UNIVERSIDADE DO VALE DO RIO DOS SINOS

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM

COMPUTAÇÃO APLICADA

Cristiano Tonietto Galina

RPAO : REPOSITÓRIO DE PADRÕES DE ANÁLISE MAPEADOS EM ONTOLOGIAS

São Leopoldo

2007.

CRISTIANO TONIETTO GALINA

RPAO : REPOSITÓRIO DE PADRÕES DE ANÁLISE MAPEADOS EM ONTOLOGIAS

Dissertação a ser apresentada à Universidade do Vale do Rio dos Sinos como requisito parcial para a obtenção do título de Mestre em Computação Aplicada

Orientador: Prof. Dr. Sérgio Crespo C. S. Pinto

São Leopoldo

2007.

"*É durante as fases de maior adversidade que surgem as grandes oportunidades de se fazer o bem a si mesmo e aos outros.*"
(Dalai-Lama)

DEDICATÓRIA

Dedico este trabalho a pessoa mais importante
da minha vida: minha noiva, Daniela Cattani, pelo
seu companheirismo e compreensão;
meus pais e meus irmãos pelas boas vibrações.

AGRADECIMENTOS

Gostaria de agradecer a algumas pessoas que contribuíram com o andamento do meu trabalho:

A todos os bons mentores espirituais que me acompanharam nas horas de baixo astral e me deram luz e força para continuar o trabalho diante de todos os problemas da vida;

Aos meus pais, pelo interesse e incentivo;

A minha noiva a qual acompanhou todas as frustrações e me amparou com seu carinho e amor para buscar o título de mestre;

Ao meu melhor amigo, Gugu (Gustavo Bisog), que por muitas vezes através de telefonemas, msn e e-mail foi um grande amigo para as mais diversas situações;

A empresa Tlantic a qual me forneceu subsídios tecnológicos, estudo de caso e o grande interesse da minha conclusão de dissertação.

Ao professor Dr. Sérgio Crespo Coelho da Silva Pinto, primeiramente pela bolsa de estudos fornecida a qual financiou os meus estudos durante o mestrado, pela sua orientação, auxílio e paciência nos momentos de reclamações, discussões e realizações.

A querida amiga e irmã espiritual Karina Dallagno que sempre esteve interessada na minha conclusão;

A todos que, de alguma forma, contribuíram para a realização deste trabalho.

RESUMO

A aplicação de padrões de análise visa promover o reuso, de maneira eficiente, de boas práticas da engenharia de software. O reuso deve-se à experiência dos engenheiros de software ao identificar os padrões e descrevê-los de forma que outros projetos também usufruam do padrão de análise já identificado. A engenharia de requisitos tem uma grande influência na descoberta de padrões de análise. Esta dissertação se propõe a criar um repositório de padrões de análise, visto que o mercado não oferece hoje uma ferramenta propícia para o armazenamento dos padrões de análise e mapeamento em ontologias no formato OWL. As ontologias são geradas pelo software toda vez que um padrão é catalogado, portanto, todo padrão documentado é disponível por uma ontologia com a estrutura do padrão.

Além do armazenamento de padrões de análise, a ferramenta realiza um mapeamento com requisitos e artefatos, os quais possivelmente originam um padrão, desta forma apresentando uma base de conhecimento do padrão. Em seguida é possível visualizar através de um gráfico o reuso dos padrões de análise para os projetos relacionados. A partir de um formato XMI(XML *Metadata Interchange)* é possível criar um processo de importação de casos de uso que representam a fase de elaboração de análise.Desta forma a dissertação utiliza abordagens de engenharia de *software* focadas para o reuso de padrões de análise e a criação de uma base de conhecimento que possa ser apresentado em ontologias.

Palavras-chave: Padrões de Análise, Padrões de Software, Reuso de Software, Modelos de Padrões, UML, Ontologia.

Ferramentas de Apoio ao Desenvolvimento de Sistemas, UML, XMI, Protégé.

ABSTRACT

Application of analysis patterns aims at promoting reuse under efficient software engineering. Reuse is due to software engineers' experience in identifying patterns and in describing them in a way that other kinds of projects are able to employ analysis patterns already identified. Engineering of requisites has great influence in developing analysis patterns. This paper aims at creating a selection of analysis patterns once there are not many tools available for the storage of such patterns. It also aims at mapping them under the OWL format ontologism. Ontologies are generated by the software every time that a pattern is categorized, therefore, all documented pattern is available by the ontology regarding its pattern structure.

Besides the pattern storage of analysis, the tool also performs a mapping with the requisites and the artifacts, which are likely to produce a pattern that present the basis of the knowledge pattern. Following, it is possible to visualize the reuse of patterns through a graphic developed for the related projects. It is possible to create an import process for the cases of use that represent the analysis development phase after a XML metadata exchange format as well.

This dissertation is therefore developed under software engineering approaches which focus on the reuse of analysis patterns and on the creation of a knowledge base that can be useful in ontologies.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AOO	*Análise Orientado a Objetos*
BO	*Business Object*
CMMI	*Capability Maturity Model Integration*
DAML	*DARPA Agent Markup Language*
DAO	*Database Access Object*
EBT	*Entity Business Transaction*
ER	*Entidade Relacionamento*
EFR	*Especificação Funcional de Requisitos*
ERS	*Especificação de Requisito de Software*
IOC	*Initial Operational Capability*
J2EE	*Java Second Enterprise Edition*
JSP	*Java Server Pages*
LCA	*Lifecycle architecture*
LCO	*Lifecycle objetive*
MVC	*Model View Controler*
OWL	*Ontology Web Language*
POO	*Projeto orientado a objetos*
PR	*Product Release*
RD	*Requirements Development*
RDF	*Resource Description Framework*
RM	*Requirements Management*
RPAO	*Repositório de padrões de análise mapeados em Ontologia*
RUP	*Rational Unified Process*
SGBD	*Sistema de Gerenciamento de Base de Dados*
SQL	*System Query Language*
UML	*Unified Model Language*
URL	*Uniform Resource Locators*
VO	*Value Object*
W3C	*World Wide Web Consortium*
WEB	*World Wide Web*
XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

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 reuso de software vêm acompanhando a evolução da engenharia de software.

Atualmente, na engenharia de software, o conceito de reutilização de software envolve muito mais do que reaproveitar código fonte de programas. Existem muitas formas de estudar meios de reutilização de componentes, documentos, modelos, fluxogramas e artefatos em geral. Segundo [GAMM 95] [BYSC 96], documentar padrões é uma maneira de se reutilizar esse conhecimento em forma de soluções genéricas para problemas recorrentes. Neste sentido, o reuso de padrões de software tem se mostrado um eficiente mecanismo para a reutilização das boas práticas de engenharia de software.

