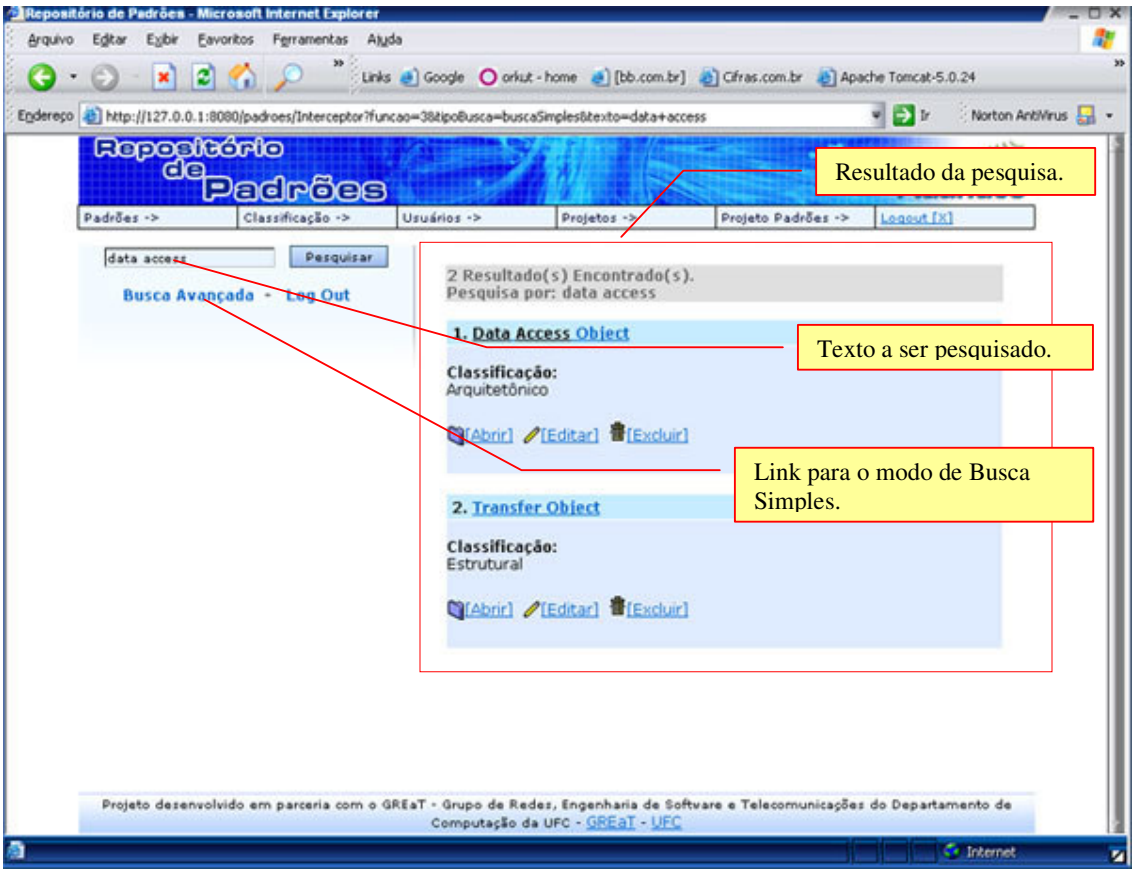

Figura 8 - Busca Simples

Dois tipos de busca, ambos reunindo as características descritas anteriormente, foram elaborados para a realização da pesquisa no repositório de padrões de *software*:

- ***Busca Simples***

A busca simples consiste de uma consulta realizada nos campos nome e palavras-chaves que são adicionadas ao *template* no ato de sua inclusão. A única responsabilidade do usuário é entrar com o texto para a pesquisa e acionar a busca. A Figura 8 mostra a janela proposta para a realização de busca simples no repositório de padrões. Na esquerda da janela estão localizados a caixa de entrada do texto, o botão que aciona a pesquisa e um link para o modo de *Busca Avançada* detalhado no próximo item. Na direita da janela, são listados os resultados encontrados.

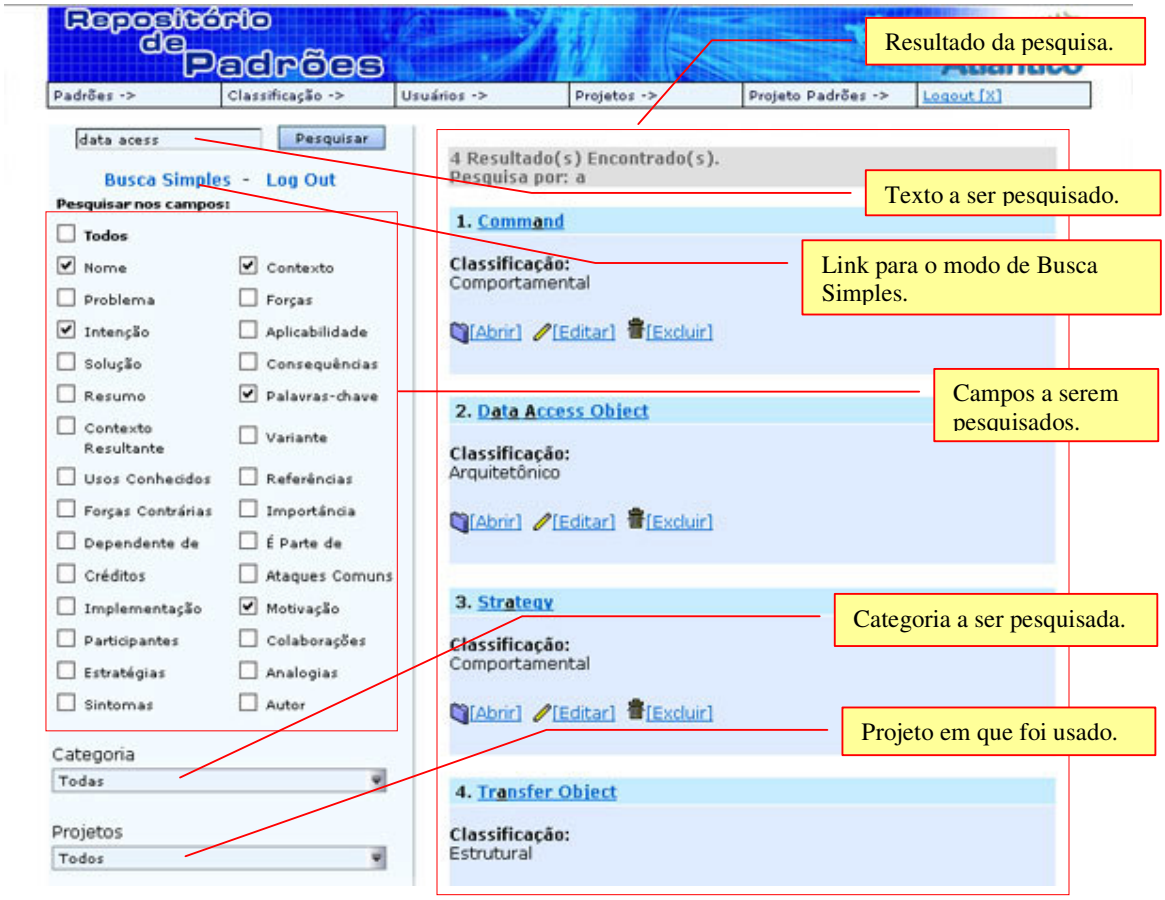

Figura 9 - Busca Avançada

- ***Busca Avançada***

A busca avançada permite que sejam selecionados múltiplos campos do *template* do padrão para a consulta. Pode-se ainda refinar a pesquisa selecionando uma categoria para o padrão procurado, ou ainda os projetos cadastrados pelo sistema que estejam relacionados ao padrão.

A Figura 9 mostra a janela proposta para a realização de busca avançada no repositório de padrões. Na esquerda da janela ficam localizados os parâmetros da consulta que consistem da caixa de entrada do texto de consulta, das opções de seleção múltipla dos campos de pesquisa, da caixa de seleção da categoria do padrão, da caixa de seleção de projetos, do botão que aciona a busca e de um link para o modo de *Busca Simples*. Assim como na Busca Simples os resultados encontrados são listados no lado direito da janela.

3.6 Estrutura e Armazenamento dos Dados do Repositório

Na Seção 2.2, apresentamos alguns dos formatos de padrões mais utilizados na literatura. O fato de não existir um formato comum para os padrões dificulta a modelagem de uma entidade de dados que represente diferentes tipos de padrões (veja também Seção 2.5) e preserve o formato do padrão descrito pelos autores. Para resolver esse problema é necessário o uso de uma abordagem semi-estruturada para o armazenamento dos dados. Como XML [97] tem se tornado uma norma técnica *de facto*[3] para a definição de dados semi-estruturados, um esquema XML para a definição da estrutura de um padrão é um dos requisitos desejáveis para um repositório de padrões.

Para armazenar um padrão definimos os campos componentes de um *template* genérico de um padrão que é apresentado na Tabela 5. Escolhemos campos dos formatos dos padrões mais encontrados na literatura. O *template* gerado constitui então uma união dos conjuntos de campos que formam os *templates* apresentados na Seção 2.2. Estes campos são basicamente de três tipos: campos com valores inteiros (tipo **Inteiro**); campos que possuem somente texto (tipo **Texto**); e campos que precisam suportar uma estrutura mais complexa, onde além de um texto descritivo, o elemento deve permitir que um ou mais arquivos sejam associados a ele (tipo **Complexo**).

Além dos campos pertencentes a *templates* de padrões existentes, outros campos foram adicionados à estrutura para fins de implementação, entre eles:

- *Tipo:* campo que define os direitos de publicação do padrão cadastrado como "*proprietário*" ou "*aberto*". Quando o padrão é proprietário, somente alguns dos campos são disponibilizados para visualização (i.e., *Nome, Resumo, Fonte e Autor*);
- *Palavras-chave:* campo que contém palavras-chave inseridas no cadastro de cada padrão a fim de facilitar o mecanismo de busca;
- *Identificador do Catálogo* e *Identificador da Linguagem:* campos responsáveis pelas referências às entidades *Catálogo de Padrões* e *Linguagem de Padrões* respectivamente, às quais o padrão pode pertencer. Na estrutura do banco de dados relacional fazem o papel de chaves estrangeiras.

Na primeira modelagem dos dados, dois produtos principais foram gerados, o esquema XML e o diagrama de classes UML [10] do modelo, que serviram de base para as outras fases

[3] "*de facto*" refere-se a normas técnicas que foram adotadas pela comunidade, ao contrário das normas "*de jure*" que são normas elaboradas por um órgão de padronização internacional.

do desenvolvimento. No Apêndice A apresentamos o esquema XML gerado que representa a estrutura de um padrão e de outras entidades de dados do repositório.

Campo	Nome Físico	Tipo
Nome	nome	Texto(50)
Identificador do Padrão	Id_padrao	Inteiro
Contexto	contexto	Complexo
Solução	solucao	Complexo
Problema	problema	Complexo
Aplicabilidade	aplicabilidade	Complexo
Estrutura	estrutura	Complexo
Dinâmica	dynamics	Complexo
Intenção	intencao	Complexo
Código-Exemplo	codigo_exemplo	Complexo
Contexto Resultante	contexto_resultante	Complexo
Motivação	motivacao	Complexo
Conseqüências	consequencias	Complexo
Usos Conhecidos	usos_conhecidos	Complexo
Ataques Comuns	ataques_comuns	Complexo
Analogias	analogias	Texto(50)
Resumo	resumo	Texto
Descrição	descricao	Texto
Sintomas	sintomas	Texto
Depende de	depende_de	Texto
É parte de	E_parte_de	Texto
Créditos	creditos	Texto
Referências	referencias	Texto
Colaborações	colaboracoes	Texto
Fonte	fonte	Texto
Palavras-chave	palavras_chave	Texto
Tipo	tipo	Inteiro
Identificador do Catálogo	Id_catalogo	Inteiro
Identificador da Linguagem	Id_linguagem	Inteiro
Autor	autor	Texto(100)

Tabela 5 - Template Genérico de um Padrão

Para modelar a estrutura que representa um padrão, as seguintes classes foram criadas:

- Classe ***Padrao***: representa a entidade principal de um padrão, e associa-se às classes necessárias para a representação dos demais elementos do *template*;
- Classe ***EstruturaType***: representa a entidade que serve de base para os campos descritos na Tabela 5 com o tipo ***Complexo*** que possuem uma estrutura capaz de armazenar um texto (atributo *descrição*) e a associação a um ou mais arquivos (Classe *Arquivo,* descrita a seguir).
- Classe ***Arquivo***: representa os arquivos associados à classe *EstruturaType,*e possui três atributos. O *caminho* guarda o caminho físico (*path*) do arquivo armazenado no servidor de arquivos; a *descrição*, referente à descrição do arquivo; e o *tipo*, que é opcional e possui valor pré-definido como "*estática"* ou "*dinâmica",* referente ao tipo

de estrutura representada pelo arquivo. Estruturas estáticas geralmente se referem a diagramas de classes e estruturas dinâmicas correspondem a diagramas de seqüência, diagramas de colaboração ou outros tipos de diagramas que representam o comportamento dinâmico do sistema em questão.

Para o armazenamento dos modelos de padrões no repositório deve ser feito o envio do arquivo (*upload*) durante o cadastro do padrão. Apesar do sistema de *upload* suportar qualquer tipo de arquivo, os modelos são inseridos no formato XMI para facilitar a integração com outras ferramentas. Essa abordagem é discutida no Capítulo 4.

A Figura 10 ilustra uma parte do diagrama de classes referente à estrutura de um padrão e seu relacionamento com as classes que representam os campos com tipo ***Complexo***. Essas classes são todas subclasses da classe *EstruturaType*, e estão relacionadas à classe *Padrao* através de composições.

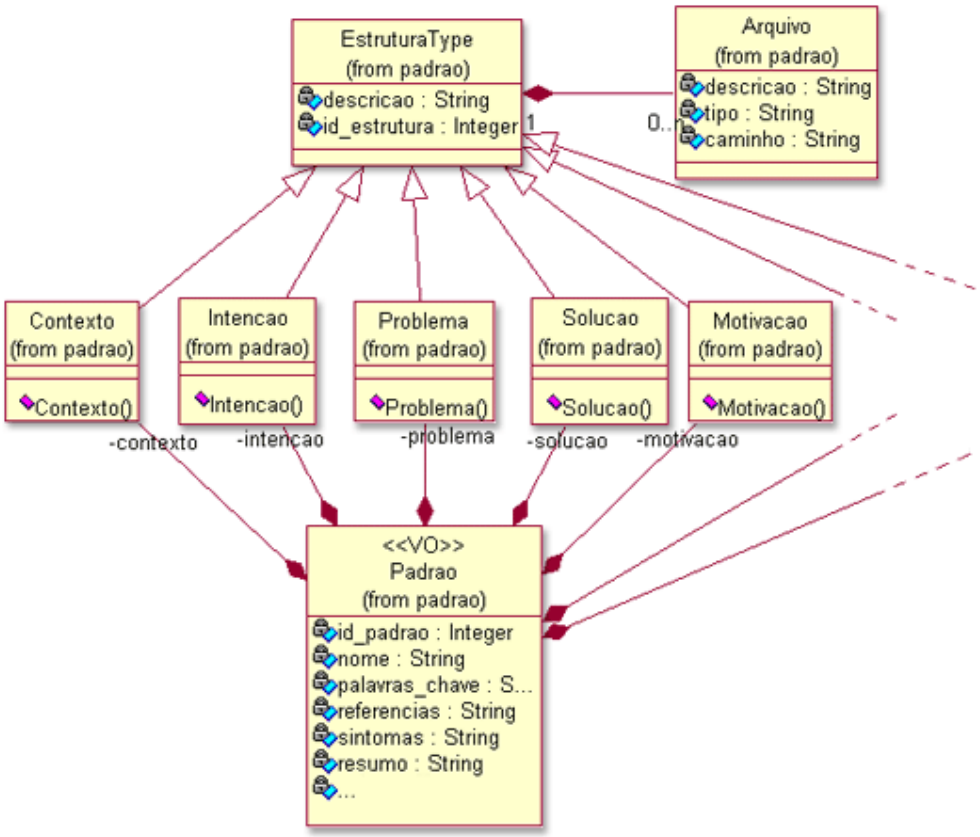

Figura 10 - Estrutura Resumida da Classe Padrão

A abordagem utilizada nas fases iniciais do desenvolvimento do repositório, utilizando XML, aponta para o uso de um banco de dados XML nativo, que é um servidor de dados

capaz de armazenar XML na sua forma original. Um exemplo de um banco de dados XML nativo é o *Tamino* [86]. Alguns SGBDs relacionais, como o *Oracle* [62] e o *SQL Server* [55], já trazem recursos para o armazenamento de documentos XML.

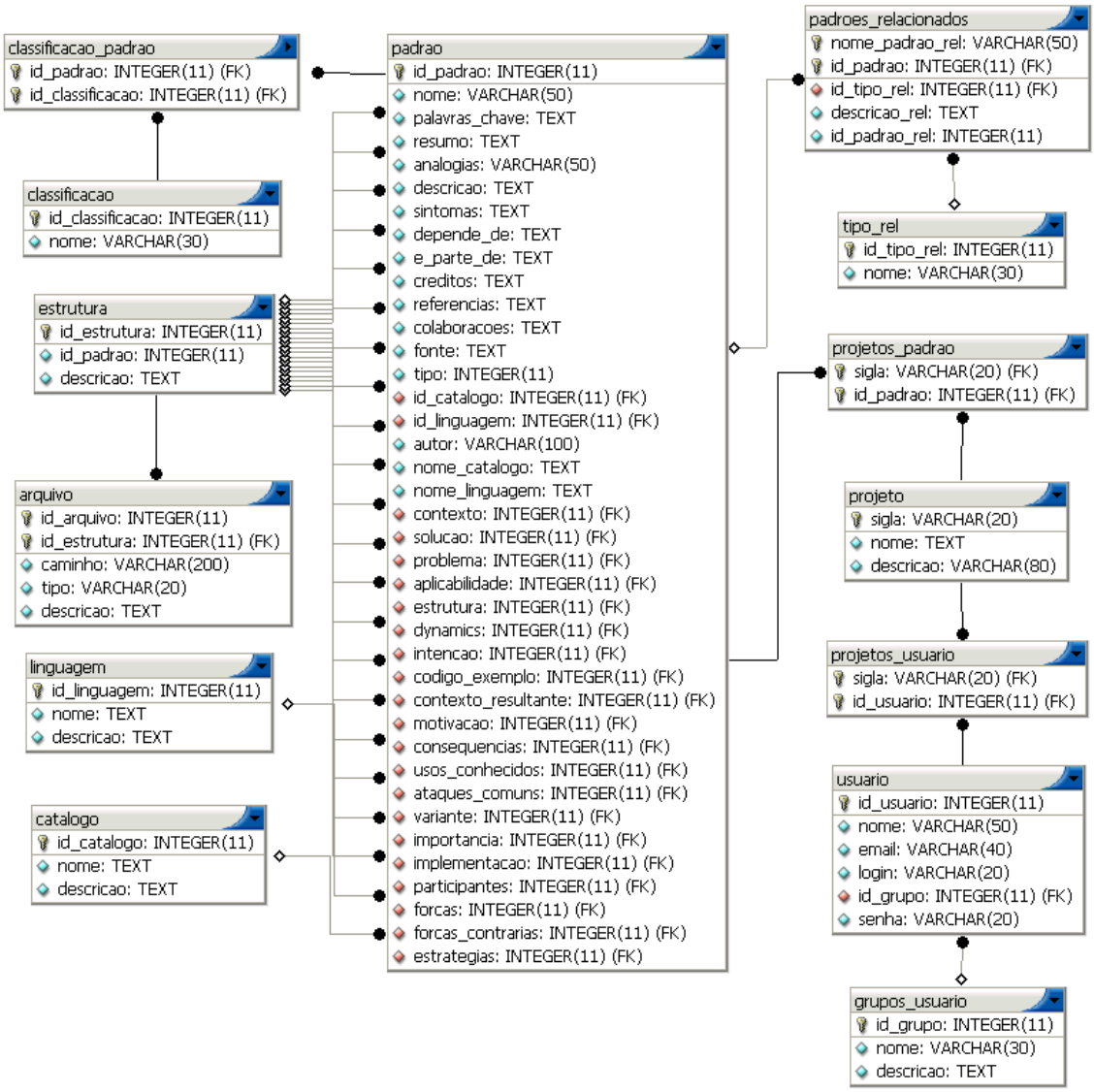

Figura 11 - Diagrama de Entidades e Relacionamentos do Repositório de Padrões

Para a implementação do primeiro protótipo do repositório de padrões, no entanto, foi utilizada uma modelagem relacional dos dados devido a restrições e direitos de uso das tecnologias de armazenamento citadas anteriormente, existentes no nosso ambiente de desenvolvimento. O mapeamento das estruturas em diagramas de entidades e

relacionamentos, a fim de se projetar o banco de dados relacional utilizado foi então desenvolvido.

As demais entidades de dados que são mantidas pelo repositório de padrões, como descrito na Seção 3.3, são apresentadas na Figura 11 que ilustra o diagrama de entidades e relacionamentos que compõem a base de dados do repositório.

3.7 Arquitetura do Repositório de Padrões

O repositório de padrões de *software* apresentado nesta dissertação tem como um dos requisitos funcionais ***visualizar dados***, como descrito na Seção 3.3, onde a sua apresentação é prevista através de uma interface *web*, a fim de aumentar a sua disponibilidade para todos os tipos de usuários. Outras características como manutenibilidade, escalabilidade, suporte a concorrência são requisitos não-funcionais desejados para o repositório.

Esses requisitos foram importantes na escolha da plataforma J2EE para o desenvolvimento do repositório. As tecnologias do J2EE (e.g., JSP, *Servlets*, *JavaBeans*, entre outras), aliadas às características da própria linguagem de programação Java mostraram-se adequadas para a implementação dos componentes do sistema seguindo os requisitos desejados.

Além disso, utilizamos como padrão para a arquitetura do sistema o *Model-View-Controller* (MVC) [41], que ajuda a estruturar os componentes em camadas. O padrão *Web Interceptor*, descrito em [83], é um padrão usado como componente centralizador de requisições *web* e é usado como *Controller* [41] da arquitetura MVC. Vale ressaltar que o padrão *Web Interceptor* funciona como uma adaptação do padrão *Facade* [33] para sistemas Web. Outros padrões de *software* são usados na implementação do repositório como o *Data Access Object* (DAO), o *Value Object* (VO) [24] e o *Abstract Factory* [33].

A Figura 12 mostra a forma na qual a arquitetura do repositório está organizada levando em consideração o browser para visualizar os dados, os elementos da plataforma J2EE e alguns padrões utilizados bem como o SGBD e o servidor de arquivos mencionados na Seção 3.4.

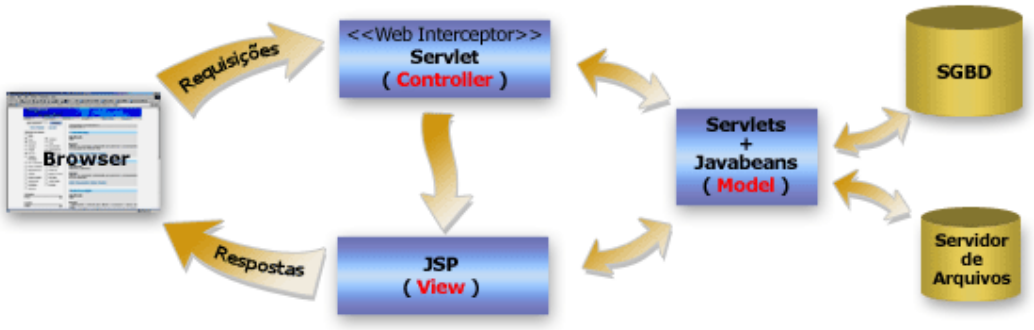

Figura 12 - Arquitetura do Repositório de Padrões

3.8 Conclusão

O repositório de padrões apresentado neste capítulo foi desenvolvido para tentar suprir as necessidades existentes nos processos de armazenamento e busca de padrões de *software*, apresentadas na Seção 3.2, tais como: suporte à inclusão de vários formatos de padrões; uso de várias categorias para classificação de padrões; e o suporte ao armazenamento de modelos de padrões e outros tipos de arquivos.

Os modelos de padrões armazenados no repositório estão disponíveis em um formato adequado para a integração com outras ferramentas. O processo para a integração de modelos de padrões com uma ferramenta de apoio ao desenvolvimento de sistemas, por exemplo, uma ferramenta de modelagem de classes, é o foco principal desta dissertação e é apresentado no capítulo seguinte.

.

Capítulo 4
Um Processo de Integração

Este capítulo apresenta um processo para a integração de modelos de padrões de *software* com ferramentas de apoio ao desenvolvimento de sistemas. A Seção 4.1 introduz a proposta no contexto da engenharia de *software*. Os conceitos essenciais para a abordagem utilizada são discutidos neste capítulo através de uma visão geral da linguagem UML apresentada na Seção 4.2 e da definição de modelos de padrões na Seção 4.3. A Seção 4.4 referencia alguns dos trabalhos relacionados com ferramentas para a aplicação de padrões de *software*. Na Seção 4.5 são apresentados o conceito e algumas características de XMI. O processo de integração proposto é detalhado na Seção 4.6. E, finalmente, uma conclusão sobre a abordagem apresentada é discutida na Seção 4.7.

4.1 Introdução

Atualmente existe um grande número de ferramentas CASE (*Computer Aided Software Engeneering*) que são utilizadas por analistas e desenvolvedores para auxiliar no desenvolvimento de sistemas. Essas ferramentas atuam sobre o planejamento, especificação de requisitos, análise, projeto, codificação e reestruturação de código, com o objetivo de aumentar a qualidade do *software* produzido, melhorar a documentação dos sistemas e aumentar a produtividade.

Com o advento da programação orientada a objetos e o amadurecimento dos processos de produção de *software*, as ferramentas de modelagem UML [10] ganharam destaque devido ao grau de automação oferecido. Essas ferramentas hoje dispõem de recursos como: geração automática de código e engenharia reversa de modelos (*roundtrip engineering* [1][36]), e refatoração (*refactoring* [30]).

Entre os recursos destas ferramentas, um, em particular, é objeto de estudo desta dissertação, a aplicação automática de padrões de *software* usando modelos de padrões. O objetivo principal deste trabalho é criar um mecanismo que auxilie na reutilização automática de modelos de padrões de *software* através da aplicação desses modelos em ferramentas de desenvolvimento. Documentações de padrões são úteis para o aprendizado necessário na aplicação da solução. Se além de sua documentação, dispomos do seu modelo de classes e de

uma ferramenta que possa trabalhar com esses elementos de modelagem, o reuso desses padrões tornar-se-á mais efetivo.

As próximas subseções se dedicam a apresentar os conceitos necessários para entender a aplicação automática de padrões de *software,* exemplos de ferramentas e, por fim, é apresentado o processo de integração de modelos de padrões de *software* com ferramentas de apoio ao desenvolvimento de sistemas proposto por esta dissertação.

4.2 UML

A UML (*Unified Modeling Language*) [10][59] foi criada no final da década de 80 como uma tentativa de padronizar a modelagem de sistemas orientada a objetos. A partir de sua versão 1.1, a UML foi adotada como norma pelo OMG (*Object Management Group*) e adotada pela comunidade de desenvolvimento de *software*.

A UML não é um método, processo ou metodologia, mas sim uma linguagem visual para a construção, especificação, visualização e documentação de artefatos para o desenvolvimento de sistemas. Ela é baseada em diferentes tipos de diagramas que fornecem elementos para a modelagem de todas as etapas do processo do desenvolvimento de um *software*, como resumido a seguir:

- **Diagramas de Classes:** diagramas que mostram conjuntos de elementos de modelos declarativos como classes, seus conteúdos e relacionamentos;
- **Diagramas de Colaboração:** diagramas que mostram as interações organizadas em torno da estrutura de um modelo usando classificadores e associações ou instâncias e *links*;
- **Diagramas de Objetos:** diagramas que mostram os objetos e seus relacionamentos em um determinado ponto no tempo. Um diagrama de objetos pode ser considerado um caso especial de um diagrama de classes ou de um diagrama de colaborações;
- **Diagramas de Casos de Uso:** diagramas que mostram os casos de uso, os atores e seus relacionamentos. São usados para descrever os requisitos do sistema;
- **Diagramas de Seqüência:** diagramas que mostram as interações entre os objetos e as trocas de mensagens entre eles descritas em uma sequência temporal. Diferente dos diagramas de colaboração, os diagramas de sequência incluem sequência de tempo mas não incluem relacionamentos entre os objetos. Um diagrama de sequência pode existir em uma forma genérica (descrevendo todos os cenários possíveis) ou em uma

forma de instância (descrevendo um cenário atual). Diagramas de sequência e de colaboração expressam idéias semelhantes mas de maneiras diferentes;

- **Diagramas de Estados:** diagramas que mostram uma máquina de estados representando a sequência de estados pelos quais um objeto ou uma interação passa durante seu tempo de vida;
- **Diagramas de Atividades:** tipo especial de diagramas de estados que mostram o fluxo entre atividades dentro de um sistema. São semelhantes aos fluxogramas, exceto pela concorrência que pode ser modelada nos diagramas de atividades;
- **Diagramas de Componentes:** diagramas que mostram a organização e a dependência entre um conjunto de componentes;
- **Diagramas de Implantação:** diagramas que mostram a configuração dos nós de processamento e dos componentes, processos e objetos que residem nele. São usados para modelar o ambiente em que o sistema será executado.

Com todos estes recursos, a UML se torna uma ferramenta poderosa no auxílio ao desenvolvimento de sistemas, principalmente quando aliado às boas práticas da engenharia de *software*.

O Processo Unificado da Rational [49], também conhecido como RUP, é um processo iterativo de desenvolvimento de sistemas que fornece uma abordagem disciplinada para a atribuição de tarefas e responsabilidades para desenvolvedores e equipes de desenvolvimento. O RUP é desenvolvido e mantido pela *Rational Software* e integrado com seu conjunto de ferramentas para o desenvolvimento de *software*.

O RUP suporta práticas de desenvolvimento de sistemas como: desenvolvimento iterativo e incremental de sistemas, gerenciamento de requisitos, uso de arquiteturas baseadas em componentes, visualização de modelos de *software,* verificação da qualidade do *software* e controle de mudanças no *software*.

Larman [51] apresenta aspectos importantes para a análise e projeto orientados a objetos aplicando UML, Padrões e o Processo Unificado, destacando a importância de se aprender a como projetar sistemas orientados a objetos e não somente aprender uma notação. Em seu livro, Larman levanta questões, por exemplo: Como as responsabilidades devem ser alocadas para as classes e objetos? Como os objetos devem interagir? Que classes devem fazer o quê? O autor destaca, mais uma vez, a importância de seguir um processo de desenvolvimento como o RUP e aplicar padrões de *software* para criar melhores projetos.

4.3 Modelos de Padrões

Um **modelo de *software*** é uma representação física ou abstrata de um sistema. Como visto na Seção anterior, a UML é uma linguagem visual poderosa para o auxílio no desenvolvimento de sistemas orientados a objetos através da representação de modelos de *software* [57].

Modelos de padrões são a representação dos elementos de modelagem utilizados para a aplicação de padrões de *software* em sistemas orientados a objetos. Alguns tipos de padrões, tais como, padrões arquitetônicos e padrões de projeto, geralmente agregam à sua documentação modelos na forma de diagramas descritos na linguagem UML. Padrões como os da coleção da *Gang of Four* [33], por exemplo, vêm acompanhados de diagramas de classes representando sua estrutura, a fim de tornar o seu aprendizado e reuso mais fácil.

A disponibilização de modelos de padrões em repositórios de dados e/ou em ferramentas facilita o processo de reuso dos modelos na prática. A Sessão seguinte descreve algumas ferramentas de apoio ao desenvolvimento de sistemas que trazem recursos para a aplicação de modelos de padrões de software.

4.4 Ferramentas Existentes Para a Aplicação de Padrões de Software

Ferramentas para o auxílio à aplicação de padrões de software que trabalham com geração automática de código [13][19][20] e de modelagem de classes [68][56][89][11], estão disponíveis na literatura e são apresentadas a seguir.

Estas ferramentas podem aplicar padrões de duas formas, a inserção *estática* e a inserção *dinâmica* de modelos de padrões [15]. Na inserção **estática** os elementos são inseridos sem que haja uma maior interação do usuário no processo de delegação de responsabilidades desses elementos em um modelo existente. A outra forma de inserção é chamada **dinâmica**, onde o usuário deve informar, no ato da aplicação do padrão, que elementos do modelo existente desempenham papéis específicos entre os elementos do modelo inserido.

Budinsky et al [13] apresentam uma das primeiras ferramentas para a geração automática de código para padrões. A ferramenta adota o formato apresentado em [33] para o *template* de padrões e gera código-fonte referente aos padrões a partir de alguns parâmetros

informados pelo usuário de maneira dinâmica. O processo de *wizard* permite que a implementação gerada do padrão esteja adequada às necessidades do problema.

O CodePro Studio [19] é uma ferramenta comercial que também é integrado a ambientes Java de desenvolvimento como o Eclipse [27] e o Websphere Studio [38]. O módulo *Java Pattern Wizard* da ferramenta permite gerar classes implementando vários padrões de projeto e outros idiomas Java. Os padrões catalogados na ferramenta são divididos em 6 categorias, a seguir: padrões de Criação, Comportamentais, Estruturais, GUI (Interface Gráfica), J2EE e Teste. Novos padrões podem ser inseridos ao repositório criando novas instâncias xml dos padrões. A estrutura do arquivo xml, definida pelo sistema, é bastante limitada (*id*, *name*, *icon*, *description*, *category*, *source* e *strategy*), o que faz com que a documentação dos padrões não seja armazenada em seu formato original.

O UMLStudio [89] é uma ferramenta comercial de modelagem de classes em UML que possui um catálogo de padrões que podem ser selecionados em uma lista e inseridos na modelagem. A ferramenta inclui a modelagem dos seguintes padrões de [33]: *Command, Iterator, Memento, Abstract Factory, Factory Method, Adapter* e *Proxy*. A inserção é feita de forma *estática*, onde qualquer interação com classes já existentes ou modificações nas classes ou nos métodos devem ser feitas manualmente após a inserção.

Uma abordagem *dinâmica* para a aplicação de modelos de padrões é implementada no ModelMaker [56] e no Together [11]. O ModelMaker é uma ferramenta comercial com suporte à modelagem de dados em UML, geração de classes e geração de pacotes de componentes para o Borland Delphi. O ModelMaker implementa os seguintes padrões de [33]: *Adapter, Mediator, Singleton, Decorator, Visitor* e *Observer*. Além disso, ele implementa os padrões *Lock* [53][56] e *Reference Count* [56]. A ferramenta tem um nível de automação avançado, por exemplo, classes envolvidas com padrões respondem de maneira automática e inteligente às mudanças feitas pelo projetista em outras partes do modelo UML.

O IBM Rational XDE Developer [69] é ao mesmo tempo um ambiente de desenvolvimento para modelagem de dados e um ambiente de programação, com versões para Java e .Net. O XDE Developer possui um recurso para a aplicação de modelos de padrões da *Gang of Four* e de modelos de padrões definidos pelo usuário. O usuário pode, dentro da própria ferramenta, especificar os elementos de modelagem de um padrão e definir os seus parâmetros[4]. A grande diferença dos recursos disponíveis nesta ferramenta e a abordagem utilizada por esta dissertação é que o XDE Developer utiliza ou os padrões disponibilizados

[4] Os parâmetros de um modelo de um padrão são os elementos do padrão que são adequados ao projeto em questão, como classes, métodos ou atributos.

pela ferramenta ou os padrões modelados por usuários utilizando a própria ferramenta. A nossa proposta consiste em utilizar modelos de padrões a partir de um formato de especificação padronizado e aberto, o XMI.

O Rational Rose [68], utilizado para implementação do estudo de caso desta dissertação, é uma ferramenta comercial para modelagem de dados na linguagem UML. O Rose, a partir da versão 2001.04.00, possui o suporte à aplicação de alguns padrões da *Gang of Four*. Entretanto, esse suporte é limitado a estes padrões. O Rose fornece recursos para a expansão de suas funcionalidades (ver Seção 5.2.1). Um exemplo da utilização dos recursos de extensão do Rose para a aplicação de padrões é apresentado em [29], onde é apresentada uma forma de definir e combinar elementos de padrões (classes e relacionamentos) utilizando a aplicação de *scripts* no Rose. Na *web* encontram-se também *plug-ins* para a aplicação de padrões isolados através da execução de *scripts* no Rose , como o *Singleton Pattern Script* [46], que é um *plug-in* para a aplicação do padrão de projeto *Singleton* [33], disponibilizado pela própria Rational. Utilizando os recursos de extensão foi possível implementar o estudo de caso desta dissertação para a aplicação do processo de integração proposto. O Rational Rose é apresentado com mais detalhes no próximo capítulo.

A Tabela 6 apresenta um resumo da comparação feita entre as ferramentas analisadas e o processo de integração proposto por esta dissertação.

Ferramentas	Nível de Automação	Modo de Inserção	Extensão do Conjunto de Padrões	Formato de Extensão
Budinsky	código-fonte	dinâmico	X	–
CodePro Studio	código-fonte	estático	✓	XML (próprio)
ModelMaker	diagramas	dinâmico	X	–
Borland Together	diagramas	dinâmico	X	–
UMLStudio	diagramas	estático	X	–
Rational Rose	diagramas	dinâmico	X	–
IBM Rational XDE Developer	diagramas	dinâmico	✓	Proprietário
Processo de Integração Proposto	Código-fonte ou diagramas	estático	✓	XMI (OMG)

Tabela 6 - Comparação entre os trabalhos existentes e o proposto

4.5 XMI (XML Metadata Interchange)

O processo de integração proposto nesta dissertação consiste em utilizar modelos de padrões de *software* a partir de um formato de especificação padronizado e aberto, o XMI (*XML Metadata Interchange*) [60].

Como é citado no Capítulo 3, XML é um padrão para definição e representação de dados, utilizado atualmente para diversos fins, por exemplo: descrever, disponibilizar e trocar dados entre sistemas.

O XMI é um padrão para troca de modelos de sistemas orientados a objeto, que tenta solucionar o problema de interoperabilidade entre ferramentas de modelagem, repositórios de meta-dados e outras ferramentas de desenvolvimento através da definição de um padrão de codificação genérico, o XML, e da definição de um padrão para os esquemas conceituais, chamado MOF (*Meta Object Facility*) [57]. O MOF é um padrão usado para definir, manipular e integrar meta-modelos, entre eles, os orientados a objetos como, por exemplo, a UML. Por este motivo, MOF é considerado um meta-meta-modelo. O MOF define conceitos como pacotes, classes, métodos e atributos. O XMI especifica como devem ser gerados documentos XML para a representação de modelos de dados a partir de MOF [70][58].

Por possuir as características mencionadas anteriormente, o XMI foi escolhido como formato de armazenamento dos modelos de padrões disponíveis no repositório de padrões de *software* apresentado no Capítulo 3 e como formato de entrada para o processo de integração com ferramentas de apoio ao desenvolvimento de sistemas.

A Figura 13 ilustra algumas das possibilidades de uso do XMI no desenvolvimento de sistemas.