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 prática efetiva 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 [COPL 96] [COPL 00] [FOWL 97] [PREE 95]. O conceito e o reuso de padrões de software vêm cada vez mais sendo difundidos entre analistas e desenvolvedores de sistemas, o que reafirma a importância desta área de pesquisa dentro da engenharia de software.

Atualmente, padrões podem ser utilizados individualmente na resolução de problemas isolados ou específicos, ou podem ser utilizados em conjunto, através do uso de coleções de padrões para resolver problemas complexos ou para documentar e gerar arquiteturas de software. A documentação desses padrões ocorre de duas formas:

- 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.

Outra linha de estudo na área de reutilização de software é a engenharia de domínio, definida em [WERN 00] 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 aos 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.

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 [BORL 05] [BUDI 96] [CODE 06] [CONT 02]. Estas ferramentas surgiram com o propósito de facilitar a aplicação de padrões de projeto no desenvolvimento de software, modelagem de classe e geração de código.

Os padrões aplicados automaticamente, através da utilização das ferramentas citadas são geralmente restritos a um conjunto de padrões fornecidos 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.

Booch [JACO 02] afirma que uma empresa de software de sucesso é aquela que, consistentemente, desenvolve sistemas que atendam às necessidades dos usuários. Diagramas de fácil entendimento, reuniões estruturadas, programação com técnicas apuradas e todo o resto, são secundários, mas não irrelevantes. Para desenvolver um sistema que atenda às necessidades do usuário necessita-se que ele se engaje no projeto de maneira efetiva e organizada, definindo os reais requisitos do sistema. Para desenvolver rapidamente, eficientemente e efetivamente, com o mínimo de re-trabalho e perdas, é necessário ter as pessoas certas, as ferramentas adequadas e o foco correto.

1.1. Motivação

Fowler [FOWL 97] salienta que muitas empresas o procuram hoje para ser o mentor dos projetos por já ter uma experiência vasta em reuso de modelos. Para Fowler, sua experiência foi adquirida pela aplicabilidade dos modelos e pelo fato de aprender a observar os mesmos em muitos projetos decorrentes, assim aprendeu a reusá-los de maneira adequada e criando os padrões relacionados entre os modelos. Fowler lembra que "*um padrão é uma idéia que tenha sido proveitosa em um contexto prático e provavelmente seja proveitosa em outros*" [FOWL 97].

Uma analogia a padrões pode ser dada por ontologias, as quais oferecem mecanismos que valorizam a semântica dos dados por meio da definição de conceitos, relacionamentos e possíveis restrições de um domínio. Para que a ontologia apresente todas as suas vantagens e corresponda ao objetivo da representação de certo domínio, sugere-se que algumas etapas sejam realizadas durante seu desenvolvimento. Assim, a construção de uma ontologia envolve processos, desde a identificação de objetivos até a sua representação propriamente dita [NATA 06].

Os itens apresentados motivaram o desenvolvimento à proposta deste trabalho, que deseja criar um repositório de padrões de análise sendo este repositório flexível por meio de uma base de conhecimento de ontologias. Para criar o ambiente de repositório de padrões de análise integrados às ontologias, pretende-se utilizar técnicas computacionais e de engenharia de *software*, tais como padrões de análise, padrões de projetos, Ontologias e métodos de integração de dados. Julga-se necessário então, criar uma ferramenta que possibilite apresentar padrões de análise catalogados em uma base de conhecimento por meio de uma ontologia, permitindo que o engenheiro de software obtenha uma coleção de artefatos reutilizáveis na construção de um software.

1.2. Contexto do trabalho

Este trabalho está inserido dentro da linha de pesquisa da Engenharia de Software e Linguagens de Programação. Para o seu desenvolvimento são necessários estudos sobre Design Patterns, Microsoft .NET, XML (eXtensible Markup Language), Engenharia de Requisitos e Ontologias. A Figura 1 especifica as áreas relacionadas.

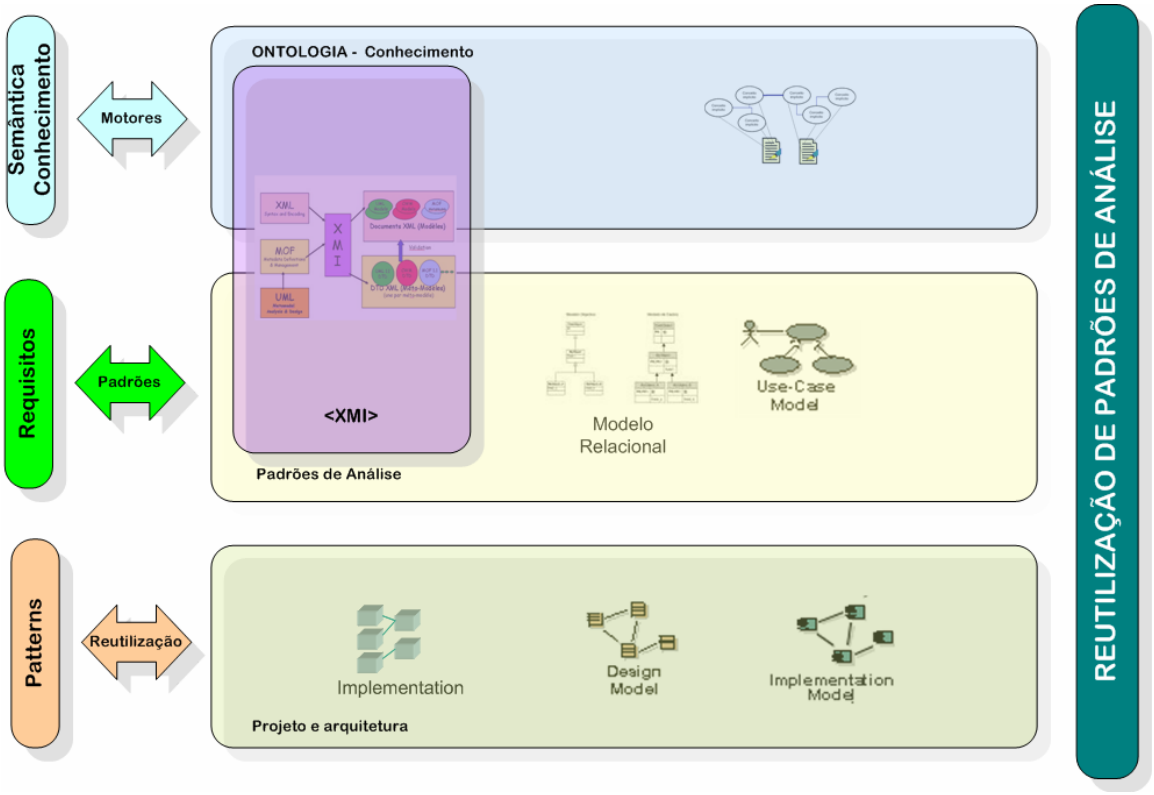

Figura 1: Áreas relacionadas ao trabalho

1.3. Problema

O problema deste trabalho está em criar uma ontologia de padrões de análise integrada com uma ferramenta que realize a consulta dinâmica destes padrões. As ontologias poderão ser criadas através da ferramenta de repositório de padrões com mecanismos para gerir as informações realizando a integração das ontologias.

1.4. Questão de Pesquisa

A questão central deste trabalho é: como armazenar padrões de análise, consultá-los de forma dinâmica e fornecê-los para sistemas externos viabilizando a interoperabilidade entre as soluções, de forma que a solução desenvolvida proporcione reuso, qualidade e até mesmo competitividade para a empresa?

1.5. Objetivos

O objetivo geral deste trabalho será modelar e desenvolver uma arquitetura de software que permita o desenvolvimento de um ambiente de reutilização de modelos de análise.

Visando alcançar este objetivo, definiram-se os seguintes objetivos específicos:

- realizar um estudo aprofundado sobre as tecnologias necessárias para o desenvolvimento de *Design Patterns,* Componentização*,* Engenharia de Requisitos, *Analisys Patterns* e Ontologia ;
- abordar a metodologia RUP para desenvolvimento focado em uma metodologia;
- modelar os recursos a serem desenvolvidos;
- implementar as diversas camadas existentes no sistema;
- enquadrar todos os processos realizados em análise e criar documentação de toda a aplicação de desenvolvimento;
- testar e preparar distribuição final do software.

1.6. Organização do Volume

Esta dissertação está divida em 6 capítulos, sendo o primeiro esta introdução. Os demais capítulos são descritos a seguir:

- Capítulo 2: Conceitos Fundamentais – fase em que são descritos alguns conceitos fundamentais para o entendimento do contexto do trabalho. Assim, o texto fala de ontologias e metodologias de desenvolvimento.
- Capítulo 3: Tecnologia de Engenharia de *Software* – Capítulo que trabalha com as tecnologias de engenharia de *software*. O capítulo apresenta um estudo focado em padrões de projetos, padrões de análise e RUP(*Rational Unified Process).*
- Capítulo 4: Trabalhos Relacionados – realiza a leitura de alguns trabalhos acadêmicos cujo tema se aplica à padrões de análise e ontologias;

- Capítulo 5: RPAO - Repositório de padrões de análises mapeados em Ontologias – descreve a solução a ser desenvolvida e aplicada no estudo de caso;
- Capítulo 6: Estudo de caso – descreve um exemplo real da aplicação da ferramenta desenvolvida;
- Capítulo 7: Conclusão – apresenta algumas considerações quanto ao desenvolvimento do trabalho, indicando pontos de melhoria, bem como a continuidade do trabalho;

2. TECNOLOGIAS DE ENGENHARIA DE SOFTWARE

O reuso de sistemas de aplicações é um dos objetivos principais de grande parte de desenvolvedores de software, visto que, custos gerais de desenvolvimento podem ser reduzidos, menos componentes de software têm que ser especificados, projetados, implementados e validados. O objetivo deste capítulo é apresentar ao leitor tecnologias ligadas a Engenharia de Software que buscam a reutilização de componentes e a interoperabilidade entre sistemas.

2.1. Engenharia de Requisitos

O processo de especificação de um sistema para o computador pode ter diferentes níveis de aceitação. Para os engenheiros de hoje, torna-se um desafio saber se o sistema foi corretamente especificado e se atende às expectativas dos clientes. Não existe uma resposta correta para essa questão, porém existe um procedimento sólido para a sua identificação, que é a Engenharia de Requisitos. É uma expressão nova proposta para cobrir todas as atividades envolvidas no descobrimento, documentação, análise e manutenção de um conjunto de requisitos para um sistema baseado em computador. O uso do termo "engenharia" consiste em aplicar técnicas sistemáticas e repetíveis a serem usadas para assegurar que os requisitos do sistema sejam completos, relevantes e eficazes [SOMM 04].

De acordo com Nuseibeh [NUSE 00] a Engenharia de Requisitos é uma etapa da Engenharia de Software, que tem por objetivo realizar um levantamento das funções, necessidades de um software, bem como também identificar os stakeholders e suas necessidades e documentar todo o processo de análise e documentação das etapas de desenvolvimento de software. Recentes estudos comprovam que, ao desenvolver um software, um dos principais problemas ocorridos é verificado no processo de Engenharia de Requisitos. Através da análise de 8.000 projetos de 350 companhias norte-americanas, foi possível observar que a maioria dos projetos obteve apenas acabamento parcial, por causa de uma precária análise da Engenharia de Requisitos, o que acarretou aumento de custos e de cronograma de entrega [LAMS 00].

O desenvolvimento correto da Engenharia de Requisitos ocorre através da identificação dos objetivos e operações que o sistema deverá suportar, e busca descobrir responsabilidades para os agentes, tais como com os seres humanos, dos dispositivos e do *software*. Estudos contemporâneos comprovam que a preocupação das empresas é crescente com relação aos processos de Engenharia de Requisitos, pois através deles torna-se viável a elaboração de projetos em menor tempo, fato que proporciona ao cliente uma maior eficiência do produto desenvolvido [LAMS 00].

2.1.1. **Tipos de requisitos**

Segundo Wiegers [WIEG 03], são usados muitos termos comuns para especificar a Engenharia de Requisitos. Assim os requisitos de *software* são incluídos em três níveis distintos: requisitos de negócio, requisitos de usuários e requisitos funcionais. Em adição, os sistemas terão uma grande variedade de requisitos não-funcionais. A Figura 2 ilustra diversos tipos de requisitos que podem ser utilizados na especificação de um sistema.