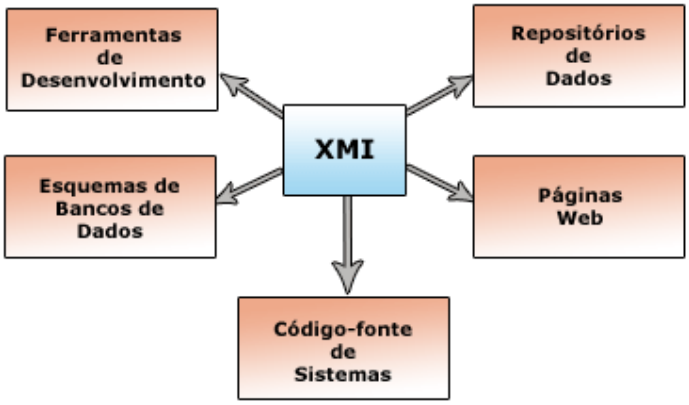

Figura 13 - Uso do XMI (Adaptado de [61])

XMI define uma rigorosa abordagem para geração de uma XML DTD (*Document Type Definition*) a partir de um meta-modelo e a geração de instâncias de documentos XML. A Tabela 7 mostra alguns atributos da sintaxe XMI e suas definições.

Atributo XMI	TIPO	Definição
xmi.id	ID	Define um identificador do elemento que contém esse atributo.
xmi.idref	IDREF	Contém uma referência a um atributo xmi.id correspondente em um outro elemento do mesmo documento.
xmi.label	CDATA	Fornece uma descrição para o elemento.
href	CDATA	Contém um ponteiro para um elemento dentro do mesmo documento ou que esteja em outro documento.
xmi.value	CDATA	Contém um dos valores da lista de enumeração, definida na DTD.