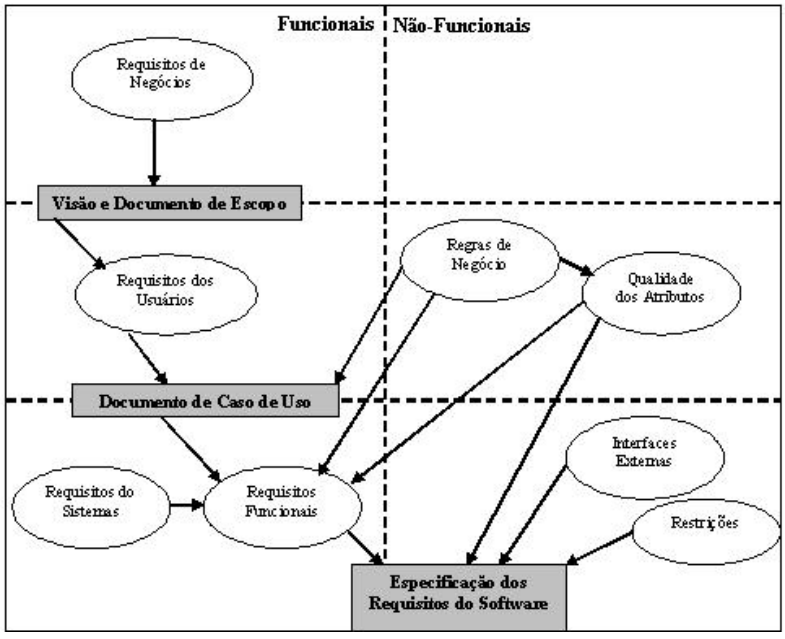

Figura 2: Relacionamento entre diversos tipos de requisitos [WIEG 03].

De acordo com a Figura 2, os requisitos de negócios representam os objetivos da organização ou as expectativas do cliente num nível mais elevado. Através desses requisitos é possível ter uma visão geral sobre o procedimento e a obtenção de um documento de escopo do sistema. Os requisitos dos usuários descrevem os objetivos e tarefas dos usuários com o sistema, e podem ser representados através de documentos de casos de usos, de descrições de

cenários e de eventos. Os requisitos funcionais especificam a funcionalidade do *software*, que habilita o usuário a desempenhar tarefas que satisfaçam os requisitos de negócios. Já os requisitos de sistema ocorrem quando um requisito funcional depende de um outro requisito para o seu correto funcionamento, como, por exemplo, ocorre num sistema operacional.

As regras de negócios são políticas de sistemas, e não são consideradas requisitos, mas atuam para especificar as qualidades dos atributos que são implementados. Os requisitos funcionais são documentados em uma especificação de requisitos de *software* (ERS), que descreve todo o comportamento necessário do *software*. Estes SRS contêm também requisitos não funcionais como qualidade dos atributos, descrevendo, assim, a funcionalidade do *software* através de características como portabilidade, eficiência e usabilidade. Outro requisito não-funcional que pode ser mencionado são as *interface*s externas, que descrevem a relação existente entre o sistema e o mundo real, entre o projeto e a implementação das restrições. Essas representam exceções que devem ser levadas em consideração no desenvolvimento do projeto. Na Figura 2, foram analisados os requisitos em um fluxo *"top-down"*, ou seja, fica explícito o procedimento que descreve a interação entre os requisitos de negócio, de usuários e os funcionais. Logo, dividem-se os requisitos em dois grupos:

Requisitos funcionais: são aqueles que estão explícitos, ou seja, que são definidos como as funções ou as atividades que o sistema executa (ou executará), descritos de maneira clara e formal. De acordo com Mylopoulos [MYLO 92], essa etapa é fundamental a qualquer projeto de *software*, porque aqui fica definido o que o sistema fará através dos requisitos globais dos custos de desenvolvimento e dos custos operacionais, proporcionando, assim, a abrangência das questões de qualidade, que são os requisitos não-funcionais.

Requisitos não-funcionais: são aqueles que não podem ser definidos em termos de funcionalidade. A crescente necessidade de definir modelos conceituais capazes de lidar com metas, retratando a necessidade do mundo real, proporciona a capacidade de representar requisitos não-funcionais, como confidencialidade, performance, qualidade e precisão. Com o crescente aumento da tecnologia, ocorre o surgimento de impactos, causados por restrições operacionais de *software* e de evolução da tecnologia; isso se torna hoje um grande desafio no momento de definição dos requisitos não-funcionais. Como exemplo de trabalhos desenvolvidos na área, pode ser citado o esforço de Mylopoulos [MYLO 92]. Este autor propõe um *framework* para representar os requisitos não-funcionais no processo de desenvolvimento do *software.* Esse é composto por cinco elementos básicos que provêm a representação dos requisitos não-funcionais, através dos seus objetivos. Para cada objetivo é possível observar a evolução dos requisitos não-funcionais. Tais requisitos ainda buscam

metas conflitantes, nas quais são identificadas, genericamente, e refinadas até que se chegue a um conjunto de requisitos que satisfaçam o principal. Este trabalho ainda busca ajudar nas decisões relativas ao domínio da aplicação e auxiliar na detecção de erros provenientes da utilização desse método.

2.2. Design Patterns

Ao questionar-se, por que a tecnologia de orientação a objetos é mais recomendada que as suas concorrentes, grande parte dos engenheiros de software responderiam "reuso". Na realidade, a reutilização de componentes de software está diretamente relacionada a duas questões que afetam diretamente aos projetistas e aos usuários dos produtos: qualidade e produtividade [METS 04].

É difícil construir um produto totalmente reutilizável logo na primeira tentativa. Isso significa que a experiência dos projetistas é fundamental para desenvolver bons componentes de software. Profissionais experientes costumam reutilizar soluções que já funcionaram no passado, ao invés de resolver cada problema partindo do zero [GAMM 95].

Um padrão é uma maneira de fazer algo, ou de buscar um objetivo. Em qualquer atividade que esteja madura ou em vias de amadurecer, encontraremos métodos eficazes comuns para atingir objetivos e para resolver problemas em vários contextos. A comunidade de pessoas que praticam um ofício geralmente cria um jargão que os ajuda a falar a respeito dele. Tal jargão freqüentemente se refere a padrões, ou maneiras padronizadas de atingir certos objetivos. Os escritores documentam esses padrões ajudando a padronizar o jargão. Além disso, também garantem que a sabedoria acumulada de um ofício esteja disponível para futuras gerações de profissionais [METS 04].

Os primeiros conceitos de padrões surgiram na área de arquitetura, com os trabalhos de Christopher Alexander, nas décadas de 60 e 70 [ALEX 79] [ALEX 77]. 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:

"*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" [ALEX 77].

Segundo [RHEI 04], *Design Patterns* (Padrões de Projeto) são soluções genéricas para problemas recorrentes em Engenharia de *Software*. Conforme Christopher Alexander [GAMM 95]: "cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira". Cada padrão identifica classes e instâncias participantes com seus papéis, colaborações e a distribuição de responsabilidades, sendo que estes elementos podem ser customizados para resolver um problema num contexto particular [GAMM 95].

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, será considerado esse conceito de padrões.

Padrões capturam o conhecimento de profissionais experientes, auxiliando no compartilhamento da informação entre os desenvolvedores. A utilização de padrões também pode ser vista como ferramenta de aprendizado, mostrando boas técnicas de modelagem e desenvolvimento [ERIC 00]. Com isso, o projeto deixa de ser dependente de uma única pessoa, a qual não transmite seus padrões e suas técnicas.

2.3. A necessidade do uso de Modelos

Desenvolvimento de software é uma engenharia, cujo processo de aprendizado pode ser catalisado pelo fato de poder olhar e consultar outras construções. Procurar acertos de outros engenheiros, entender os problemas e visualizar se estes foram ou não resolvidos, são aspectos relacionados ao aprendizado humano. No caso de software, um dos problemas está no fato das estruturas de programação não estarem aparentes como as estruturas de construção civil ou de máquinas de manufatura [FOWL 97]. Desta forma, faz-se necessário o uso de modelos para comunicar estas estruturas sendo necessário que seus atores as divulguem.

A modelagem é a parte central do desenvolvimento de *software* de qualidade. Modelos são construídos para comunicar a estrutura desejada, o comportamento do sistema, para

visualizar e controlar a arquitetura e permitir o entendimento do sistema em construção, explorando oportunidades de simplificação, reuso e administrando riscos [JACO 02].

2.4. Benefícios do Uso 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 [VLIS 97] 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 [ANDE 01] e publicada em [ANDR 03], 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 gerem 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;
- Facilidade de aprendizado de novas áreas de conhecimento para uma equipe sem experiência na aplicação a ser desenvolvida.

2.5. Classificação

O *design patterns* possui quatro elementos essenciais [GAMM 95]:

- **nome** – identifica o problema e a solução adotada em uma ou duas palavras;
- **problema** – informa em que ocasiões o padrão pode ser utilizado;
- **solução** – descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaboração. A solução não descreve uma implementação articular. Ao invés disso, fornece uma descrição abstrata de um problema e como uma combinação de classes e objetos o resolve;
- **conseqüências -** informam os resultados e desafios na utilização do padrão a fim de esclarecer os custos e benefícios de sua aplicação.

Devido à granularidade e ao seu nível de abstração, padrões de projeto são agrupados em famílias. Tal agrupamento é feito a partir de dois critérios. O primeiro diz respeito à finalidade (que reflete o que o padrão faz) e podem ter finalidade de criação, estrutural ou comportamental. O segundo refere-se ao escopo e especifica se o padrão se aplica primeiramente a classes ou a objetos. Desta forma os padrões podem ser tabelados conforme exibido na Figura 4 [GAMM 95]:

		Propósito		
		1 - Criação	2 -Estrutura	3 - Comportamento
Escopo	Classe	*Factory Method*	*Class Adapter*	*Interpreter* *Template Method*
	Objeto	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Object Adapter* *Bridge* *Composite* *Decoretor* *Façade* *Flyweight* *Proxy*	*Chain of Responsibility* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

Figura 3: *Design Patterns* segundo GAMMA [GAMM 95].

Padrões de criação abstraem o processo de instanciação. Ajudam a tornar o sistema independente de como os objetos são criados, compostos e representados. Um padrão de criação de classes usa a herança para variar a classe que é instanciada, enquanto um padrão de criação de objetos delega a instanciação para outro objeto. Os padrões de criação dão muita flexibilidade quanto ao que é criado, quem cria, como e quando é criado. Permitem configurar um sistema com objetos "produto" que variam amplamente em estrutura e funcionalidade, sendo que esta configuração pode ser estática (especificada em tempo de compilação) ou dinâmica (especificada em tempo de execução);

Padrões estruturais preocupam-se em como classes e objetos são compostos para formar estruturas maiores. Os padrões estruturais de classes utilizam a herança para compor interfaces ou implementações. Este tipo de padrão é particularmente útil para fazer bibliotecas de classes desenvolvidas independentemente trabalharem juntas. Já os padrões estruturais de objetos, em lugar de compor interfaces ou implementações, descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade provém da capacidade de mudar a composição em tempo de execução, o que é impossível com a composição estática de classes;

Padrões comportamentais preocupam-se com algoritmos e a atribuição de responsabilidades entre objetos. Não descrevem apenas padrões de objetos ou classes, mas também os padrões de comunicação entre eles. Estes padrões caracterizam fluxos de controle difíceis de seguir em tempo de execução; eles afastam o foco do fluxo de controle para que seja possível concentrar-se somente na maneira como os objetos são interconectados. Padrões comportamentais de classes utilizam herança para distribuir o comportamento entre classes, enquanto padrões comportamentais de objetos utilizam a composição de objetos. Alguns

descrevem como um grupo de objetos pares (*peer objects*) cooperam para a execução de uma tarefa que nenhum objeto sozinho poderia executar por si mesmo.

2.6. Relacionamento 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.

Estes são exemplos que descrevem alguns dos tipos de relacionamentos existentes entre padrões de software. Muitos templates de padrões 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. A Figura 5 apresenta os relacionamentos entre os padrões.

Zimmer [ZIMM 95] 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 [CONT 02], são formalizados quatro tipos de relacionamentos possíveis entre padrões.

- Se um padrão P1 ***usa*** um padrão P2, então a solução do padrão P1 deve ser expressa usando P2.

- Se um padrão P1 ***refina*** um padrão P2, então o problema do padrão P1 deve ser uma especialização do padrão P2.

- Se um padrão P1 ***requer*** um padrão P2, então aplicação do padrão P2 é exigida na aplicação de P1.

- Se um padrão P1 é ***alternativo*** a um padrão P2, então os padrões P1 e P2 fornecem soluções diferentes para o mesmo problema (o relacionamento

alternativo é apresentado em [GERB 99] como o relacionamento ***É Conflitante*** (Conflicts)).