Tabela 7 – Exemplos de Atributos XMI

Na Figura 14 apresentamos um exemplo de um trecho de código XMI que representa um modelo contendo as classes *Departamento*, *Instrutor*, *Professor* e *Posdoc*, e uma associação entre as classes *Instrutor* e *Departamento*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<XMI version="1.1" xmlns:UML="org.omg/UML1.3">
<XMI.header>
  <XMI.model xmi.name="Modelo Departamento" href="Departamento.xml" />
  <XMI.metamodel xmi.name="UML" href="UML.xml" />
</XMI.header>
<XMI.content>
  <UML:Class name="Departamento" xmi.id="1" />
  <UML:Class name="Instrutor" xmi.id="2" />
  <UML:Class name="Professor" xmi.id="3" generalization="Instrutor" />
  <UML:Class name="Posdoc" xmi.id="4" generalization="Instrutor" />
  <UML:Association>
      <UML:Association.connection>
        <UML:AssociationEnd name="instrutores" type="Instrutor" />
        <UML:AssociationEnd name="membroDe" type="Departmento" />
      </UML:Association.connection>
    </UML:Association>
</XMI.content>
</XMI>
```

Figura 14 - Exemplo resumido de um arquivo XMI

Além das características apontadas anteriormente, o uso de XMI é bastantte apropriado devido à variedade de recursos oferecidos para a manipulação e transformação de dados no formato XML, o que facilita a geração de outros arquivos a partir do XMI.

4.6 O Processo de Integração

O processo de integração proposto nesta dissertação define as etapas e atividades necessárias para a implementação de uma *Interface de Integração* que tem como principal função receber como entrada modelos de padrões de *software,* no formato XMI, e gerar os

componentes de extensão necessários para a sua aplicação em uma ferramenta de desenvolvimento. Um cenário genérico para a aplicação do processo de integração é ilustrado na Figura 15.

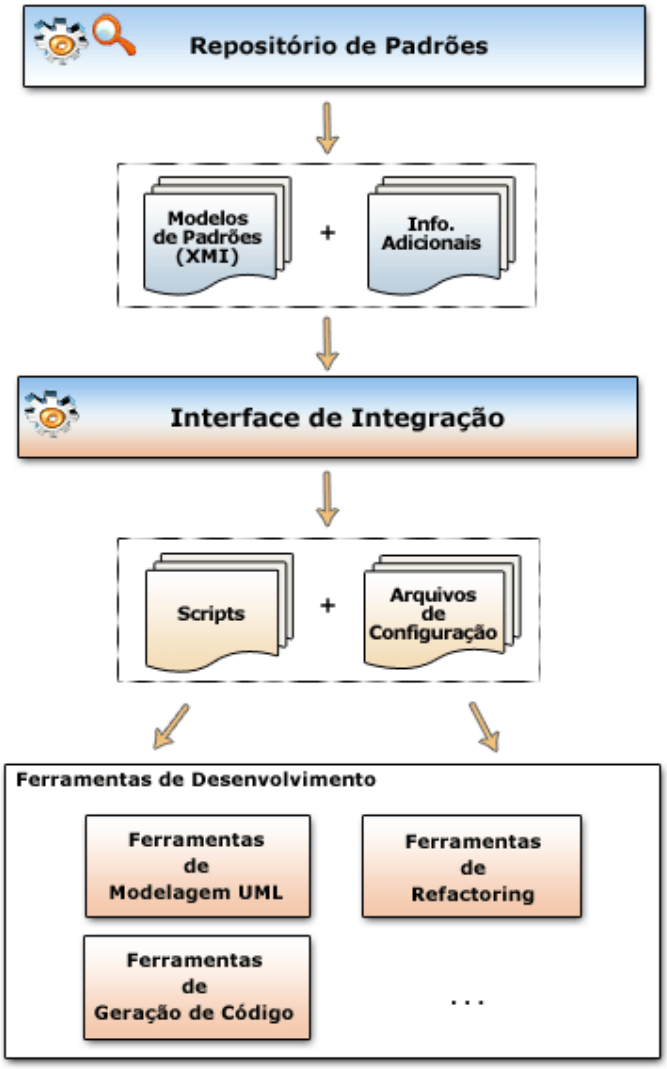

Figura 15 - Cenário de Aplicação da Interface de Integração

A *Interface de Integração* funciona como uma porta de entrada para inclusão de recursos para a aplicação de novos modelos de padrões na ferramenta de desenvolvimento para qual foi implementada. Esta inclusão é feita por demanda, por exemplo, o desenvolvedor pode inserir um modelo de um padrão hoje, amanhã pode inserir outros modelos de padrões encontrados e, assim por diante, de acordo com as suas necessidades.

Além de gerar os arquivos responsáveis pela aplicação do padrão, a *Interface de Integração* deve fornecer os recursos necessários para a personalização da ferramenta de

desenvolvimento escolhida. Portanto, é necessário que estejamos trabalhando com uma ferramenta de código aberto ou com ferramentas comerciais que forneçam recursos de extensão, como o Rational Rose, ferramenta de modelagem escolhida como estudo de caso para este trabalho.

Em conjunto com o repositório de padrões (ver Capítulo 3), a *Interface de Integração* funciona como uma ponte de atualização de modelos para a ferramenta de desenvolvimento para qual foi implementada. Sempre que o usuário encontrar um padrão no repositório, que tenha o modelo XMI disponível, ele pode inseri-lo como um novo recurso da ferramenta de desenvolvimento. Este processo de obtenção de um modelo XMI no repositório é realizado, atualmente de forma manual, onde o usuário realiza o download do(s) arquivo(s) e o fornece como entrada para a *Interface de Integração*.

Para a implementação do processo de integração, três etapas devem ser executadas. Existe um componente da *Interface de Integração* responsável por cada etapa do processo, como mostra a Tabela 8. As atividades de cada etapa são descritas nas subseções seguintes.

Etapa	Componente Responsável
1. Conversão	Conversor de Modelos
2. Preparação	Gerador de Configuração
3. Ativação	Ativador

Tabela 8 - Etapas do Processo de Integração

Os componentes responsáveis pela execução das três etapas do processo de integração formam a *Interface de Integração,* ilustrada na Figura 16.

A etapa 1 (***Conversão***) consiste em gerar, a partir dos modelos no formato XMI, os arquivos necessários para a aplicação dos modelos de padrões na ferramenta de desenvolvimento escolhida. A etapa 2 (***Preparação***) consiste em produzir todos os demais arquivos necessários para a etapa 3 (***Ativação***), que é onde a ferramenta de desenvolvimento é configurada e os seus elementos de extensão para a aplicação dos padrões são instalados.

Figura 16 - Arquitetura da Interface de Integração

As atividades das etapas do processo de integração são ilustradas na Figura 17 através de um diagrama de atividades UML. As colunas delimitam as atividades por etapa do processo. A barra superior de cada coluna mostra o nome do componente que executa as atividades de cada etapa.

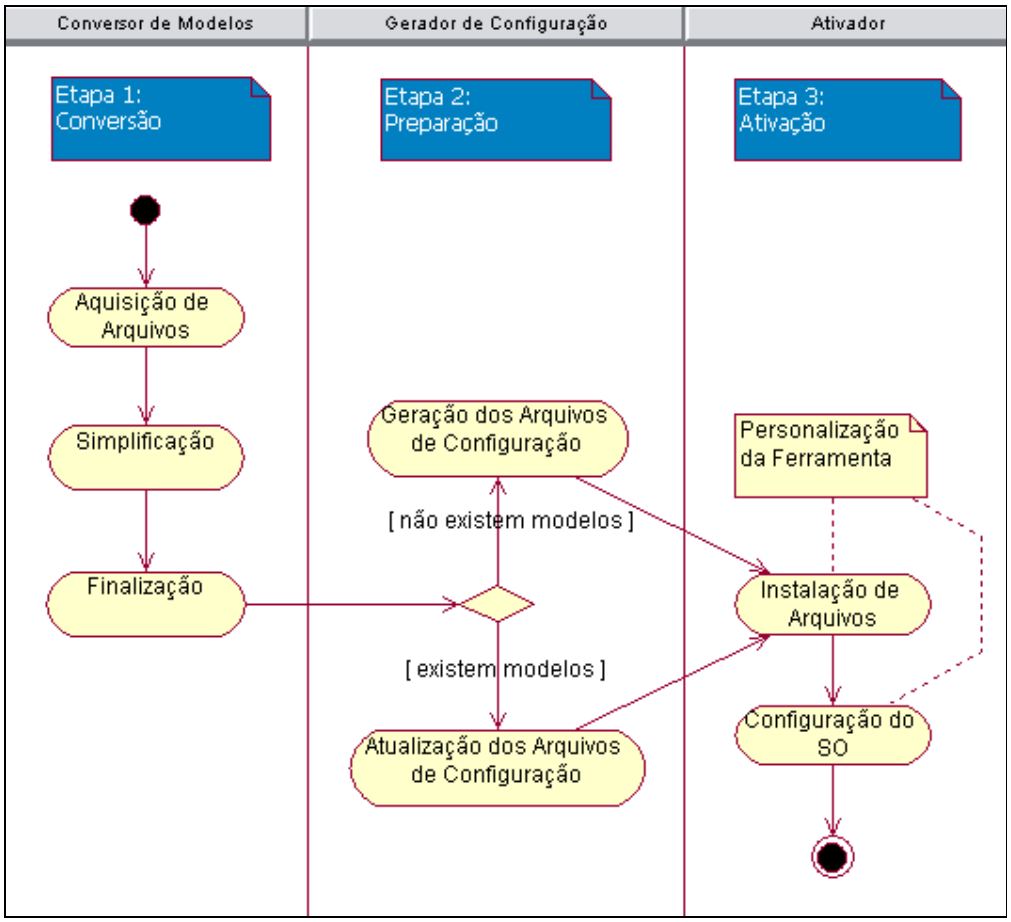

Figura 17 - Atividades das Etapas do Processo de Integração

As relações de dependência entre as etapas do processo de integração são ilustradas na Figura 18 através de um diagrama de casos de uso, onde os casos de uso representam as

etapas do processo e suas atividades. As relações de dependência são expressas no diagrama através dos relacionamentos entre os casos de uso com o estereótipo "*depende da*", que indica que a etapa 2 (*Preparação*) depende da execução da etapa 1 (*Conversão*), e a etapa 3 (*Ativação*), depende das etapas 1 e 2.

Os demais relacionamentos entre os casos de uso indicam relação de inclusão. A etapa 1, expressa pelo caso de uso *Conversão*, inclui na sua execução as atividades *Aquisição de Arquivos*, *Simplificação* e *Finalização,* também expressas por casos de uso. A etapa 2 expressa pelo caso de uso *Preparação,* inclui as atividades *Geração de Arquivos de Configuração* e *Atualização dos Arquivos de Configuração*. A etapa 3, expressa pelo caso de uso *Ativação*, inclui na sua execução a atividade *Personalização da Ferramenta*, que por sua vez inclui as atividades *Instalação de Arquivos* e *Configuração do Sistema Operacional*. Essas atividades são detalhadas nas próximas subseções.

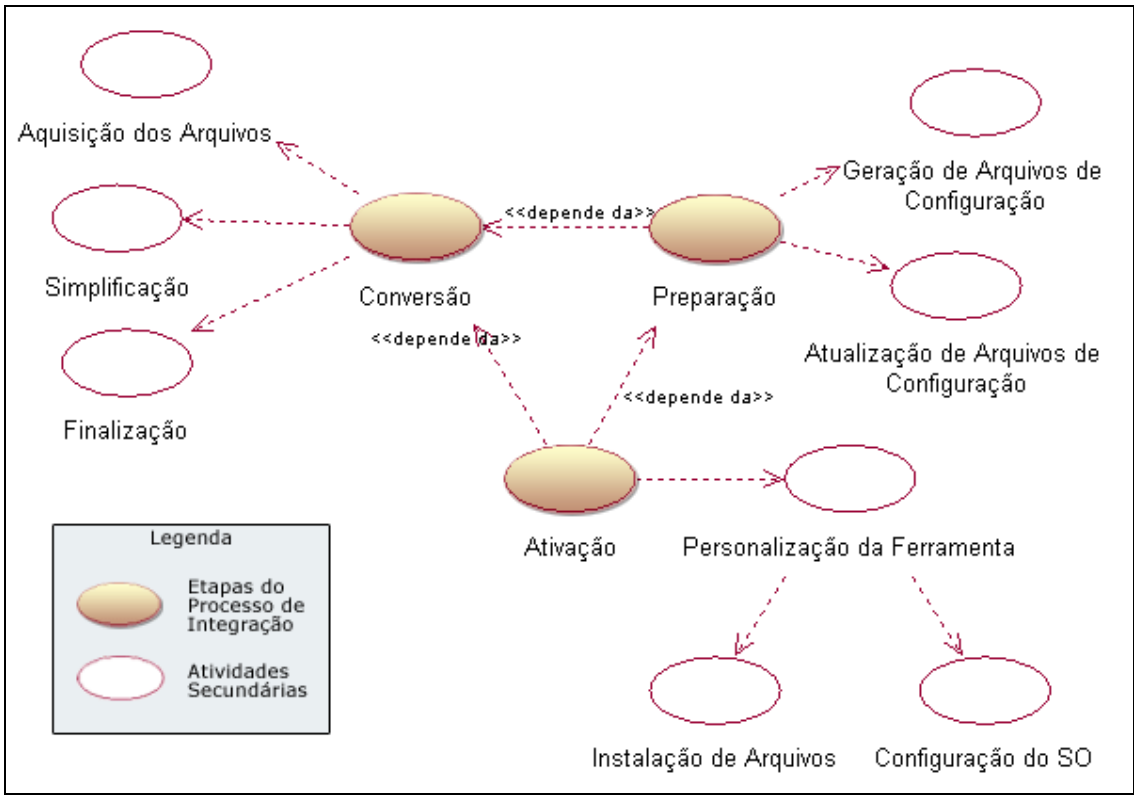

Figura 18 - Dependência entre as Etapas do Processo de Integração

4.6.1 Etapa 1: Conversão

A principal etapa do processo de integração é a conversão dos modelos do formato XMI para um formato aplicável na ferramenta de desenvolvimento escolhida. Na prática, essa conversão consiste em gerar os arquivos necessários para a aplicação do padrão em uma ferramenta de desenvolvimento a partir de um arquivo XMI que contém o modelo de padrão. Esta etapa é executada pelo componente *Conversor de Modelos* da *Interface de Integração*.

Como foi ilustrado na Figura 18, e comentado na Seção anterior, para a execução do processo de conversão são executadas três atividades, a ***Aquisição de Arquivos***, a ***Simplificação*** e a ***Finalização***. Essas atividades são descritas a seguir.

4.6.1.1 *Aquisição dos Arquivos*

Esta atividade consiste em ter acesso ao conteúdo dos arquivos referentes aos modelos de padrões no formato XMI e dos arquivos necessários para a etapa de *Preparação* (ver Seção 4.6.2).

Para a personalização da ferramenta de desenvolvimento durante a etapa de *Ativação*, a *Interface de Integração* precisa de mais informações do que as contidas no modelo XMI. Não se pode garantir, por exemplo, que o nome do padrão é uma informação que é trazida pelo arquivo XMI. No entanto, o nome do padrão é uma informação necessária para a criação de opções de menu, por exemplo. Assim como o nome do padrão, outras informações adicionais podem ser adquiridas pela *Interface de Integração*, como: o nome da coleção a qual o padrão pertence, o resumo do padrão ou qualquer outra informação sobre o padrão que se deseje disponibilizar na ferramenta de desenvolvimento.

O repositório de padrões de *software* apresentado no Capítulo 3 disponibiliza os modelos dos padrões através de arquivos no formato XMI ou em pacotes (no formato .zip), que contêm, além do arquivo XMI, um arquivo texto no formato .ini[5] com informações adicionais sobre o padrão. A estrutura dos pacotes e o formato do arquivo .ini são ilustrados na Figura 19a e Figura 19b usando como exemplo um modelo do padrão *Mediator* da *Gang of Four* [33].

[5] Os arquivos com extensão ".ini" geralmente são do tipo texto no padrão ASCII, com várias opções de configurações dispostas em seções.

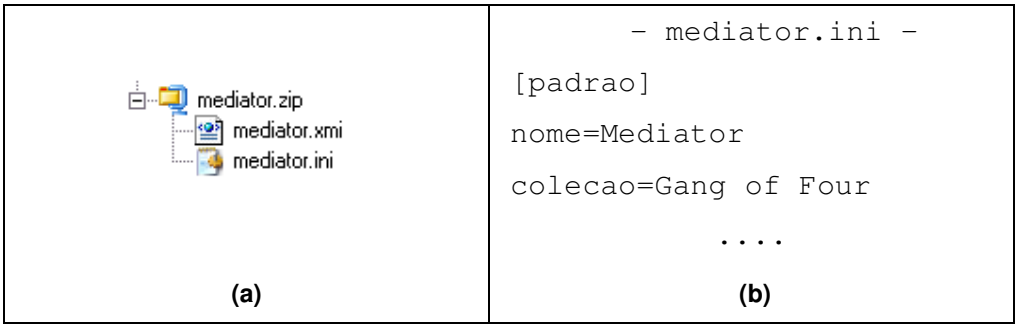

Figura 19 – (a): Estrutura do pacote de arquivos que pode ser recebido como entrada pela Interface de Integração. (b): Formato do arquivo .ini contendo as informações adicionais sobre o padrão aplicado

4.6.1.2 *Simplificação e Finalização*

O processo de conversão consiste em duas fases. A *Simplificação* é comum a todas as implementações do processo. A diferenciação entre as implementações para ferramentas diferentes ocorre na *Finalização*. A duas fases do processo de conversão são ilustradas na Figura 20.

Figura 20 - Processo de Conversão dos Modelos

Ambas as fases de conversão são realizadas utilizando XSLT (*Extensible Stylesheet Language Transformations*) [100], que é uma linguagem usada para realizar transformações em arquivos XML, convertendo-os em outros arquivos XML, em HTML ou em texto.

O objetivo da fase de ***Simplificação*** é converter um modelo do formato XMI para um outro arquivo XML mais simples, que estamos chamando de ***XML Simplificado***. Este segundo arquivo XML é mais simples por dois motivos: pelo fato de conter somente os elementos de modelagem necessários para a próxima fase; e por possuir nomes de elementos mais simples. A partir do *XML Simplificado* são gerados os arquivos no ***Formato final de aplicação*** pela fase de *Finalização*.

A Tabela 9 contém o mapeamento dos principais elementos do formato XMI para o formato *XML Simplificado* considerados para este trabalho.

Elemento	XMI (versão 1.0)	XML simplificado
Classe	<Foundation.Core.Class xmi.id=””> </Foundation.Core.Class>	<class id=””> </class>
Atributo	<Foundation.Core.Attribute> </Foundation.Core.Attribute>	<attribute> </attribute>
Operação	<Foundation.Core.Operation xmi.id=""> </Foundation.Core.Operation>	<operation id=””> </operation>
Parâmetro	<Foundation.Core.Parameter xmi.id=""> </Foundation.Core.Parameter>	<parameter id=””> </parameter>
Associação	<Foundation.Core.Association xmi.id=""> </Foundation.Core.Association>	<association id=””> </association>
Generalização	<Foundation.Core.Generalization xmi.id=""> </Foundation.Core.Generalization>	<generalization> </generalization>

Tabela 9 . Mapeamento de Elementos de XMI para XML simplificado

O fato de o processo de conversão possuir duas fases facilita a criação de transformações para outros formatos na fase de *Finalização* e, por conseqüência, facilita a extensão do processo para outras ferramentas. Essa abordagem de transformação em duas fases é utilizada em [43], onde são definidos *templates*[6] XSL para a geração de vários tipos de arquivos a partir de arquivos XMI. O *script* XSL utilizado para realizar a transformação da fase de *Simplificação* encontra-se no Apêndice B desta dissertação.

As duas fases facilitam ainda a inclusão do suporte a novos elementos de modelagem. Atualmente, considerando o vasto conjunto de elementos de modelagem existentes na UML, e que possuem representação em XMI, o conversor de modelos trabalha com os elementos mostrados na Tabela 9 e os sub-elementos listados a seguir:

- Classes
  - Nome
  - Estereótipo
  - Documentação
- Atributos
  - Nome
  - Tipo
- Operações
  - Nome
  - Tipo de Retorno
  - Parâmetros
    - Nome
    - Tipo

[6] O termo “*template*” aqui se refere aos arquivos no formato XSL utilizados para realizar as transformações XSLT em arquivos XML, diferente do termo “*template*” definido no Capítulo 2 e utilizado como sinônimo para o termo “*formato*” de um padrão de *software*.

- Associações
  - Navegabilidade
  - Cardinalidade
  - Agregação
- Generalizações/Especializações

A fase de ***Finalização*** do processo de conversão de modelos é particular a cada implementação do processo de integração, ao contrário da *Simplificação*, que é comum a todas as implementações. A *Finalização* é a mais importante do processo, pois os arquivos nela gerados são os responsáveis pela aplicação do modelo do padrão na ferramenta de desenvolvimento em questão. Essa fase deve utilizar uma transformação XSL no arquivo *XML Simplificado* gerado na *Simplificação*, a fim de gerar os arquivos no *Formato final de aplicação*, ilustrado na Figura 20.

No estudo de caso introduzido no Capítulo 5, a fase de *Finalização* gera *scripts* que são executados pelo Rational Rose para a aplicação do modelos através da inclusão dos elementos de modelagem do padrão.

Todas as três etapas do processo de integração possuem particularidades que dependem da ferramenta de desenvolvimento para qual a *Interface de Integração* foi implementada. Um exemplo de um outro cenário para aplicação do processo de integração proposto é a sua implementação para a geração de recursos para aplicação de *refactoring* [30] em ambientes de desenvolvimento. Uma das maneiras de se aplicar padrões de *software* a código-fonte é através do uso de técnicas de *refactoring*. A aplicação do processo, neste caso, consistiria em implementar uma *Interface de Integração* que gerasse os arquivos necessários para a execução das primitivas para a aplicação dos padrões utilizando técnicas de *refactoring* a partir de modelos de padrões no formato XMI. A etapa de *Conversão* seria a responsável por mapear os elementos de modelagem do padrão em *refactorings* simples, tais como: criar classes (*AddClass*), renomear métodos (*RenameMethod*), entre outros.

4.6.2 Etapa 2: Preparação

Esta etapa é executada pelo componente *Gerador de Configuração* da *Interface de Integração* e consiste em gerar ou atualizar os demais arquivos necessários para a criação do componente de extensão da ferramenta de desenvolvimento escolhida. Até este momento foram gerados os arquivos referentes à inclusão propriamente dita dos elementos de modelagem na ferramenta, resta agora gerar os arquivos que chamamos de ***Arquivos de***

Configuração, que são: arquivos de recursos, arquivos de inicialização, arquivos de propriedades, arquivos de configuração de menus, entre outros.

Como a *Interface de Integração* funciona para inclusão de novos modelos de padrões por demanda, os arquivos gerados nesta etapa podem ser a atualização dos *Arquivos de Configuração* já existentes. Neste caso, é necessária a busca por arquivos já existentes para a atualização. A decisão por gerar ou atualizar os arquivos de configuração é ilustrada na Figura 17, onde o fluxo das atividades na etapa 2 segue pela atividade ***Geração dos Arquivos de Configuração*** se ainda não existem modelos de padrões incluídos na ferramenta, ou para a atividade ***Atualização dos Arquivos de Configuração*** se já existem modelos incluídos na ferramentas pela *Interface de Integração*. No caso do IIMPaR, ferramenta desenvolvida nesta dissertação (ver Capítulo 5), a etapa de *Preparação* é responsável por gerar ou atualizar os arquivos que configuram os menus no Rational Rose para a aplicação dos modelos através da chamada aos *scripts* gerados na etapa de *Conversão*.

4.6.3 Etapa 3: Ativação

A etapa de ***Ativação*** é executada pelo componente *Ativador* da *Interface de Integração* e é responsável por configurar e personalizar a ferramenta instalando os arquivos gerados nas duas primeiras etapas (*Conversão* e *Preparação*) e configurando o que for necessário no sistema operacional (respectivamente, atividades ***Instalação de Arquivos*** e ***Configuração do SO***). Esta etapa deve promover a instalação e/ou atualização dos componentes de extensão gerados para a ferramenta de desenvolvimento. É a concretização funcional do processo.

Da mesma forma que o ambiente de trabalho da ferramenta de desenvolvimento foi alterado para suportar a aplicação dos padrões, a *Interface de Integração* deve executar também o procedimento inverso, responsável por desinstalar os componentes de extensão gerados, restaurando a configuração inicial do ambiente (ferramenta e sistema operacional), para o caso do desenvolvedor optar por não possuir mais os recursos disponíveis na ferramenta.

4.7 Conclusão

Neste capítulo apresentamos um processo para a integração de modelos de padrões de *software* com ferramentas de desenvolvimento que consiste em uma nova proposta para a reutilização de padrões no desenvolvimento de sistemas a partir de modelos de padrões no formato XMI.

O processo de integração proposto é aberto o suficiente para a implementação com outros tipos de ferramentas de desenvolvimento que não sejam necessariamente ferramentas de modelagem de dados na linguagem UML, como: ferramentas de geração de código, ferramentas de *refactoring* e outros ambientes de desenvolvimento.

No capítulo seguinte, apresentamos o estudo de caso desta dissertação, que consiste na implementação do processo de integração para a ferramenta de modelagem Rational Rose, denominada IIMPaR (*Interface de Integração de Modelos de Padrões de Software para o Rose*).

Capítulo 5
IIMPaR – Uma Interface de Integração de Modelos de Padrões de Software para o Rational Rose

Este capítulo apresenta IIMPaR (*Uma Interface de Integração de Modelos de Padrões de Software para o Rose*). A Seção 5.2 descreve o Rational Rose e os seus recursos de extensão. A Seção 5.3 define a estrutura e o funcionamento da IIMPaR, detalhando as atividades realizadas na implementação do processo de integração. A Seção 5.4 apresenta um resumo sobre a utilização da ferramenta. A Seção 5.5 cita algumas limitações da implementação. Finalmente, a Seção 5.5 traz a conclusão do capítulo.

5.1 Introdução

No capítulo anterior foi apresentado um processo para a integração de modelos de padrões de *software* com ferramentas de apoio ao desenvolvimento de sistemas que consiste em uma nova proposta para a reutilização de modelos de padrões de *software* em ferramentas de desenvolvimento. O processo é definido através da descrição das atividades executadas nas suas três etapas (*Conversão, Preparação* e *Ativação*).

O processo de integração proposto é aberto o suficiente para a implementação com diferentes tipos de ferramentas de desenvolvimento. Como estudo de caso foi escolhida a ferramenta de modelagem de sistemas Rational Rose [68], que tem sido usado com sucesso no desenvolvimento de sistemas. Apesar de ser uma ferramenta comercial, o Rational Rose fornece recursos para a extensão de suas funcionalidades. Para isso, foi implementada uma *Interface de Integração de Modelos de Padrões de Software para o Rose* denominada IIMPaR, que é responsável por receber os modelos de padrões no formato XMI, gerar e instalar os componentes de extensão do Rose para a aplicação automática dos modelos. Dessa forma, a IIMPaR proporciona ao desenvolvedor a oportunidade de estar criando uma biblioteca particular de padrões dentro do seu ambiente de desenvolvimento, de forma que estes modelos de padrões sempre possam ser reutilizados em seus projetos.

Neste capítulo são discutidas as particularidades das etapas do processo de integração aplicado ao Rose. Na etapa de *Conversão* são gerados os *scripts* que executam as funções para a inserção dos modelos no Rose. Esses *scripts* precisam ser executados no Rose de maneira transparente para o usuário. Para isso, outros arquivos são gerados para a personalização do Rose, de maneira que os padrões possam ser aplicados através da seleção de opções no próprio ambiente da ferramenta. Esses arquivos são gerados na etapa de *Preparação*. Para que esses componentes de extensão possam ser utilizados no Rose é necessário que os arquivos gerados nas etapas de *Conversão* e *Preparação* sejam instalados e que alguns recursos do sistema operacional sejam configurados. A instalação destes arquivos e a configuração de qualquer recurso adicional são executadas durante a etapa de *Ativação*.

5.2 Rational Rose

O Rational Rose é uma ferramenta CASE que suporta a análise e o projeto de sistemas orientados a objetos através da modelagem de dados na linguagem UML (ver Seção 4.2). O Rose também fornece recursos para geração de código e engenharia reversa de sistemas em várias linguagens de programação, integração com bancos de dados, entre outros. Além disso, outros desenvolvedores implementam *Add-Ins*[7] para a extensão das funcionalidades do Rose.

Existem várias ferramentas de modelagem que oferecem suporte para a aplicação de padrões de *software*. O Rose oferece recursos para a aplicação automática de alguns padrões da *Gang of Four* [33]. Porém, estes recursos não são extensíveis para o uso de outros padrões. Através de diferentes abordagens, trabalhos propõem estender a ferramenta ampliando os seus recursos para a aplicação de padrões. Em [29] é sugerido um esquema para a criação de *scripts* para a inserção de um novo elemento de modelagem (*Pattern*[8]) que suporta a especificação de modelos de padrões no Rose, propondo um ambiente de modelagem orientado a padrões. Nele o desenvolvedor pode especificar e combinar as classes de um padrão e o relacionamento entre elas, diferente da ferramenta desenvolvida no nosso trabalho que consiste em utilizar os recursos de extensão do Rose para aplicar padrões de uma

[7] *Add-In* é o nome dado para componentes de extensão de ferramentas que podem ser adicionados ou removidos do ambiente conforme a necessidade. Os recursos do Rose para a criação de *Add-Ins* são apresentados na Seção 5.2.1

[8] No trabalho referenciado, *Pattern* é um elemento que está sendo adicionado ao ambiente de modelagem, assim como *Package*, *Class* e *Interface*.

biblioteca de modelos de padrões gerada pelo usuário a partir de modelos no formato XMI. A estrutura da interface de extensão do Rose é descrita na subseção seguinte.

5.2.1 Rational Rose Extensibility

O Rational Rose possui uma interface de extensão chamada *Rose Extensibility Interface* (REI) [75], que permite a extensão e personalização de suas capacidades para atender necessidades de desenvolvimento específicas. As seguintes extensões são permitidas:

- Personalizar os menus do Rose;
- Automatizar funções manuais do Rose através de *scripts* (i.e., criação de diagramas, criação de classes, geração de documentação);
- Executar funções do Rational Rose dentro de outras aplicações usando o objeto ***Rational Rose Automation*** (***RoseApp***);
- Acessar classes, propriedades e métodos do Rational Rose dentro de seu ambiente de desenvolvimento de software (ex.: Visual Basic, Visual C++) incluindo o ***Rational Rose Extensibility Type Library*** no seu ambiente;
- Ativar o ***Rational Rose Add-ins*** usando o ***Add-In Manager.***

A Figura 21 mostra os componentes da *Rose Extensibility Inteface* e da *Rose Automation* e ilustra os relacionamentos entre eles. Estes componentes são descritos a seguir:

- ***Rational Rose Application*** – objetos do *Rose Extensibility* que fazem a interface com as funcionalidades da aplicação do Rational Rose;
- ***Diagrams*** – objetos do *Rose Extensibility* que fazem a interface com os diagramas e visões do Rational Rose;
- ***Model Elements*** – objetos do *Rose Extensibility* que fazem interface com os elementos do modelo do Rational Rose;
- ***Rose Extensibility Interface*** – um conjunto comum de interfaces usadas pelo *Rose Script* e *Rose Automation* para acessar o Rational Rose;

- ***Rose Script*** – um conjunto de objetos que permite automatizar funções do Rational Rose utilizando *Rose Scripts*[9]. A linguagem de *script* utilizada é uma versão estendida da linguagem *Summit BasicScript*;
- ***Rational Rose Automation*** – um conjunto de objetos do Rational Rose para funcionar como um controlador ou servidor de automação OLE.

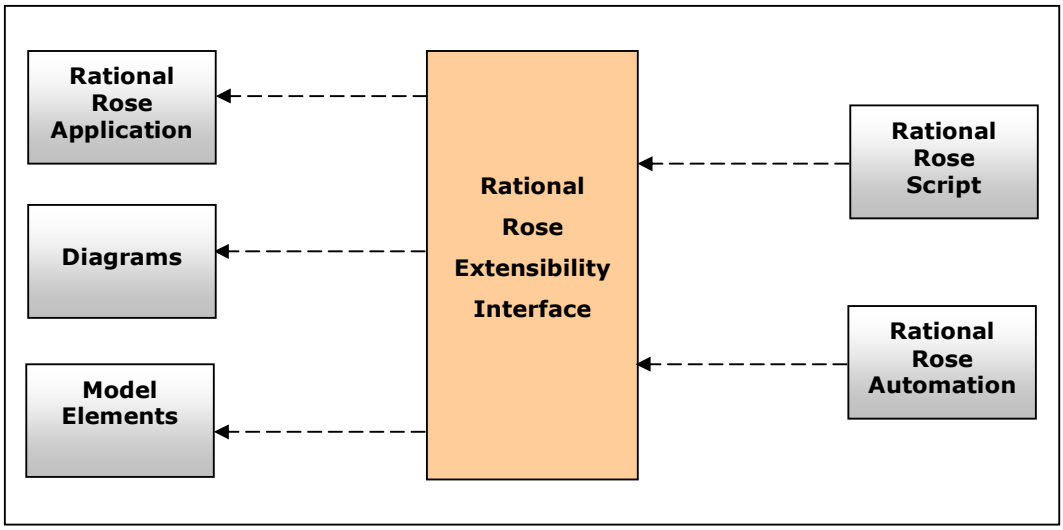

Figura 21 - Componentes do *Rational Rose Automation* e do *Rose Extensibility Interface*

Utilizando os recursos oferecidos pela REI, a *Interface de Integração* desenvolvida no nosso estudo de caso, utiliza a linguagem de *script* do Rose para a manipulação dos objetos da *Rose Application*, *Diagrams* e *Model Elements* (descritos anteriormente) com o intuito de aplicar modelos de padrões de software. Detalhes desta *Interface de Integração* são apresentados na próxima subseção.

5.3 A Interface de Integração IIMPaR

A IIMPaR (*Interface de Integração de Modelos de Padrões de Software para o Rose*) é a implementação do processo de integração de modelos de padrões com ferramentas de desenvolvimento, proposto por esta dissertação para o Rational Rose.

A Figura 22 mostra a estrutura dos componentes do IIMPaR, bem como os insumos e produtos necessários para a geração do componente de extensão que interage com o Rose através da *Rose Extensibility Interface* (REI) (ver Seção 5.2.1).

[9] O termo "*Rose Script"* é utilizado para fazer referência tanto aos *scripts* na linguagem interpretada pelo Rose, como ao conjunto dos objetos e recursos utilizados nesses *scripts*.

Como descrito no capítulo anterior, existe um componente da *Interface de Integração* (*Conversor de Modelos, Gerador de Configuração* e *Ativador*) responsável por cada uma das três etapas do processo de integração proposto. Na seção seguinte são descritas as particularidades das etapas do processo de integração implementadas para a IIMPaR.

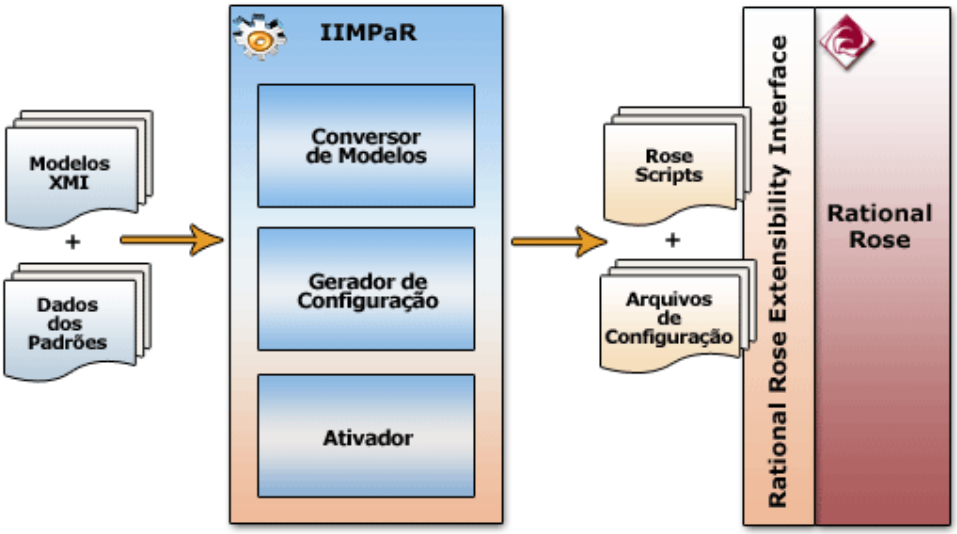

Figura 22 - Estrutura do IIMPaR

5.3.1 Conversão

A etapa de conversão consiste em gerar os arquivos para inserção dos modelos de padrões no Rose a partir dos modelos no formato XMI. O processo de conversão utiliza um processador de transformações XSL e é executado em duas fases (*Simplificação* e *Finalização*), como pode ser visto na Figura 23. A etapa de *Conversão* do processo de integração é apresentada na íntegra no Capítulo 4.

Figura 23 - Processo de conversão dos modelos, baseado em XSLT

Como é mostrado na Seção 4.6.1, a fase de *Simplificação* é responsável pela conversão de um modelo no formato XMI em um outro arquivo XML simplificado que contém somente os elementos de modelagem necessários para o próximo passo. A fase de

Finalização da conversão é onde ocorre a transformação XSL na qual é gerado o *script* que será responsável pela inserção propriamente dita do modelo na ferramenta. Esses *scripts* são chamados de ***Rose Scripts***. Eles são escritos em uma linguagem que é interpretada pelo Rational Rose através dos recursos da REI (ver Seção 5.2.1) e possuem a extensão ".ebs". A Figura 24 ilustra os elementos que participam das duas fases da conversão. *XSLT Docs* são os documentos XSLT que definem as regras de transformações utilizadas nas duas fases e o *Transformador XSL* é o componente que recebe um documento XML e um documento XSLT da transformação, executa a transformação e retorna o documento gerado no formato especificado no XSLT.

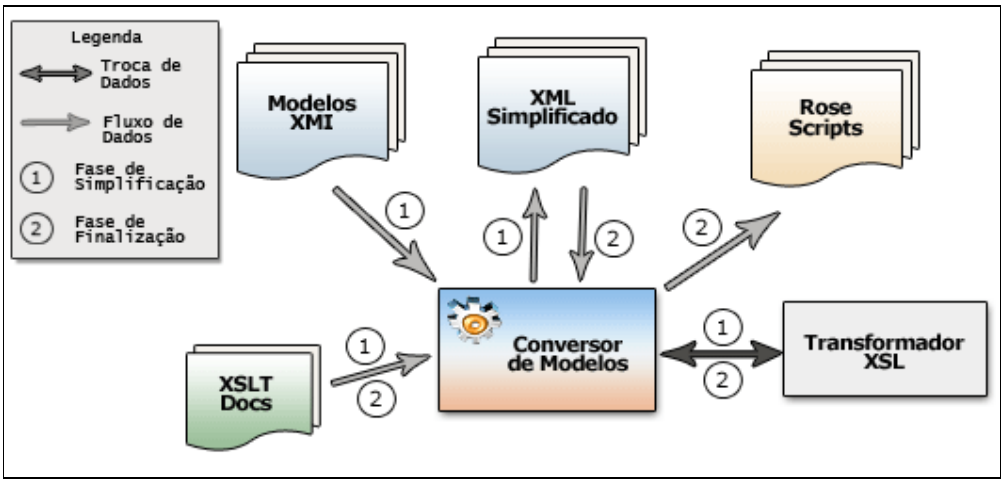

Figura 24 - Participantes do Processo de conversão

Na prática, a conversão consiste em gerar, utilizando a linguagem de *scripts* do Rose, as primitivas necessárias para a aplicação do modelo do padrão. O mapeamento entre os principais elementos de modelagem de XMI para o formato *XML Simplificado* é exposto na Seção 4.6.1.2. A Tabela 10 mostra o mapeamento entre os principais elementos do *XML Simplificado* e as primitivas para inserção de elementos nos *Rose Scripts*.

Como pode ser visto no mapeamento mostrado na Tabela 10, o processo de conversão para o Rose consiste em gerar as diretivas para a inserção dos elementos de modelagem provenientes dos modelos de padrões em XMI. Além disso, os *scripts* devem conter todos os demais comandos necessários para a instanciação dos objetos do ambiente de trabalho do Rose; a manipulação dos objetos para atribuição das propriedades dos elementos citados na Tabela 10 (i.e., classes, associações, métodos, atributos); e a criação de caixas de diálogos

para o processo de aplicação dos padrões. Um exemplo de *Rose Script* gerado para a inclusão de um padrão é mostrado no Apêndice C.

Elemento	XML simplificado	Rose Script
Classe	`<class id="">` `</class>`	`theCategory.AddClass(ClassName)`
Atributo	`<attribute>` `</attribute>`	`theClass.AddAttribute(AttrName, AttrType, InitValue)`
Operação	`<operation id="">` `</operation>`	`theClass.AddOperation(OperationName, OperationType)`
Parâmetro	`<parameter id="">` `</parameter>`	`theObject.AddParameter(ParameterName, ParameterType, InitValue, Position)`
Associação	`<association id="">` `</association>`	`theClass.AddAssociation(theSupplierRoleName, theSupplierName)`
Generalização	`<generalization>` `</generalization>`	`theClass.AddInheritRel(theName, theParentClassName)`

Tabela 10 - Mapeamento entre elementos do *XML Simplificado* e diretivas do Rose Script

Na atual abordagem utilizada, os arquivos gerados para a aplicação dos modelos de padrões no Rose, quando acionados, atuam inserindo os elementos dos modelos (i.e., classes, atributos, relacionamentos, entre outros) no ambiente de trabalho do Rose, mais precisamente na visão lógica do ambiente (*Logical View*). Essa inserção é feita de maneira *estática* (ver Seção 4.4). Para que os modelos de padrões fossem aplicados de forma *dinâmica*, eles deveriam passar por processo de detecção dos pontos de flexibilização, também chamados de *hot-spots*. Dessa forma, os modelos de padrões a serem aplicados funcionariam como *frameworks* para a aplicação desses padrões.

Além do *script* de inserção dos elementos, são gerados outros arquivos necessários para a personalização da ferramenta, discutidos na próxima subseção.

5.3.2 Preparação e Ativação

Como mencionado na Seção 5.2.1, através da REI, o Rational Rose permite a extensão e personalização de suas capacidades, de forma a atender a necessidades de desenvolvimento específicas, como personalização de menus, automatização de funções manuais do Rose através de *scripts* e execução de funções do Rose dentro de outras aplicações.

O uso desses recursos torna possível a criação dos elementos necessários para a inserção dos modelos de padrões na ferramenta. Como visto na Seção anterior, o *script* de inserção do modelo gerado na conversão é o principal desses elementos. Além deste *script*,

são utilizados outros arquivos, como aqueles para a personalização dos menus (.mnu) e os de inicialização (.ini). A geração desses arquivos é realizada durante a etapa de *Preparação* e é de responsabilidade do componente *Gerador de Configuração* do IIMPaR, ilustrado na Figura 22.

Tanto o *Gerador de Configuração* quanto o *Ativador* (responsável pela etapa de *Ativação*, conforme Seção 4.6.3) atuam por demanda, atualizando os componentes de extensão gerados a cada inclusão de um novo modelo de padrão. O *Gerador de Configuração* da IIMPaR tem a responsabilidade de verificar a existência de arquivos de configuração e atualizá-los. Para cada modelo de padrão adicionado como novo recurso do Rose, existe um *script* de aplicação gerado na etapa de *Conversão*. Esses arquivos não precisam ser atualizados, eles são individuais de cada padrão. No entanto, o *script* de extensão do menu do Rose deve ser atualizado a cada adição de um novo padrão, pois ele contém o atalho para a execução dos *scripts* de aplicação de cada padrão.

Um exemplo de um *script* de extensão das opções de menu do rose para a aplicação de padrões é ilustrado na Figura 25.