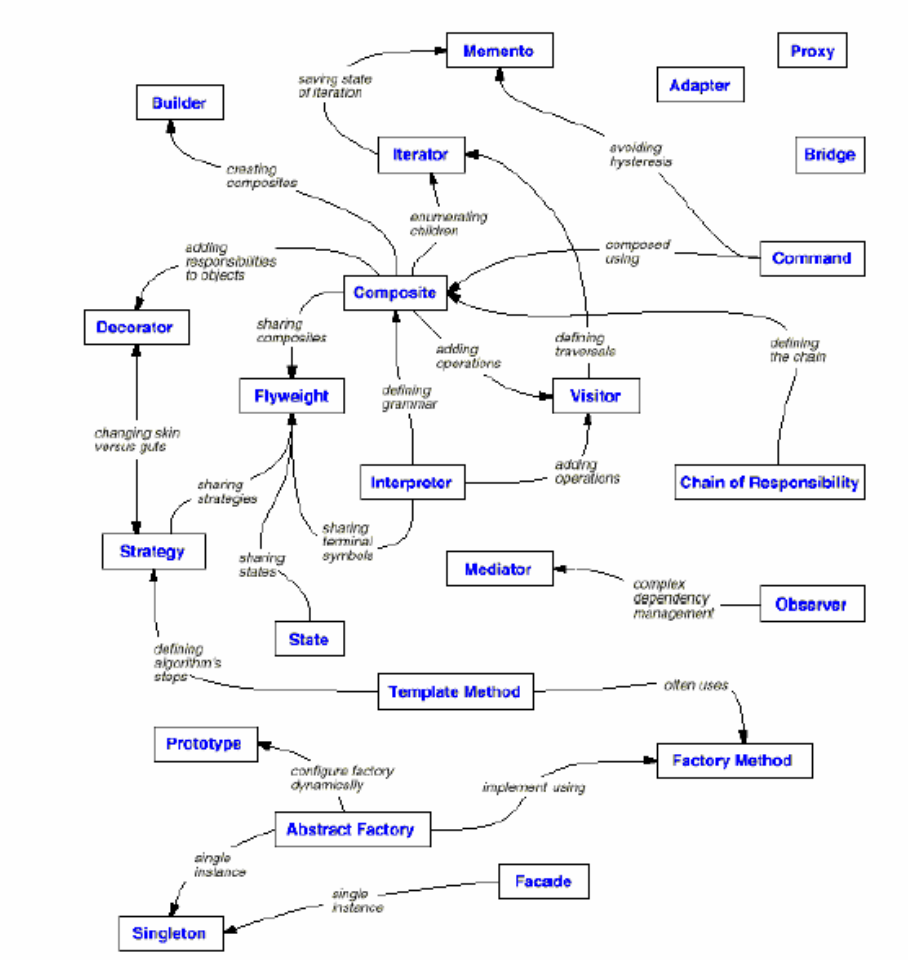

Figura 4: Relacionamentos entre Padrões da Gang of Four.

2.7. Padrões de Análise (*Analysis Pattern)*

Diferentemente dos padrões de projeto, os padrões de análise não descrevem implementações em software propriamente ditas. Descrevem, isto sim, modelos de objetos de negócio os quais resultam repetidamente da fase de análise e modelagem no desenvolvimento de software. De forma geral, são apresentados de forma genérica o que contribui para torná-los mais aplicáveis a diferentes domínios, tais como finanças, manufatura, saúde, etc [DEVE 02].

Na última década, os padrões de análise emergiram como uma técnica promissora para melhorar a qualidade e reduzir o custo e o tempo de desenvolvimento do software [SOMM 04][GAMM 95]. Um padrão de análise pode geralmente ser definido como: "*Uma idéia que fosse útil em um contexto prático e seja provavelmente útil em outro*" [FOWL 97].

Já em 1992, Peter Coad observava que as atividades de análise e projeto orientados a objeto apresentavam padrões recorrentes [COAD 92]: "AOO (*Análise orientadas a objetos) e* POO (*projeto orientado a objetos) utilizam-se de classes e objetos como seus menores blocos de construção. Estas classes e objetos formam padrões com relacionamento específico entre eles. Grupos de classes em um ambiente orientado a objetos tendem a ser utilizados repetidamente. Muitos padrões podem ser encontrados por tentativa e erro e por observação. Exemplos de tipos de padrões incluem a descrição de um item, sua associação com o tempo, log de eventos, os papéis exercidos, o estado em uma coleção, e a difusão. Padrões padronizam (standardize) pequenas peças de trabalho em unidades maiores que se tornam blocos para o desenho e a construção de problemas".*

Segundo [ALEX 77], um padrão de análise é qualquer parte de uma especificação de requisitos originados em um projeto e que possa obter reuso em um ou em mais projetos. Os projetos da análise começam lentamente porque o analista inicia com uma folha de papel limpa a rabiscar os requisitos do sistema. O analista poderia ter em mãos no início do projeto uma coleção de produtos potencialmente reusáveis da análise. Poderia o analista pensar nos padrões de análise como um livro de consulta.

Para descrever a estrutura dos padrões de análise, também se utiliza de notações gráficas em uma linguagem de modelagem, como nos padrões de projeto. Utiliza-se, além disso, de exemplos para ilustrar a aplicação dos padrões.

Geralmente os padrões de análise são descritos em grupos, sendo que muitas vezes os grupos estão associados a domínios, tais como telecomunicações, saúde, finanças, dentre outras, podendo ser aplicados a mais de um domínio, ou até mesmo a diversos domínios.

Na Figura 6 são apresentadas duas tarefas principais onde padrões de análise contribuem no processo do desenvolvimento de software. Primeiramente, os padrões de análise aumentam a velocidade do desenvolvimento dos modelos abstratos de análise que capturam os requisitos principais do problema concreto, fornecendo assim modelos de análise reusáveis com exemplos e descrição das vantagens e das limitações. Em segundo, os padrões

de análise facilitam a transformação do modelo da análise em um modelo de projeto sugerindo padrões de projeto e soluções de confiança para problemas comuns.

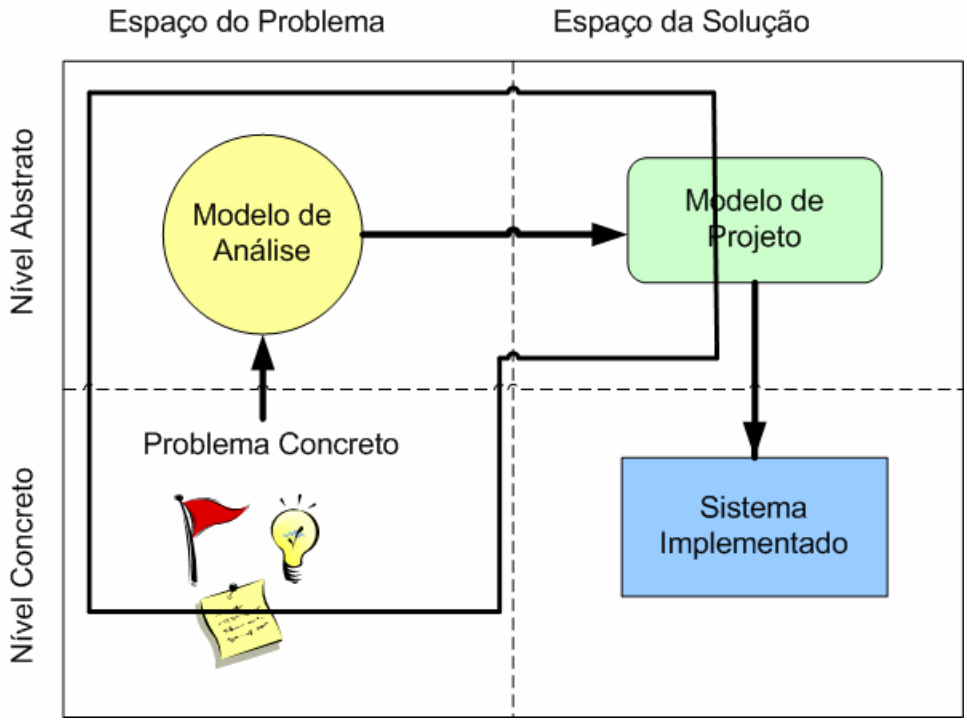

Figura 5: Padrões de Análise no desenvolvimento de *software*.

Segundo [HASL 02], Fowler prefere e usa um formato muito livre e informal para a escrita do padrão de análise. Adota um formato uniforme e consistente para descrever padrões da análise. Na Tabela 1 é apresentado um molde proposto por Michael para descrever padrões de análise.

Tabela 1:Quadro de campos para documentação de padrões de análise.

Nome do Padrão	[GAMM 95] [BYSC 96]	O nome do padrão expressa a essência de um padrão precisamente. É iniciado com partes do vocabulário usado nas análises.
Intenção	[GAMM 95]	O que o padrão de análise faz e qual o problema a ele relacionado.
Motivação	[GAMM 95]	Um cenário que ilustra o problema e como o padrão de análise contribui para a solução do cenário concreto.
Forças e Contexto	[ALEX 77]	Discussões de forças e intenções as quais deveriam ser resolvidos pelo padrão de análise
Solução	[BYSC 96]	Descrição da solução e do nivelamento das forças conseguidas pelo padrão de análise por um cenário na seção de motivação. Inclui todas as estruturas comportamentais relevantes.
Conseqüências	[GAMM 95] [BYSC 96]	Como o padrão descoberto atingiu os objetivos e como fez.
Projeto		Como o padrão de análise pode ser realizado em Padrões de Projeto? Simples sugestões de projetos.
Usos Comuns	[GAMM 95] [BYSC 96]	Exemplos do padrão encontrado em sistemas reais.

2.8. RUP – Rational Unified Process

O RUP é um framework de Processos Organizacionais, que ao ser usado corretamente, pode garantir o sucesso da área de engenharia de *software* [RATI 06]. É um *guideline* genérico que sugere uma metodologia de desenvolvimento de software, ferramentas, processos e cobre as principais fases de desenvolvimento: levantamento de requisitos, detalhamento de requisitos, design, construção, testes e implantação [RATI 05]. Este é um processo de desenvolvimento iterativo e incremental e o software é desenvolvido e implementado em partes. Segundo [RATI 06] o RUP:

- Define um processo genérico de negócios para Engenharia de Software Orientado a Objetos;
- Compartilha estruturas comuns para processos de Engenharia de Software de mesma família.

- Provê um controle disciplinado de tarefas e responsabilidades no desenvolvimento de sistema.

- Auxilia na implementação das seis boas práticas: desenvolvimento iterativo; gerenciamento de requisitos; arquitetura componentizada; modelagem visual; verificação contínua de qualidade; controle de mudanças.

O processo RUP é totalmente iterativo e incremental, conforme a Figura 7, o RUP possui duas dimensões. O eixo horizontal apresenta o aspecto dinâmico do processo e mostra aspectos do ciclo de vida à medida que este se desenvolve. Já o eixo vertical representa o aspecto estático do processo, como ele é descrito em termos de componentes, disciplinas, atividades, fluxos de trabalho, artefatos e papéis de processo [JACO 02].

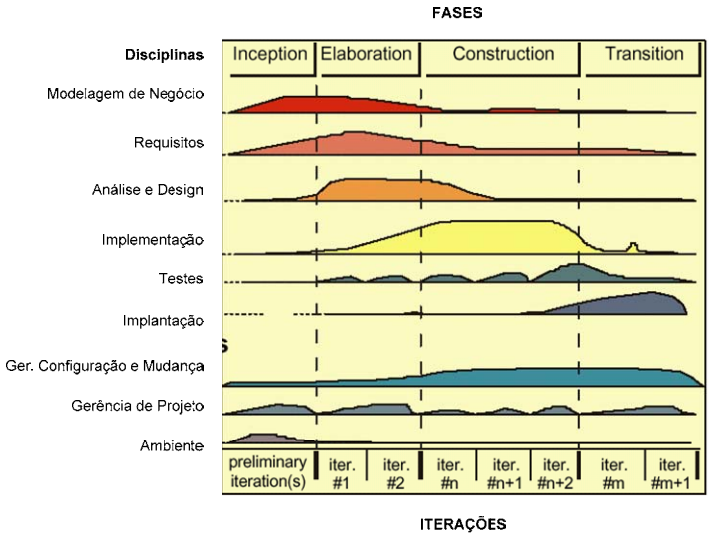

Figura 6: Ciclo de vida de desenvolvimento do RUP

2.8.1. **Fases/Iterações**

O ciclo de vida de software no RUP é dividido em quatro fases seqüenciais: iniciação, elaboração, construção e transição [RATI 05]. As fases de Iniciação e Elaboração são primordiais para as descobertas de padrões de análise. O engenheiro de *software* ao realizar especificações funcionais e gerar artefatos nestas fases pode através de um *template* documentar a coleção de requisitos que foram transformados em um padrão, assim obtendo reusabilidade dentro do processo RUP.