```
                    - arquivo iimpar.mnu -
Menu Tools
{
  Separator
  Menu "IIMPaR - Padrões"
  {
    Menu "Padrões da Gang of Four"
    {
       option "Command"
       {
           RoseScript $IIMPAR_PATH\scripts\command.ebs
       }
       option "Strategy"
       {
           RoseScript $IIMPAR_PATH\scripts\strategy.ebs
       }
    }
  }
}
```

Figura 25 - Exemplo de um arquivo de extensão de menus do Rose

No exemplo é mostrado um *script* que contém as opções de menu adicionadas para a aplicação dos padrões de projeto *Command* e *Strategy* [33]. A estrutura do *script* funciona como uma árvore de menus, submenus e opções que disparam ações como a execução dos *scripts* de inserção de modelos que são gerados na etapa de *Conversão* (ver Seção 5.3.1). Em

geral, os *Add-Ins* gerados para o Rose criam opções de menu inseridas na opção "*Tools*" do menu principal.

A disponibilização desses componentes de extensão através da instalação dos *scripts* gerados, instalação dos arquivos de configuração, definição de variáveis de ambiente e configuração do sistema operacional, são realizadas durante a etapa de *Ativação* pelo componente *Ativador* do IIMPaR (ver Figura 22), que atua como um instalador e/ou assistente de atualização do componente de extensão para o caso de inserção de novos modelos no Rose. A Figura 26 mostra um exemplo da interface gráfica do Rose personalizada através da instalação dos componentes de extensão gerados e instalados pelo IIMPaR.

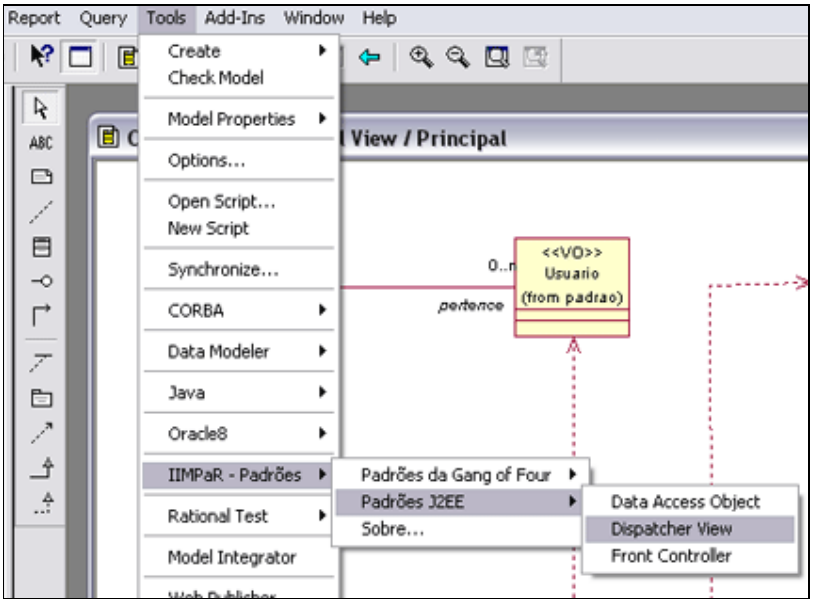

Figura 26 - Interface gráfica do Rose estendida pelo IIMPaR

5.4 Utilização da IIMPaR

Todo o processo de integração é executado na IIMPaR através de uma interface *wizard* que conduz o usuário através de passos, desde a aquisição dos arquivos até a confirmação da inclusão do modelo de padrão. Um exemplo da interface gráfica da IIMPaR durante o processo *wizard* é ilustrado na Figura 27. A execução total do processo acontece nos cinco passos descritos a seguir:

- **Passo 1: Início do Processo** – passo informativo onde o usuário inicia a execução do processo.

- **Passo 2: Localização do Modelo** – passo onde o usuário deve informar ao sistema a localização referente ao modelo do padrão (arquivo XMI ou ZIP) (ver Seção 4.6.1.1).
- **Passo 3: Confirmação dos dados** – passo onde o usuário informa e confirma os dados adicionais do padrão (i.e., nome do padrão, nome da coleção) se necessário.
- **Passo 4: Ativação da interface do Rose** – passo onde o usuário visualiza e confirma como ficará a nova estrutura das opções de menu do Rose, com a inserção da opção para a aplicação do novo modelo de padrão.
- **Passo 5: Fim do processo** – passo onde o usuário é informado de que o processo foi executado com sucesso ou se ocorreu alguma falha durante a execução.

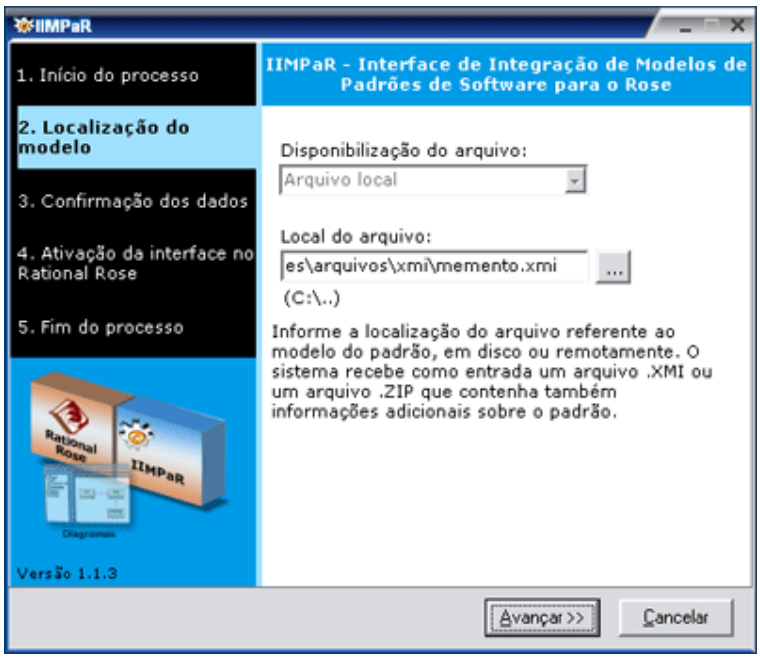

Figura 27 – Interface gráfica da IIMPaR durante o processo de *wizard* para a inclusão de um padrão

5.5 Discussão

A IIMPaR foi implementada para validar o processo de integração proposto. Em relação à execução das ações necessárias para a atingir o objetivo desta dissertação na reutilização de modelos de padrões de *software* em ferramentas de desenvolvimento, a IIMPaR obteve resultados satisfatórios apresentados no decorrer deste capítulo. Contudo, algumas limitações da implementação, e não do processo, são apontadas:

- Inserção estática dos padrões - um ponto fraco da implementação da IIMPaR é o fato de o mecanismo para a aplicação dos padrões funcionar de maneira *estática* (ver Seção 4.4), não permitindo que os elementos do padrão aplicado venham a ser criados a partir de elementos de uma modelagem já existente, o que gera um dos trabalhos futuros em relação a esta implementação, que é a criação de um mecanismo de parametrização dos modelos de padrões a partir da definição de pontos de flexibilização.
- Restrição ao uso de XMI na Versão 1.0 - A IIMPaR atualmente trabalha com arquivos XMI na versão 1.0 e até o presente momento, o XMI já está na versão 2.0. A atualização da IIMPaR para a inclusão do suporte à novas versões do XMI pode ser realizada através da atualização do documento XSLT responsável pela transformação executada na fase de *Simplificação* da *Conversão*.
- Limitação ao ambiente Windows - A ferramenta (IIMPaR) foi implementada utilizando uma linguagem de programação para o ambiente Windows. Entretanto, é interessante criar uma versão multiplataforma.

5.6 Conclusão

Neste capítulo apresentamos a *Interface de Integração de Modelos de Padrões de Software para o Rose* (IIMPaR), desenvolvida seguindo o processo de integração de modelos de padrões proposto por esta dissertação e apresentado no Capítulo 4. A IIMPaR fornece os recursos necessários à extensão das funcionalidades do Rational Rose para a aplicação de padrões de *software* a partir de modelos de padrões no formato XMI. Além disso, este processo permite ao desenvolvedor gerar dessa forma uma biblioteca de padrões dentro do Rational Rose.

O capítulo seguinte apresenta os resultados alcançados durante a elaboração desta dissertação e as possibilidades para a sua continuação em trabalhos futuros.

Capítulo 6
Conclusão

Este capítulo resume as principais conclusões apresentadas em cada capítulo, destacando na Seção 6.1 as principais contribuições obtidas durante o desenvolvimento da dissertação e as possibilidades para trabalhos futuros na Seção 6.2.

6.1 Resultados Alcançados

Com o objetivo de suprir a carência por mecanismos que auxiliem no armazenamento e disponibilização de padrões de *software*, foi proposto o desenvolvimento de um repositório de padrões de *software* no Capítulo 3, que suportasse o armazenamento de diferentes tipos e formatos de padrões de *software* existentes na literatura, visando preservar o conteúdo original e facilitar o processo de classificação e busca desses padrões.

Contudo, a maior contribuição deste trabalho foi a proposta de uma solução para a integração de modelos de padrões de *software* com ferramentas de apoio ao desenvolvimento de sistemas através da especificação de um processo de integração (ver Capítulo 4) que sugere uma forma de reutilizar modelos de padrões de *software* em um formato aberto (o XMI), de maneira integrada e por demanda em diferentes tipos de ferramentas de desenvolvimento.

Para a validação do processo proposto, foi implementada a IIMPaR, uma interface de integração de modelos de padrões de *software* para o Rational Rose, o estudo de caso desta dissertação introduzido no Capítulo 5. Essa interface é responsável por receber modelos de padrões no formato XMI e gerar os componentes de extensão necessários para a aplicação desses modelos no Rose. Os modelos XMI de padrões podem ser obtidos de qualquer fonte, inclusive do repositório de padrões de *software*.

Além disso, logo nos primeiros meses do programa de mestrado foi iniciado um trabalho de investigação para um maior aprofundamento na área de padrões de *software*. Foram apresentados um seminário de *Padrões de Software* [76] e um seminário de *Padrões de Projeto para Sistemas Web* [77] como trabalhos de disciplinas. O primeiro foi elaborado a partir de um estudo feito dos conceitos de padrões apresentados em [5] e de suas principais referências. O segundo foi elaborado a partir de um estudo sobre a dissertação de mestrado apresentada em [83]. Estes trabalhos resultaram em capacitação de estudantes de graduação e

pós-graduação, o que é importante para a formação de uma base maior de conhecimento sobre a área de pesquisa.

Nesse contexto de capacitação de recursos humanos, é importante ressaltar o projeto de colaboração com o IA [7], onde também foram apresentados seminários sobre padrões de domínio específico como o *Seminário de Padrões Web* [78], *Seminário de Padrões de Segurança* [79] e o *Seminário de Padrões para Interfaces Gráficas* [80]. A partir das entrevistas sobre o reuso de padrões, realizadas neste mesmo projeto e mencionadas na Seção 2.1 do Capítulo 2, foi confirmado o fato de o reuso de padrões ser uma boa prática no desenvolvimento de sistemas, aumentando a produtividade, diminuindo os custos e aumentando a integração entre os membros de um mesmo projeto e até de diferentes projetos.

Tivemos a oportunidade de publicar o artigo intitulado "*Uma Proposta de Um Repositório de Padrões de Software Integrado ao RUP*" [6] no SPA *(Session Pattern Application)* do *SugarLoafPLoP 2003*, Terceira Conferência Latino-americana de Linguagens de Padrões para Programação. O artigo é referente à eficácia do uso do repositório de padrões de *software* em fases específicas do processo interativo e incremental de desenvolvimento de sistemas sugeridas pelo RUP.

6.2 Trabalhos Futuros

As possíveis linhas de estudo para a continuação ou aprimoramento deste trabalho são apresentadas a seguir:

- ***Construir Frameworks para a implementação do processo de integração proposto para categorias específicas de ferramentas de apoio ao desenvolvimento de sistemas***

  Como é citado no Capítulo 4, o processo de integração é aberto o suficiente para a implementação com outros tipos de ferramentas que não sejam necessariamente ferramentas de modelagem UML. Para facilitar a implementação do processo para categorias específicas de ferramentas (i.e., ferramentas de modelagem, ferramentas para geração automática de código, ferramentas para *refactoring*) seria útil a criação de *frameworks* que atendessem aos critérios específicos de cada uma.

- ***Estudar um mecanismo de auxílio à parametrização (detecção de pontos de flexibilização) de modelos de padrões***

Existem duas formas de aplicar padrões automaticamente através do uso de ferramentas, a inserção *estática* e a *dinâmica*, como mencionado no Capítulo 4. O processo proposto nesta dissertação atualmente trabalha com a inserção *estática* dos modelos.

Para que o processo proposto promovesse a inserção *dinâmica* dos modelos de padrões, seria necessário um trabalho de detecção dos pontos de flexibilização (*hot-spots)* dos modelos de padrões, que seriam os elementos da modelagem do padrão (i.e, classes, operações, entre outros) que designam papéis específicos no cenário no qual ele está sendo aplicado.

Uma maneira de automatizar também a detecção de *hot-spots* dos modelos de padrões, seria a criação de um mecanismo que auxiliasse o usuário nesse processo, expondo os elementos da modelagem para a identificação dos pontos de flexibilização, a fim de gerar modelos parametrizados. Este processo poderia culminar na construção de *frameworks* individuais para cada modelo de padrão.

- ***Elaborar mecanismos de busca semântica em arquivos XMI***

Usuários, a fim de aplicar padrões, podem conhecer elementos da modelagem dos padrões nos quais está interessado e querer realizar uma busca por esses elementos e não só pela documentação do padrão como é feito atualmente. Poderia ser elaborado um mecanismo de busca semântica que fosse capaz de gerar consultas sobre os elementos dos arquivos XMI, utilizando linguagens de consulta para XML como XPath [98] e XQuery [99]. Esse mecanismo de busca sobre arquivos XMI poderia ser incorporado ao repositório de padrões proposto nesta dissertação para aumentar as possibilidades de busca de padrões já existentes nos mecanismos de busca atuais que trabalham sobre a documentação dos padrões.

- ***Estudar o uso dos tipos de relacionamentos entre padrões como recurso auxiliar para os mecanismos de busca do repositório***

Os tipos de relacionamentos poderiam ser mais um artifício útil na busca por padrões. O usuário poderia estar interessado em saber, por exemplo, que padrões "*especializam*" o padrão "*Strategy*". Atualmente, não existe uma maneira de se obter esse tipo de informação a partir do repositório de padrões proposto nesta dissertação ou em outros trabalhos.

- ***Estudar mecanismos para a visualização dos padrões e seus relacionamentos em forma de grafos***

O usuário poderia visualizar mapas de linguagens de padrões ou quaisquer grupos de padrões relacionados cadastrados no repositório. A visualização poderia ser feita através da estrutura de um grafo onde os vértices seriam os padrões e as arestas seriam os relacionamentos entre eles. Os níveis de profundidade do grafo, bem como os tipos de relacionamentos de interesse, seriam definidos pelo usuário. Existem API's Java que geram a visualização de grafos bidimensionais e tridimensionais, e que permitem a manipulação dos seus elementos graficamente [28][50].

A visualização e navegação sobre um mapa de uma linguagem de padrões ou de quaisquer padrões relacionados poderiam agilizar o processo de busca por padrões alternativos ou complementares dentro do repositório de padrões de *software* proposto.

- ***Automatizar o processo de integração entre o repositório de padrões de software e a ferramenta IIMPaR propostos nesta dissertação***

Atualmente, a ferramenta IIMPaR recebe como entrada modelos de *software* no formato XMI, que são adiquiridos manualmente de qualquer fonte de dados, por exemplo, do repositório de padrões proposto nesta dissertação. Este processo de aquisição de arquivos poderia ser realizado de forma automática, através da integração da ferramenta IIMPaR com o mecanismo de busca do repositório, o que permitiria que o usuário utilizasse somente a IIMPaR para buscar e adicionar modelos de padrões à sua ferramenta.

Referências Bibliográficas

[1] AßMANN, U. ***Automatic Roundtrip Engineering.*** In Eletronic Note in Theorical Computer Science, Vol. 82, No. 5, 2003.

[2] ALEXANDER, C. ***The Timeless Way of Building.*** Oxford University Press, New York, NY, 1979.

[3] ALEXANDER, C. et al. ***A Pattern Language****: Towns, Buildings, Construction*. Oxford University Press, New York, NY, 1977.

[4] AMBLER, S. W. ***The Process Patterns Resource Page***. Disponível em: http://www.ambysoft.com/processPatternsPage.html. Acessado em: 20/10/2003.

[5] ANDRADE, R. ***Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems***. Ph.D. Thesis, School of Information Technology and Engineering (SITE), University of Ottawa, Ottawa, Ontario, Canada, May 2001.

[6] ANDRADE, R.; MARINHO, F.; SANTOS, M.; NOGUEIRA, R. ***Uma Proposta de um Repositório de Padrões Integrado ao RUP.*** Session Pattern Application (SPA), *SugarLoafPLoP 2003*, The Third Latin American Conference on Pattern Languages of Programming, Porto de Galinhas, PE, Ago. 2003.

[7] ANDRADE, R.; MARINHO, F. ***Diagnóstico do reuso de padrões de software no Instituto Atlântico – Projeto: Métodos e Técnicas associados a Padrões de Software***. Relatório Técnico do Projeto de Pesquisa e Desenvolvimento em Colaboração Departamento de Computação / Universidade Federal do Ceará e Instituto Atlântico (Convênio CVE/P&D/005/02), Fortaleza, CE, Fev. 2003.

[8] ArgoUML. Disponível em: http://argouml.tigris.org/. Acessado em: 07/12/2003.

[9] BECK, K. ***Smalltalk Best Practice Patterns***. Prentice Hall, Upper Saddle River, NJ, 1997.

[10] BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. ***The Unified Modeling Language User Guide***. Addison-Wesley, 1998.

[11] Borland Together. Disponível em: http://www.borland.com/together. Acessado em: 03/07/2003.

[12] BRAGA, A. M.; RUBIRA, C. M. F.; DAHAB, R., ***Tropyc: A Pattern Language for Cryptographic Software.*** Pattern Languages of Program Design 4 (PLoPD4), N.D.Harrison, B. Foote, and H. Rohnert, eds., Addison-Wesley, 337-371, 2000.

[13] BUDINSKY, F.; FINNIE, M.; VLISSIDES, J.; YU, P. S., ***Automatic Code Generation From Design Patterns.*** IBM Research Journal, Vol. 35, No. 2 , 1996 . Disponível em http://www.research.ibm.com/journal/sj/352/budinsky.html.

[14] BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. ***Pattern-Oriented Software Architecture.*** John Wiley and Sons, New York, NY, 1996.

[15] BULKA, A. ***Design Pattern Automation***. In Proc. Pattern Languages of Programs 2002. Revised papers from the Third Asia-Pacific Conference on Pattern Languages of Programs, (KoalaPLoP 2002), Melbourne, Australia. Conferences in Research and Practice in Information Technology, **13**. Noble, J., Ed. ACS. 1, 2002.

[16] CAMP, D. V. ***The Object-Oriented PatternDigest.*** Disponível em: http://www.patterndigest.com. Acessado em: 12/03/2004.

[17] CHAMBERS, C.; HARRISON, W.; VLISSIDES, J. M. ***A Debate on Language and Tool Support for Design Patterns*****.** In Proceedings In 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Pages: 277 – 289, ACM Press, 2000.

[18] Code Farms. Disponível em: http://www.codefarms.com/ptl.htm. Acessado em: 30 de abril de 2003.

[19] CodePro Studio. Disponível em: http://www.instantiations.com/codepro/ws/docs/default.htm. Acessado em: 10/06/2003.

[20] CONTE, A.; FREIRE, J.; GIRAUDIN, J.; HASSINE, I.; RIEU, D. ***A Tool and A Formalism to Design and Apply Patterns.*** SugarLoafPLoP 2002

[21] COPLIEN, J. O. ***Software Patterns.*** SIGS books and Multimedia, June 1996.

[22] COPLIEN, J. O. ***C++ Idioms Patterns.*** In Brian Foote, Neil Harrison, and Hans Rohnert, editors, Pattern Languages of Program Design 4, chapter 10, 167-197. Addison Wesley, Reading, MA, 2000.

[23] COPLIEN, J. O. ***A Development Process Generative Pattern Language***. in Pattern Languages of Program Design, J. O. Coplien and Douglas C. Schmidt, Ed. Reading, MA: Addison-Wesley, 1995, pp. 183-237.

[24] Core J2EE Pattern Catalog. Disponível em: http://java.sun.com/blueprints/corej2eepatterns. Acessado em: 10/04/2003.

[25] CYSNEIROS FILHO, G. A. A. ***Ferramenta para o Suporte do Mapeamento da Modelagem Organizacional em i* para UML***. 98 p, Dissertação (Mestrado em Ciência da Computação), Universidade Federal de Pernambuco, Recife, 2001.

[26] DPT-Tool. University of Technology Munich/Departement of CS (Informatik), Disponível em: http://dpt.kupin.de. Acessado em: 03/07/2003.

[27] Eclipse Project. Disponível em: http://www.instantiations.com/codepro/ws/docs/default.htm. Acessado em: 02/07/2003.

[28] ERTEN, C.; KOBOUROV, S. Le, V.; NAVABI, A. ***Simultaneous Graph Drawing: Layout Algorithms and Visualization Schemes***. Disponível em: http://simg.cs.arizona.edu/index.html. Acessado em: 03/07/2004.

[29] FORBRIG P.; LAEMMEL, R.; MANNHAUPT, D. ***Pattern-Oriented Development with Rational Rose.*** The Rational Edge, Jan, 2001.

[30] FOWLER, M. ***Refactoring Home Page***. Disponível em: http://www.refactoring.com. Acessado em: 10/10/03.

[31] FOWLER, M. ***Analysis Patterns: Reusable Object Models***. Addison-Wesley, Reading, MA, 1997.

[32] FOWLER, M. ***Implementing Analysis Patterns***. presentation at OOP'97, 1997.

[33] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. ***Design Patterns: Elements of Reusable Object-Oriented Software***. Addison-Wesley, Reading, MA, 1995.

[34] GERBER, L. D. ***Uma Linguagem de Padrões para o Desenvolvimento de Sistemas de Apoio à Decisão Baseado em Frameworks.*** 145 f, 1999, Dissertação (Mestrado em Informática) - Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 1999.

[35] HANMER, R. ***Introduction to Pattern Languages***. SugarLoafPLoP 2003, The Third Latin American Conference on Pattern Languages of Programming, Porto de Galinhas, PE, 2003.

[36] HENRIKSSON, A.; LARSSON, H. ***A Definition of Round-trip Engineering***., Technical report (not published), 2003. Disponível em: http://www.ida.liu.se/~henla/papers/roundtrip-engineering.pdf. Acessado em: 23/10/03.

[37] Hypermedia Design Patterns Repository, Disponível em: http://www.designpattern.lu.unisi.ch/index.htm. Acessado em: 17/04/2004.

[38] IBM Patterns for e-business. Disponível em: http://www-106.ibm.com/developerworks/patterns. Acessado em: 03/05/2004.

[39] IBM Websphere Studio. Disponível em: http://www-3.ibm.com/software/info1/websphere. Acessado em: 02/07/2003.

[40] IntelliJ IDEA. Disponível em: http://www.intellij.com/idea. Acessado em: 30 de abril de 2003.

[41] Java BluePrints Patterns Catalog. Disponível em: http://java.sun.com/blueprints/patterns/catalog.html. Acessado em: 30/04/2003.

[42] Java Pattern Wizard. Disponível em: <http://www.instantiations.com/codepro/default.htm>.

[43] JENSEN, D. ***Butterfly Code Generation Project***, Sourceforge.net, Disponível em: http://sourceforge.net/projects/butterflycode. Acessado em: 10/02/2004.

[44] JOHNSON, R.; FOOTE, B. ***Designing Reusable Classes***. Journal of Object-Oriented Programming. SIGS, 1, 5 (June/July. 1988), 22-35. Disponível em: http://www-lifia.info.unlp.edu.ar/poo2001/DRC.pdf.

[45] JRefactory. Disponível em: <http://jrefactory.sourceforge.net/csrefactory.html>

[46] KELLY, K. E. ***Singleton Pattern Script***, Rational Software Corporation, 1998. Disponível em: http://www-106.ibm.com/developerworks/rational/library/content/03July/2500/2834/Rose/rational_rose.html.

[47] KONRAD, S.; CHENG, B. H. C. ***Requirements patterns for embedded systems***. In Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE02), Essen, Germany, September 2002.

[48] KROTH, E. ***Aplicação de Padrões Usando Componentes***. IX Escola Regional de Informática. 2001. Disponível em: http://inf.upf.tche.br/eri/material/eduardo_kroth.pdf. Acessado em: 30/04/2003.

[49] KRUCHTEN, P. The Rational Unified Process, An Introduction. Addison-Wesley, 1999.

[50] Large Graphic Layout. Disponível em: http://bioinformatics.icmb.utexas.edu/lgl. Acessado em: 03/07/2004.

[51] LARMAN, C. **Applying UML and Patterns**: An Introduction to Object-Oriented Analysis and Design. 2nd ed., Prentice Hall PTR, 2002.

[52] LISBOA FILHO, J.; PEREIRA, M. A. ***Desenvolvimento de uma ferramenta CASE para o modelo UML-GeoFrame com suporte para padrões de análise.*** In: IV Simpósio Brasileiro de Geoinformática, 2002, Caxambú-MG. **Anais**. Caxambú: SBC, 2002. p.147 - 154

[53] Lock Pattern, Disponível em: http://c2.com/cgi/wiki?LockPattern. Acessado em: 03/07/2004.

[54] MESZAROS, G.; DOBLE, J. ***A Pattern Language for Pattern Writing***. Pattern Language of Program Design 3, edited by Robert C. Martin, Dirk Riehle, and Frank Buschmann, Addison-Wesley (Software Patterns Series), 1997.

[55] Microsoft SQL Server. Disponível em: http://www.microsoft.com/sql. Acessado em: 22/04/2004.

[56] ModelMaker. Disponível em: http://www.modelmakertools.com. Acessado em: 03/07/2003.

[57] MOSCHOYIANNIS, S. ***Visualising and Documenting Distributed Systems Using UML***. University of Surrey, Feb, 2004.

[58] Object Management Group. **OMG** ***Meta Object Facility (MOF) Specification***. Version 1.4, Apr 2002. Disponível em: http://www.omg.org. Acessado em: 20/09/2003.

[59] Object Management Group. ***OMG Unified Modeling Language (UML) Specification***. Version 1.4, Set 2001. Disponível em: http://www.omg.org. Acessado em: 20/04/2004.

[60] Object Management Group. ***OMG XML Metadata Interchange (XMI) Specification.*** Version 1.0, 2000. Disponível em: http://www.omg.org.

[61] OmniBuilder. Disponível em: http://www.omnibuilder.com.

[62] Oracle Database. Disponível em: http://www.oracle.com/database. Acessão em: 22/04/2004.

[63] Pattern Languages of Programs. Disponível em: http://jerry.cs.uiuc.edu/~plop.

[64] phpPatterns. Disponível em: http://www.phppatterns.com. Acessado em: 11/02/2004.

[65] Portland Patterns Repository. Disponível em: http://c2.com/ppr/index.html. Acessado em: 17/04/2004.

[66] PREE, W. ***Design Patterns for Object-Oriented Software Development***. Addison-Wesley Publishing Company, ACM Press, 1995.

[67] Projeto Odissey. Disponível em: http://www.cos.ufrj.br/~odyssey. Acessado em: 24/09/2003.

[68] Rational Software. ***Rational Rose***. Disponível em: http://www-306.ibm.com/software/rational. Acessado em: 12/03/2004.

[69] Rational Software. **IBM Rational XDE Developer v2003.06.12** - Java Platform Edition Evaluators Guide. 2003.

[70] RIBEIRO, A. C.; FLORENTINI, A. C. ***O Padrão XMI****: Uma Proposta para sua Utilização em Bibliotecas Digitais.* Monografia, Universidade Federal do Rio de Janeiro, Junho, 2000.

[71] RISING, L. ***The Pattern Almanac 2000***. Software Pattern Series, Addison-Wesley, 2000. ISBN 0-201-61567-3.

[72] RISING, L. ***Patterns****: A Way to Reuse Expertise.* In: IEEE Communications Magazine, Vol. 37, No. 4, April 1999.

[73] ROBERTSON, S. ***Requirements Patterns Via Events/Use Cases***. The Atlantic Systems Guild, 1996. Disponível em: <http://www.systemsguild.com/GuildSite/SQR/Requirements_Patterns.html> Acessado em: 10/02/2004.

[74] RODRIGUES, A. R. ***Um Componente de Software para o Mapeamento Objeto-relacional. Monografia***, Dez, 2003, 45 p. Monografia (Bacharelado), Curso de Ciência da Computação, Universidade Luterana do Brasil, Gravataí. 2003.

[75] Rose Extensibility User's Guide. Version: 2001A.04.00, Rational Corp, 2001.

[76] SANTOS, M. ***Seminário de Padrões de Software***. Universidade Federal do Ceará, Ago, 2003. Disponível em: http://www.lia.ufc.br/~misael/mestrado/seminario1.pdf. Acessado em: 08/05/2003.

[77] SANTOS, M. ***Seminário de Padrões de Projeto para Sistemas Web***. Universidade Federal do Ceará, Set., 2003. Disponível em: http://www.lia.ufc.br/~misael/mestrado/seminario2.pdf. Acessado em: 08/05/2003.

[78] SANTOS, M. ***Seminário de Padrões Web***. Instituto Atlântico, Jan, 2003. Disponível em: http://www.lia.ufc.br/~misael/mestrado/web.zip. Acessado em: 08/05/2003.

[79] SANTOS, M. ***Seminário de Padrões de Segurança***. Instituto Atlântico, Fev, 2003. Disponível em: <http://www.lia.ufc.br/~misael/mestrado/seguranca.zip>. Acessado em: 08/05/2003.

[80] SANTOS, M. ***Seminário de Padrões para Interfaces Gráficas***. Instituto Atlântico, Fev, 2003. Disponível em: http://www.lia.ufc.br/~misael/mestrado/gui-patterns.zip. Acessado em: 08/05/2003.

[81] SCHMIDT, D. ***Tutorial About Design Patterns***, Disponível em: http://www.cs.wustl.edu/~schmidt/patterns.html. Acessado em: 13 Nov, 2002.

[82] SecurityPatterns.org. Disponível em: http://www.securitypatterns.org. Acessado em: 11 Fev, 2004.

[83] SOARES, G. ***Desenvolvimento de Sistemas Web em Java.*** Dissertação (Mestrado em Ciência da Computação), Universidade Federal de Pernambuco, Recife, 2002.

[84] SULLIVAN, D. ***A Webmaster's Guide to Search Engines.*** Disponível em: http://searchenginewatch.internet.com. Acessado em: 10/04/2003.

[85] Sun Microsystems. ***Java 2, Enterprise Edition***. Disponível em: http://java.sun.com/j2ee, 1999.

[86] Tamino XML Server. Disponível em: http://www.softwareag.com/tamino. Acessado em: 14/04/2003.

[87] The Diemen Repository of Interaction Design Patterns, Disponível em: <http://www.visiblearea.com/cgi-bin/twiki/view/Patterns/>. Acessado em: 14/05/2004.

[88] *The Hillside Group*. Disponível em http://www.hillside.net. Acessado em: 01 Abr, 2003.

[89] UMLStudio. Disponível em: http://www.pragsoft.com. Acessado em: 10/06/2003.

[90] VLISSIDES, J. ***Patterns: The Top Ten Misconceptions***. Object Magazine, Mar, 1997. Disponível em: <http://hillside.net/patterns/papersbibliographys.htm>. Acesso em: 12/03/2004.

[91] XMLPatterns.com. Disponível em: http://www.xmlpatterns.com. Acessado em: 10/04/03.

[92] WALLACE, N. ***Design Patterns in Web Programming***. Mar, 2000. Disponível em: http://www.e-gineer.com/articles/design-patterns-in-web-programming.phtml. Acessado em: 11/02/04.

[93] WELIE, M. V. ***Amsterdam Patterns Collection***. Disponível em: http://www.welie.com/patterns. Acessado em: 10/02/2004.

[94] WERNER, C. M. L. ***Gerência no Desenvolvimento Baseado em Reutilização***. SBQS 2003, II Simpósio Brasileiro de Qualidade de Software, Fortaleza, CE, 2003.

[95] WERNER, C. M. L.; BRAGA, R. M. M.; MATTOSO, M. L. Q.; MURTA, L. G. P.; COSTA, M. N.; SOUZE, R. P.; OLIVEIRA, A. M. ***Infra-estrutura Odyssey: estágio atual.*** XIV Simpósio Brasileiro de Engenharia de Software, Caderno de Ferramentas, João Pessoa, Out 2000.

[96] WERNER, C.; MATTOSO, M.; BRAGA, R.; BARROS, M.; MURTA, L.; DANTAS, A. ***Odissey: Infra-estrutura de Reutilização baseadas em Modelos de Domínio.*** SBES'99, Caderno de Ferramentas, Florianópolis, Out 1999, pp.17-20.

[97] World Wide Web Consortium. ***eXtensible Markup Language (XML) 1.0.*** Fev, 1998. Disponível em: http://www.w3c.org/XML.

[98] World Wide Web Consortium. ***XML Path Language (XPath) Version 1.0***. W3C Recommendation November 1999. Disponível em: http://www.w3.org/TR/xpath. Acessado em: 04/03/2004.

[99] World Wide Web Consortium. **XQuery** ***1.0: An XML Query Language*****.** W3C Working Draft 12 November 2003. Disponível em: http://www.w3.org/TR/xquery. Acessado em: 04/03/2004.

[100]World Wide Web Consortium. ***XSL Transformations (XSLT) Version 1.0***. Nov, 1999. Disponível em: <http://www.w3c.org/Style/XSL/>.

[101] ZIMMER, W. ***Relationships Between Design Patterns***. Pattern Languages of Program Design, Addison-Wesley, 1995.

Apêndice A – Esquema XML para o Repositório de Padrões

Este apêndice apresenta o esquema XML gerado na fase inicial de implementação do repositório de padrões software para representar a estrutura de um padrão de software e de outras entidades de dados relacionadas (i.e., arquivos, projetos, tipos de relacionamentos).

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="padroes">
     <xs:complexType>
        <xs:sequence>
       <xs:element name="padrao" type="padraoType" maxOccurs="unbounded" />
     </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:complexType name="padraoType">
    <xs:sequence>
      <xs:element name="nome" type="xs:string" />
      <xs:element name="analogias" type="xs:string" minOccurs="0" />
      <xs:element name="palavras_chave" type="xs:string" maxOccurs="unbounded" />
      <xs:element name="resumo" type="xs:string" minOccurs="0" />
      <xs:element name="fonte" type="xs:string" minOccurs="0" />
      <xs:element name="autor" type="xs:string" minOccurs="0" />
      <xs:element name="contexto" type="estruturaType" minOccurs="0" />
      <xs:element name="problema" type="estruturaType" minOccurs="0" />
      <xs:element name="solucao" type="estruturaType" minOccurs="0" />
      <xs:element name="forcas" type="estruturaType" minOccurs="0" />
      <xs:element name="aplicabilidade" type="estruturaType" minOccurs="0" />
      <xs:element name="intencao" type="estruturaType" minOccurs="0" />
      <xs:element name="motivacao" type="estruturaType" minOccurs="0" />
      <xs:element name="estrutura" type="estruturaType" minOccurs="0" />
      <xs:element name="dynamics" type="estruturaType" minOccurs="0" />
      <xs:element name="codigo_exemplo" type="estruturaType" minOccurs="0" />
      <xs:element name="contexto_resultante" type="estruturaType" minOccurs="0" />
      <xs:element name="consequencias" type="estruturaType" minOccurs="0" />
      <xs:element name="descricao" type="xs:string" minOccurs="0" />
      <xs:element name="sintomas" type="xs:string" minOccurs="0" />
      <xs:element name="depende_de" type="xs:string" minOccurs="0" />
      <xs:element name="e_parte_de" type="xs:string" minOccurs="0" />
      <xs:element name="referencias" type="xs:string" minOccurs="0" />
      <xs:element name="creditos" type="xs:string" minOccurs="0" />
      <xs:element name="colaboracoes" type="xs:string" minOccurs="0" />
      <xs:element name="usos_conhecidos" type="estruturaType" minOccurs="0" />
      <xs:element name="ataques_comuns" type="estruturaType" minOccurs="0" />
      <xs:element name="variante" type="estruturaType" minOccurs="0" />
      <xs:element name="importancia" type="estruturaType" minOccurs="0" />
      <xs:element name="implementacao" type="estruturaType" minOccurs="0" />
      <xs:element name="participantes" type="estruturaType" minOccurs="0" />
      <xs:element name="forcas_contrarias" type="estruturaType" minOccurs="0" />
      <xs:element name="estrategias" type="estruturaType" minOccurs="0" />
      <xs:element name="padrao_rel" type="padrao_relType" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="projetos" type="projetoType" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="classificacoes" type="classificacaoType" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="id_padrao" type="xs:string" use="required" />
  </xs:complexType>
  <xs:complexType name="classificacaoType">
    <xs:sequence>
      <xs:element name="classificacao" type="xs:string" maxOccurs="unbounded" />
```

```
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="estruturaType">
    <xs:sequence>
      <xs:element name="descricao" type="xs:string" minOccurs="0" />
      <xs:element name="arquivo" type="arquivoType" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="arquivoType">
    <xs:sequence>
      <xs:element name="caminho" type="xs:string" />
      <xs:element name="descricao" type="xs:string" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="tipo" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="estatico" />
       <xs:enumeration value="dinamico" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  </xs:complexType>
  <xs:complexType name="padrao_relType">
    <xs:sequence>
      <xs:element name="nome_padrao_rel" type="xs:string" />
      <xs:element name="descricao_rel" type="xs:string" minOccurs="0" />
    </xs:sequence>
   <xs:attribute name="tipo_rel" use="optional">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="usa" />
       <xs:enumeration value="refina" />
       <xs:enumeration value="requer" />
       <xs:enumeration value="alternativo" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="projetoType">
    <xs:sequence>
      <xs:element name="projeto" type="xs:string" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Apêndice B – Arquivo XSL referente à transformação realizada na fase de *Simplificação* da etapa de *Conversão* do Processo de Integração

Este apêndice apresenta o arquivo XSL utilizado na transformação realizada na fase de *Simplificação* da etapa de *Conversão* do processo de integração proposto no Capítulo 4. Essa transformação é responsável por converter um arquivo XMI referente ao modelo de um padrão em um arquivo XML simplificado que representa o mesmo modelo de forma resumida (ver Seção 4.6.1.2).

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes" />
<xsl:template match="/">
<project>
    <xsl:for-each select="XMI/XMI.content/Model_Management.Model">
     <xsl:call-template name="Owned" />
    </xsl:for-each>
</project>
</xsl:template>
<xsl:template name="Owned">
<xsl:for-each
select="Foundation.Core.Namespace.ownedElement/Model_Management.Package">
<package>
    <package_name>
    <xsl:value-of select="Foundation.Core.ModelElement.name" />
     </package_name>
    <package_stereotype>
    <xsl:value-of select="Foundation.Core.ModelElement.name" />
   </package_stereotype>
    <xsl:call-template name="Owned" />
 </package>
</xsl:for-each>
<xsl:for-each
select="Foundation.Core.Namespace.ownedElement/Foundation.Core.Class">
<class id="{@xmi.id}">
   <class_name>
   <xsl:value-of select="Foundation.Core.ModelElement.name" />
    </class_name>
    <xsl:for-each
select="Foundation.Core.Classifier.feature/Foundation.Core.Attribute">
  <attribute>
       <attribute_name>
       <xsl:value-of select="Foundation.Core.ModelElement.name" />
      </attribute_name>
   <attribute_name_ucase>
   <xsl:call-template name="UCase">
   <xsl:with-param name="name" select="Foundation.Core.ModelElement.name" />
     </xsl:call-template>
     </attribute_name_ucase>
    <xsl:call-template name="QueryType">
   <xsl:with-param name="id"
     select="Foundation.Core.StructuralFeature.type/Foundation.Core.Classifier/@xmi
     .idref" />
     </xsl:call-template>
     </attribute>
     </xsl:for-each>
   <xsl:for-each
     select="Foundation.Core.Classifier.feature/Foundation.Core.Operation">
   <operation>
```

```
<operation_name>
<xsl:value-of select="Foundation.Core.ModelElement.name" />
  </operation_name>
<xsl:for-each
   select="Foundation.Core.BehavioralFeature.parameter/Foundation.Core.Parameter"
   >
<parameter>
<parameter_name>
<xsl:value-of select="Foundation.Core.ModelElement.name" />
  </parameter_name>
<parameter_kind>
<xsl:value-of select="Foundation.Core.Parameter.kind/@xmi.value" />
  </parameter_kind>
<xsl:call-template name="QueryParameter">
<xsl:with-param name="id"
   select="Foundation.Core.Parameter.type/Foundation.Core.Classifier/@xmi.idref"
   />
  </xsl:call-template>
  </parameter>
  </xsl:for-each>
  </operation>
  </xsl:for-each>
  </class>
  </xsl:for-each>
<xsl:for-each
   select="Foundation.Core.Namespace.ownedElement/Foundation.Core.Association">
<association>
<xsl:for-each
   select="Foundation.Core.Association.connection/Foundation.Core.AssociationEnd"
   >
<association_end>
<association_name>
<xsl:value-of select="Foundation.Core.ModelElement.name" />
  </association_name>
<association_navigable>
<xsl:value-of select="Foundation.Core.AssociationEnd.isNavigable/@xmi.value" />
  </association_navigable>
<association_aggregation>
<xsl:value-of select="Foundation.Core.AssociationEnd.aggregation/@xmi.value" />
  </association_aggregation>
<association_multiplicy_lower>
<xsl:value-of
   select="Foundation.Core.AssociationEnd.multiplicity/Foundation.Data_Types.Mult
   iplicity/Foundation.Data_Types.Multiplicity.range/Foundation.Data_Types.Multip
   licityRange/Foundation.Data_Types.MultiplicityRange.lower" />
  </association_multiplicy_lower>
<association_multiplicy_upper>
<xsl:value-of
   select="Foundation.Core.AssociationEnd.multiplicity/Foundation.Data_Types.Mult
   iplicity/Foundation.Data_Types.Multiplicity.range/Foundation.Data_Types.Multip
   licityRange/Foundation.Data_Types.MultiplicityRange.upper" />
  </association_multiplicy_upper>
<association_type>
<xsl:value-of
   select="Foundation.Core.AssociationEnd.type/Foundation.Core.Classifier/@xmi.id
   ref" />
  </association_type>
  </association_end>
  </xsl:for-each>
  </association>
  </xsl:for-each>
<xsl:for-each
   select="Foundation.Core.Namespace.ownedElement/Foundation.Core.Generalization"
   >
<generalization>
<generalization_child>
```

```
<xsl:value-of
   select="Foundation.Core.Generalization.child/Foundation.Core.GeneralizableElem
   ent/@xmi.idref" />
   </generalization_child>
<generalization_parent>
<xsl:value-of
   select="Foundation.Core.Generalization.parent/Foundation.Core.GeneralizableEle
   ment/@xmi.idref" />
   </generalization_parent>
   </generalization>
   </xsl:for-each>
   </xsl:template>
<xsl:template name="QueryType">
<xsl:param name="id" />
<attribute_type>
<xsl:call-template name="ConvertType">
<xsl:with-param name="type"
   select="/XMI/XMI.content/Model_Management.Model/Foundation.Core.Namespace.owne
   dElement/Foundation.Core.DataType[@xmi.id=$id]/Foundation.Core.ModelElement.na
   me" />
   </xsl:call-template>
   </attribute_type>
<attribute_type_sql>
<xsl:call-template name="ConvertTypeSql">
<xsl:with-param name="type"
   select="/XMI/XMI.content/Model_Management.Model/Foundation.Core.Namespace.owne
   dElement/Foundation.Core.DataType[@xmi.id=$id]/Foundation.Core.ModelElement.na
   me" />
   </xsl:call-template>
   </attribute_type_sql>
   </xsl:template>
<xsl:template name="QueryParameter">
<xsl:param name="id" />
<parameter_type>
<xsl:call-template name="ConvertType">
<xsl:with-param name="type"
   select="/XMI/XMI.content/Model_Management.Model/Foundation.Core.Namespace.owne
   dElement/Foundation.Core.DataType[@xmi.id=$id]/Foundation.Core.ModelElement.na
   me" />
   </xsl:call-template>
   </parameter_type>
   </xsl:template>
<xsl:template name="QueryStereoType">
<xsl:param name="id" />
<package_stereotype>
<xsl:value-of
   select="/XMI/XMI.content/Model_Management.Model/Foundation.Core.Namespace.owne
   dElement/Foundation.Extension_Mechanisms.Stereotype[@xmi.id=$id]/Foundation.Co
   re.ModelElement.name" />
   </package_stereotype>
   </xsl:template>
<xsl:template name="ConvertType">
<xsl:param name="type" />
<xsl:choose>
<xsl:when test="starts-with($type,'date')">Timestamp</xsl:when>
<xsl:when test="starts-with($type,'string')">String</xsl:when>
<xsl:when test="starts-with($type,'char')">String</xsl:when>
<xsl:when test="starts-with($type,'int')">Integer</xsl:when>
<xsl:when test="starts-with($type,'long')">BigDecimal</xsl:when>
<xsl:when test="starts-with($type,'Date')">Timestamp</xsl:when>
<xsl:when test="starts-with($type,'String')">String</xsl:when>
<xsl:when test="starts-with($type,'Integer')">Integer</xsl:when>
<xsl:when test="starts-with($type,'Long')">BigDecimal</xsl:when>
<xsl:otherwise>Object</xsl:otherwise>
   </xsl:choose>
   </xsl:template>
<xsl:template name="UCase">
<xsl:param name="name" />
```

```
   <xsl:value-of
     select="translate(substring($name,1,1),'abcdefghijklmnopqrstuvwxyz','ABCDEFGHI
     JKLMNOPQRSTUVWXYZ')" />
   <xsl:value-of select="substring($name,2)" />
     </xsl:template>
</xsl:stylesheet>
```

Apêndice C – Processo de Conversão XMI / RoseScripts

Este apêndice apresenta os elementos gerados como exemplo de uma conversão de um modelo de classes em XMI para o formato de script do Rose, que acontece na etapa de *Conversão* do processo de integração implementado para o Rose (ver Seção 4.6.1). Para o exemplo foi utilizado um diagrama de classes referente ao padrão *Strategy* [33] que foi modelado utilizando a ferramenta ArgoUML [8]. O diagrama é ilustrado na Figura 28.

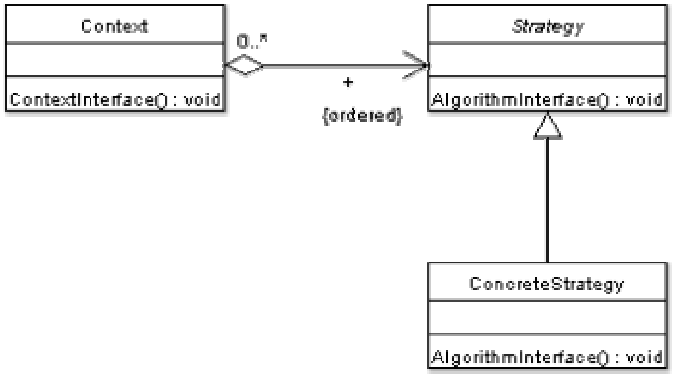

Figura 28 - Diagrama de Classes do Padrão *Strategy*

O ArgoUML traz a funcionalidade de exportação de modelos para o formato XMI. A seguir é apresentada a modelagem em XMI gerada pela ferramenta. A seguir é apresentado o arquivo XMI referente ao modelo do padrão *Strategy* gerado pela ferramenta.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <XMI xmi.version="1.0">
  - <XMI.header>
    - <XMI.documentation>
        <XMI.exporter>Novosoft UML Library</XMI.exporter>
        <XMI.exporterVersion>0.4.19</XMI.exporterVersion>
      </XMI.documentation>
      <XMI.metamodel xmi.name="UML" xmi.version="1.3" />
    </XMI.header>
  - <XMI.content>
    - <Model_Management.Model xmi.id="xmi.1" xmi.uuid="-64--88-2-2-92dcdb:fa981fa44c:-8000">
        <Foundation.Core.ModelElement.name>modelo teste</Foundation.Core.ModelElement.name>
        <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
        <Foundation.Core.GeneralizableElement.isRoot xmi.value="false" />
        <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false" />
        <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false" />
      - <Foundation.Core.Namespace.ownedElement>
        - <Foundation.Core.DataType xmi.id="xmi.2" xmi.uuid="-64--88-2-2-
            92dcdb:fa981fa44c:-7ffa">
            <Foundation.Core.ModelElement.name>void</Foundation.Core.ModelElement.name>
            <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
            <Foundation.Core.GeneralizableElement.isRoot xmi.value="false" />
            <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false" />
            <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false" />
          - <Foundation.Core.ModelElement.namespace>
              <Foundation.Core.Namespace xmi.idref="xmi.1" />
```

```
    </Foundation.Core.ModelElement.namespace>
  </Foundation.Core.DataType>
- <Foundation.Core.Class xmi.id="xmi.3" xmi.uuid="-64--88-2-2-92dcdb:fa981fa44c:-
    7ff4">
    <Foundation.Core.ModelElement.name>Context</Foundation.Core.ModelElement.name>
    <Foundation.Core.ModelElement.visibility xmi.value="public" />
    <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isRoot xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false" />
    <Foundation.Core.Class.isActive xmi.value="false" />
  - <Foundation.Core.ModelElement.namespace>
      <Foundation.Core.Namespace xmi.idref="xmi.1" />
    </Foundation.Core.ModelElement.namespace>
  - <Foundation.Core.Classifier.feature>
    - <Foundation.Core.Operation xmi.id="xmi.4" xmi.uuid="-64--88-2-2-
        92dcdb:fa981fa44c:-7ff3">
        <Foundation.Core.ModelElement.name>ContextInterface</Foundation.Core.Mo
        delElement.name>
        <Foundation.Core.ModelElement.visibility xmi.value="public" />
        <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
        <Foundation.Core.Feature.ownerScope xmi.value="instance" />
        <Foundation.Core.BehavioralFeature.isQuery xmi.value="false" />
        <Foundation.Core.Operation.concurrency xmi.value="sequential" />
        <Foundation.Core.Operation.isRoot xmi.value="false" />
        <Foundation.Core.Operation.isLeaf xmi.value="false" />
        <Foundation.Core.Operation.isAbstract xmi.value="false" />
      - <Foundation.Core.Feature.owner>
          <Foundation.Core.Classifier xmi.idref="xmi.3" />
        </Foundation.Core.Feature.owner>
      - <Foundation.Core.BehavioralFeature.parameter>
        - <Foundation.Core.Parameter xmi.id="xmi.5" xmi.uuid="-64--88-2-2-
            92dcdb:fa981fa44c:-7ff2">
            <Foundation.Core.ModelElement.name>return</Foundation.Core.ModelElem
            ent.name>
            <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
            <Foundation.Core.Parameter.kind xmi.value="return" />
          - <Foundation.Core.Parameter.behavioralFeature>
              <Foundation.Core.BehavioralFeature xmi.idref="xmi.4" />
            </Foundation.Core.Parameter.behavioralFeature>
          - <Foundation.Core.Parameter.type>
              <Foundation.Core.Classifier xmi.idref="xmi.2" />
            </Foundation.Core.Parameter.type>
          </Foundation.Core.Parameter>
        </Foundation.Core.BehavioralFeature.parameter>
      </Foundation.Core.Operation>
    </Foundation.Core.Classifier.feature>
  </Foundation.Core.Class>
- <Foundation.Core.Class xmi.id="xmi.6" xmi.uuid="-64--88-2-2-92dcdb:fa981fa44c:-
    7ff1">
    <Foundation.Core.ModelElement.name>Strategy</Foundation.Core.ModelElement.name
    >
    <Foundation.Core.ModelElement.visibility xmi.value="public" />
    <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isRoot xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isAbstract xmi.value="true" />
    <Foundation.Core.Class.isActive xmi.value="false" />
  - <Foundation.Core.ModelElement.namespace>
      <Foundation.Core.Namespace xmi.idref="xmi.1" />
    </Foundation.Core.ModelElement.namespace>
  - <Foundation.Core.GeneralizableElement.specialization>
      <Foundation.Core.Generalization xmi.idref="xmi.7" />
    </Foundation.Core.GeneralizableElement.specialization>
  - <Foundation.Core.Classifier.feature>
    - <Foundation.Core.Operation xmi.id="xmi.8" xmi.uuid="-64--88-2-2-
        92dcdb:fa981fa44c:-7ff0">
        <Foundation.Core.ModelElement.name>AlgorithmInterface</Foundation.Core.Mo
        delElement.name>
        <Foundation.Core.ModelElement.visibility xmi.value="public" />
        <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
        <Foundation.Core.Feature.ownerScope xmi.value="instance" />
        <Foundation.Core.BehavioralFeature.isQuery xmi.value="false" />
        <Foundation.Core.Operation.concurrency xmi.value="sequential" />
        <Foundation.Core.Operation.isRoot xmi.value="false" />
        <Foundation.Core.Operation.isLeaf xmi.value="false" />
        <Foundation.Core.Operation.isAbstract xmi.value="false" />
```

```
        - <Foundation.Core.Feature.owner>
            <Foundation.Core.Classifier xmi.idref="xmi.6" />
          </Foundation.Core.Feature.owner>
        - <Foundation.Core.BehavioralFeature.parameter>
          - <Foundation.Core.Parameter xmi.id="xmi.9" xmi.uuid="-64--88-2-2-
              92dcdb:fa981fa44c:-7fef">

              <Foundation.Core.ModelElement.name>return</Foundation.Core.ModelE
              lement.name>
            <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
            <Foundation.Core.Parameter.kind xmi.value="return" />
          - <Foundation.Core.Parameter.behavioralFeature>
              <Foundation.Core.BehavioralFeature xmi.idref="xmi.8" />
            </Foundation.Core.Parameter.behavioralFeature>
          - <Foundation.Core.Parameter.type>
              <Foundation.Core.Classifier xmi.idref="xmi.2" />
            </Foundation.Core.Parameter.type>
          </Foundation.Core.Parameter>
        </Foundation.Core.BehavioralFeature.parameter>
      </Foundation.Core.Operation>
    </Foundation.Core.Classifier.feature>
  </Foundation.Core.Class>
- <Foundation.Core.Class xmi.id="xmi.10" xmi.uuid="-64--88-2-2-92dcdb:fa981fa44c:-
    7fec">

      <Foundation.Core.ModelElement.name>ConcreteStrategy</Foundation.Core.ModelE
      lement.name>
    <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isRoot xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false" />
    <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false" />
    <Foundation.Core.Class.isActive xmi.value="false" />
  - <Foundation.Core.ModelElement.namespace>
      <Foundation.Core.Namespace xmi.idref="xmi.1" />
    </Foundation.Core.ModelElement.namespace>
  - <Foundation.Core.GeneralizableElement.generalization>
      <Foundation.Core.Generalization xmi.idref="xmi.7" />
    </Foundation.Core.GeneralizableElement.generalization>
  - <Foundation.Core.Classifier.feature>
    - <Foundation.Core.Operation xmi.id="xmi.11" xmi.uuid="-64--88-2-2-
        92dcdb:fa981fa44c:-7fea">
        <Foundation.Core.ModelElement.name>AlgorithmInterface</Foundation.Core.Mo
        delElement.name>
        <Foundation.Core.ModelElement.visibility xmi.value="public" />
        <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
        <Foundation.Core.Feature.ownerScope xmi.value="instance" />
        <Foundation.Core.BehavioralFeature.isQuery xmi.value="false" />
        <Foundation.Core.Operation.concurrency xmi.value="sequential" />
        <Foundation.Core.Operation.isRoot xmi.value="false" />
        <Foundation.Core.Operation.isLeaf xmi.value="false" />
        <Foundation.Core.Operation.isAbstract xmi.value="false" />
      - <Foundation.Core.Feature.owner>
          <Foundation.Core.Classifier xmi.idref="xmi.10" />
        </Foundation.Core.Feature.owner>
      - <Foundation.Core.BehavioralFeature.parameter>
        - <Foundation.Core.Parameter xmi.id="xmi.12" xmi.uuid="-64--88-2-2-
            92dcdb:fa981fa44c:-7fe9">

              <Foundation.Core.ModelElement.name>return</Foundation.Core.ModelE
              lement.name>
            <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
            <Foundation.Core.Parameter.kind xmi.value="return" />
          - <Foundation.Core.Parameter.behavioralFeature>
              <Foundation.Core.BehavioralFeature xmi.idref="xmi.11" />
            </Foundation.Core.Parameter.behavioralFeature>
          - <Foundation.Core.Parameter.type>
              <Foundation.Core.Classifier xmi.idref="xmi.2" />
            </Foundation.Core.Parameter.type>
          </Foundation.Core.Parameter>
        </Foundation.Core.BehavioralFeature.parameter>
      </Foundation.Core.Operation>
    </Foundation.Core.Classifier.feature>
  </Foundation.Core.Class>
- <Foundation.Core.Generalization xmi.id="xmi.7" xmi.uuid="-64--88-2-2-
    92dcdb:fa981fa44c:-7feb">
    <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
  - <Foundation.Core.ModelElement.namespace>
```

```
            <Foundation.Core.Namespace xmi.idref="xmi.1" />
          </Foundation.Core.ModelElement.namespace>
        - <Foundation.Core.Generalization.child>
            <Foundation.Core.GeneralizableElement xmi.idref="xmi.10" />
          </Foundation.Core.Generalization.child>
        - <Foundation.Core.Generalization.parent>
            <Foundation.Core.GeneralizableElement xmi.idref="xmi.6" />
          </Foundation.Core.Generalization.parent>
        </Foundation.Core.Generalization>
      - <Foundation.Core.Association xmi.id="xmi.13" xmi.uuid="-64--88-2-2-
          92dcdb:fa981fa44c:-7fe8">
          <Foundation.Core.ModelElement.name />
          <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
          <Foundation.Core.GeneralizableElement.isRoot xmi.value="false" />
          <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false" />
          <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false" />
        - <Foundation.Core.ModelElement.namespace>
            <Foundation.Core.Namespace xmi.idref="xmi.1" />
          </Foundation.Core.ModelElement.namespace>
        - <Foundation.Core.Association.connection>
          - <Foundation.Core.AssociationEnd xmi.id="xmi.14" xmi.uuid="-64--88-2-2-
              92dcdb:fa981fa44c:-7fe7">
              <Foundation.Core.ModelElement.visibility xmi.value="public" />
              <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
              <Foundation.Core.AssociationEnd.isNavigable xmi.value="false" />
              <Foundation.Core.AssociationEnd.ordering xmi.value="unordered" />
              <Foundation.Core.AssociationEnd.aggregation xmi.value="aggregate" />
              <Foundation.Core.AssociationEnd.targetScope xmi.value="instance" />
            - <Foundation.Core.AssociationEnd.multiplicity>
              - <Foundation.Data_Types.Multiplicity xmi.id="xmi.15">
                - <Foundation.Data_Types.Multiplicity.range>
                  - <Foundation.Data_Types.MultiplicityRange xmi.id="xmi.16">
                      <Foundation.Data_Types.MultiplicityRange.lower>0</Foundation.Da
                      ta_Types.MultiplicityRange.lower>
                      <Foundation.Data_Types.MultiplicityRange.upper>-
                        1</Foundation.Data_Types.MultiplicityRange.upper>
                    </Foundation.Data_Types.MultiplicityRange>
                  </Foundation.Data_Types.Multiplicity.range>
                </Foundation.Data_Types.Multiplicity>
              </Foundation.Core.AssociationEnd.multiplicity>
              <Foundation.Core.AssociationEnd.changeability xmi.value="changeable" />
            - <Foundation.Core.AssociationEnd.association>
                <Foundation.Core.Association xmi.idref="xmi.13" />
              </Foundation.Core.AssociationEnd.association>
            - <Foundation.Core.AssociationEnd.type>
                <Foundation.Core.Classifier xmi.idref="xmi.3" />
              </Foundation.Core.AssociationEnd.type>
            </Foundation.Core.AssociationEnd>
          - <Foundation.Core.AssociationEnd xmi.id="xmi.17" xmi.uuid="-64--88-2-2-
              92dcdb:fa981fa44c:-7fe6">
              <Foundation.Core.ModelElement.visibility xmi.value="public" />
              <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
              <Foundation.Core.AssociationEnd.isNavigable xmi.value="true" />
              <Foundation.Core.AssociationEnd.ordering xmi.value="ordered" />
              <Foundation.Core.AssociationEnd.aggregation xmi.value="none" />
              <Foundation.Core.AssociationEnd.targetScope xmi.value="instance" />
            - <Foundation.Core.AssociationEnd.multiplicity>
              - <Foundation.Data_Types.Multiplicity xmi.id="xmi.18">
                - <Foundation.Data_Types.Multiplicity.range>
                  - <Foundation.Data_Types.MultiplicityRange xmi.id="xmi.19">

                      <Foundation.Data_Types.MultiplicityRange.lower>1</Foundation
                      .Data_Types.MultiplicityRange.lower>

                      <Foundation.Data_Types.MultiplicityRange.upper>1</Foundation
                      .Data_Types.MultiplicityRange.upper>
                    </Foundation.Data_Types.MultiplicityRange>
                  </Foundation.Data_Types.Multiplicity.range>
                </Foundation.Data_Types.Multiplicity>
              </Foundation.Core.AssociationEnd.multiplicity>
              <Foundation.Core.AssociationEnd.changeability xmi.value="changeable" />
            - <Foundation.Core.AssociationEnd.association>
                <Foundation.Core.Association xmi.idref="xmi.13" />
              </Foundation.Core.AssociationEnd.association>
            - <Foundation.Core.AssociationEnd.type>
                <Foundation.Core.Classifier xmi.idref="xmi.6" />
              </Foundation.Core.AssociationEnd.type>
```

```
            </Foundation.Core.AssociationEnd>
          </Foundation.Core.Association.connection>
        </Foundation.Core.Association>
      - <Foundation.Core.DataType xmi.id="xmi.20" xmi.uuid="-64--88-2-2-
          92dcdb:fa981fa44c:-7fe5">
          <Foundation.Core.ModelElement.name>int</Foundation.Core.ModelElement.name>
          <Foundation.Core.ModelElement.isSpecification xmi.value="false" />
          <Foundation.Core.GeneralizableElement.isRoot xmi.value="false" />
          <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false" />
          <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false" />
        - <Foundation.Core.ModelElement.namespace>
            <Foundation.Core.Namespace xmi.idref="xmi.1" />
          </Foundation.Core.ModelElement.namespace>
        </Foundation.Core.DataType>
      </Foundation.Core.Namespace.ownedElement>
    </Model_Management.Model>
  </XMI.content>
</XMI>
```

Modelo XMI do padrão *Strategy* gerado pelo ArgoUML

O processo de conversão usado consiste de duas fases (ver Seção 4.6.1). A primeira delas (*Simplificação*) gera um arquivo XML simplificado contendo somente os elementos necessários para a transformação final, que irá gerar os *scripts* do Rose. O arquivo XML gerado pela primeira transformação é apresentado a seguir:

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
  <class id="xmi.3">
    <class_name>Context</class_name>
    <operation>
      <operation_name>ContextInterface</operation_name>
      <parameter>
        <parameter_name>return</parameter_name>
        <parameter_kind>return</parameter_kind>
        <parameter_type>Object</parameter_type>
      </parameter>
    </operation>
  </class>
  <class id="xmi.6">
    <class_name>Strategy</class_name>
    <operation>
      <operation_name>AlgorithmInterface</operation_name>
      <parameter>
        <parameter_name>return</parameter_name>
        <parameter_kind>return</parameter_kind>
        <parameter_type>Object</parameter_type>
      </parameter>
    </operation>
  </class>
  <class id="xmi.10">
    <class_name>ConcreteStrategy</class_name>
    <operation>
      <operation_name>AlgorithmInterface</operation_name>
      <parameter>
        <parameter_name>return</parameter_name>
        <parameter_kind>return</parameter_kind>
        <parameter_type>Object</parameter_type>
      </parameter>
    </operation>
  </class>
  <association>
    <association_end>
      <association_name />
```

```
        <association_navigable>false</association_navigable>
        <association_aggregation>aggregate</association_aggregation>
        <association_multiplicy_lower>1</association_multiplicy_lower>
        <association_multiplicy_upper>1</association_multiplicy_upper>
        <association_type>xmi.3</association_type>
      </association_end>
      <association_end>
        <association_name />
        <association_navigable>true</association_navigable>
        <association_aggregation>none</association_aggregation>
        <association_multiplicy_lower>0</association_multiplicy_lower>
        <association_multiplicy_upper>-1</association_multiplicy_upper>
        <association_type>xmi.6</association_type>
      </association_end>
    </association>
    <generalization>
      <generalization_child>xmi.10</generalization_child>
      <generalization_parent>xmi.6</generalization_parent>
    </generalization>
  </project>
```

Arquivo XML simplificado gerado a partir na fase de *Simplificação* da *Conversão*.

A segunda transformação XSL (*Finalização*) tem por finalidade gerar o *script* para o Rose que será responsável pela inclusão do modelo na ferramenta (*RoseScript*). A seguir é apresentado o *script* gerado por esta transformação XSL.

```
'--------------------------------------------
'        strategy.ebs
'
'        Scripts para a insercao de modelos...
'
'        Autor: Misael Santos
'        Data: 04/02/04
'
'--------------------------------------------

' declara funcoes do funcoes.ebx
Declare Function CriarClasse(className As String, stereotype As String, doc As String) As
Class
Declare Function GetClassByName(t As String) As Class

' Subrotina Principal
Sub Main

  ' load library To use functions
  RoseApp.LoadScriptModule "funcoes.ebx"

 Dim novaClasse As Class
  Dim novoAtributo As Attribute
  Dim novaOperacao As Operation
  Dim novaAssociacao As Association
  Dim role1 As Role
  Dim role2 As Role
  Dim result As Boolean

  Dim Mensagem As String
  Mensagem = "Confirma a insercao do modelo?"
  If MsgBox(Mensagem, ebYesNo Or ebQuestion, "Inserindo o modelo")=ebYes Then

        result = true

        '================================================================
        ' Criando a CLASSE Context
        Set novaClasse = CriarClasse("Context", "", "Classe Context...")
        ' Verificando se a classe foi inserida corretamente...
        If novaClasse Is Nothing Then
                result = false
```

```
                GoTo Fim
        End If
        '-----------------------------------------------------------------
        ' Criando ATRIBUTOS da classe Context
        '-----------------------------------------------------------------
        ' Criando OPERACOES da classe Context
        Set novaOperacao = novaClasse.AddOperation("ContextInterface", "")

        novoParametro = novaOperacao.AddParameter("return", "Object", "", 0)

        '=================================================================

        '=================================================================
        ' Criando a CLASSE Strategy
        Set novaClasse = CriarClasse("Strategy", "", "Classe Strategy...")
        ' Verificando se a classe foi inserida corretamente...
        If novaClasse Is Nothing Then
                result = false
                GoTo Fim
        End If
        '-----------------------------------------------------------------
        ' Criando OPERACOES da classe Strategy
        Set novaOperacao = novaClasse.AddOperation("AlgorithmInterface", "")
        novoParametro = novaOperacao.AddParameter("return", "Object", "", 0)
        '=================================================================


        '=================================================================
        ' Criando a CLASSE ConcreteStrategy
        Set novaClasse = CriarClasse("ConcreteStrategy", "", "Classe ConcreteStrategy...")
        ' Verificando se a classe foi inserida corretamente...
        If novaClasse Is Nothing Then
                result = false
                GoTo Fim
        End If
        '-----------------------------------------------------------------
        ' Criando OPERACOES da classe ConcreteStrategy
        Set novaOperacao = novaClasse.AddOperation("AlgorithmInterface", "")
        novoParametro = novaOperacao.AddParameter("return", "Object", "", 0)
        '=================================================================

        '=================================================================
        ' Criando as ASSOCIACOES
        Set novaAssociacao = GetClassByName("Context").AddAssociation ("", "Strategy")
        Set role1 = novaAssociacao.GetCorrespondingRole(GetClassByName("Context"))
        Set role2 = novaAssociacao.GetCorrespondingRole(GetClassByName("Strategy"))
        role1.Navigable = false
        role2.Navigable = true
        role1.Cardinality = "1..1"
        role2.Cardinality = "0..*"
        role1.Aggregate = true
        role2.Aggregate = false
        '=================================================================

        '=================================================================
        ' Criando as GENERALIZACOES
        Set theInheritRel = GetClassByName("ConcreteStrategy").AddInheritRel ("",
"Strategy")
        '=================================================================

        Fim:

        If result Then
                MsgBox "Modelo inserido com sucesso!", ebInformation, "Criando uma classe"
        Else
                MsgBox "Erro ao inserir o novo modelo...", ebOkOnly, "Erro"
        End If

  End If

  ' unload library to use functions
  RoseApp.FreeScriptModule "funcoes.ebx"

End Sub
```

Script do Rose gerado pela transformação XSL.

O *script* gerado está pronto pra ser executado pelo Rose. A seguir, a Figura 29 ilustra o diagrama gerado pelo *script* executado pelo Rose.

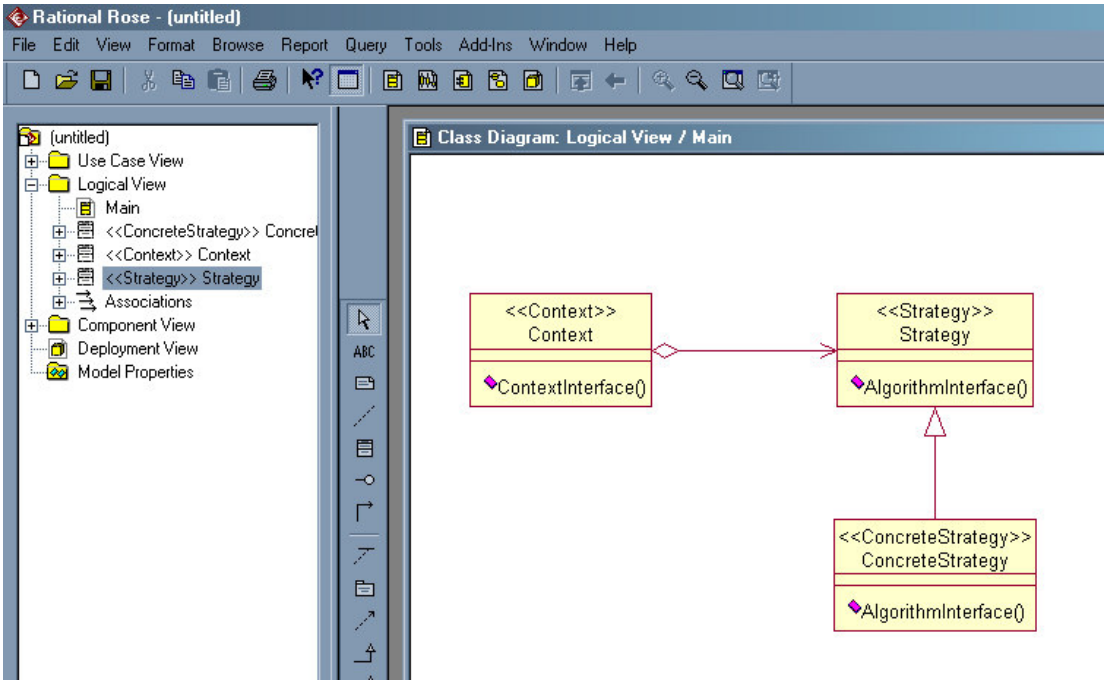

Figura 29 - Diagrama gerado no Rose a partir da execução do *script* obtido na *Conversão*

mailto: misael@lia.ufc.br

Instalação do IIMPaR: http://www.lia.ufc.br/~misael/iimpar-install.exe