1. *Inception* (Iniciação) – Define o escopo do projeto, o que é incluído e o que não é incluído. Nesse momento são identificadas todas as entidades externas (atores) com as quais o sistema irá interagir. Isto envolve a identificação de todos os casos de uso e uma breve descrição dos mais significativos. É definido um plano de negócios, que inclui critérios de sucesso, avaliação de risco, estimativa de recursos necessários ao desenvolvimento do projeto e um plano de fases indicando as datas dos principais pontos de controle (major milestones).

2. *Elaboration* (Elaboração) – A elaboração é focada na compreensão dos requisitos do sistema e na definição de uma arquitetura base. Nesse momento é possível chegar a 80% dos casos de uso detalhados. Nesta fase, um protótipo de arquitetura executável é construído em uma ou mais iterações – dependendo do escopo, tamanho e risco do projeto. Demais resultados desta fase: Descrição da arquitetura de software, lista de riscos e plano de negócios revisados.

3. *Construction* (Construção) – Durante esta fase, um produto completo é desenvolvido de forma iterativa e incremental, até gerar uma versão beta. Isto implica em descrever os requisitos remanescentes e os critérios de aceitação, completando a implementação e o teste de software. Demais resultados desta fase: Produto de Software Integrado às plataformas adequadas; Manuais de Usuário; Descrição da Versão Corrente.

4. Transition (Implantação) – Implantação do sistema para o usuário final focado em treinamento, instalação e suporte. Uma vez que o produto tenha sido entregue ao usuário final freqüentemente surgem questões que requerem o desenvolvimento de novas versões, correção de alguns problemas ou a finalização de características que foram postergadas.

A figura 8 apresenta em detalhes o ciclo de vida das fases do RUP e também os Milestones.

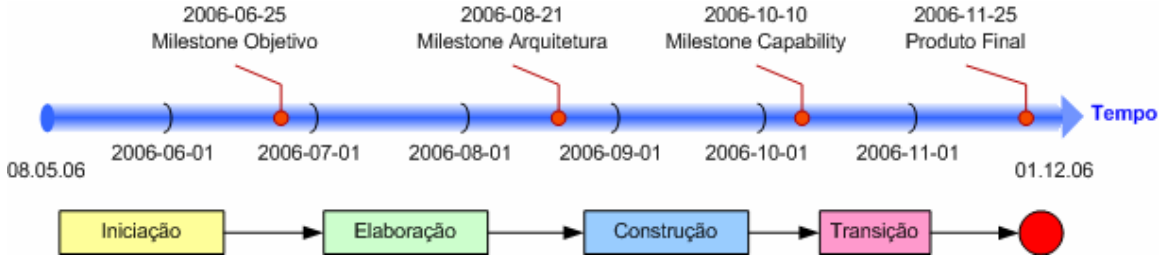

Figura 7: Fases e Milestones do RUP em exemplo do presente trabalho.

O tempo utilizado em cada fase pode variar conforme a complexidade do sistema, cada fase pode conter uma ou mais iterações. Uma iteração representa uma seqüência distinta de atividades baseadas em um plano previamente estabelecido e um critério de avaliação, resultando em uma versão final interna ou externa.

Para desenvolvimento interno, cada versão pode ser apresentada aos chamados usuários chave (*stakeholder*). Para desenvolvimento externo esse procedimento é mais complexo, ocorrendo, na maioria das vezes, durante a fase de Implantação (*Transition*).

No final do ciclo de vida da fase de transição, os objetivos devem ter sido atendidos e o projeto deve estar em uma posição para fechamento. Em alguns casos, o fim do ciclo de vida atual pode coincidir com o início de outro ciclo de vida no mesmo produto, conduzindo à nova geração ou versão do produto. O sistema de repositório de padrões de análise deverá ter futuras manutenções de correções e disponibilidade para novas características a serem descobertas. As alterações deverão gerar novas documentações que façam com que o processo seja realizado novamente após a descoberta do novo requisito.

Segundo [JACO 02], a iniciação e a elaboração abrangem as atividades de engenharia do ciclo de vida do desenvolvimento. A construção e a transição constituem na sua produção. Como mostrado na Figura 8, em cada fase podem ocorrer várias iterações. Uma iteração representa um ciclo completo de desenvolvimento. O gráfico mostra como a ênfase varia através do tempo. Por exemplo, nas iterações iniciais, é dedicado mais tempo aos requisitos. Já nas iterações posteriores, é gasto mais tempo com implementação.

2.8.2. **Pontos de controle (milestones)**

Os pontos de controle são usados para avaliar e identificar possíveis falhas no desenvolvimento de software. São divididos em dois tipos: *major milestone* e *minor milestone.*

Os pontos de controle menores (*minor milestone*) marcam o final de cada iteração independente da fase de desenvolvimento. Nesse momento são avaliados os resultados obtidos durante a iteração, revisando planos futuros conforme a necessidade.

Os pontos de controle principais (*major milestone*) marcam o final de cada fase. A cada *major milestone* é feita uma revisão geral do projeto. De acordo com o resultado da

revisão o projeto pode ser continuado como planejado inicialmente, pode ser abortado ou reavaliado. Os critérios usados para tomar essas decisões variam para cada fase.

O LCO (*Lifecycle objetive*) marca o final da fase de iniciação, caracterizando a concepção inicial do sistema. Os critérios de avaliação incluem: usuário chave participando da definição do escopo e estimativas de custo e tempo; entendimento dos requisitos baseado nos casos de uso primário gerado; credibilidade das estimativas de custo e tempo; prioridades definidas; risco;

O LCA (*Lifecycle architecture*) marca o final da elaboração, definindo a arquitetura do sistema. Os critérios utilizados são:

- clareza quanto à visão do produto e da arquitetura utilizada; resolução dos riscos mais significativos;
- planejamento adequado para a continuação do projeto; aceite do usuário chave pela definição do produto e plano de desenvolvimento;
- Nível de despesa aceitável.

O IOC (*Initial Operational Capability*) marca o final da construção. Os critérios são: estabilidade e maturidade do produto, usuário chave pronto para a implantação e nível de despesa aceitável.

Na transição (*PR – Product Release*) deve-se medir o grau de satisfação do usuário. Freqüentemente os pontos de controle coincidem com o início de outro ciclo de desenvolvimento.

2.8.3. **Disciplinas**

O RUP organiza o conteúdo através de seis disciplinas principais chamado *workflow* base:

Modelagem de Negócio, Requisitos, Análise e Design, Implementação, Teste e Utilização. Existem algumas disciplinas complementares chamadas de *workflow* de apoio:Gerenciamento de mudanças e configuração, Gerenciamento de projetos, Suporte ao Desenvolvimento.

Cada disciplina é associada a um ou mais modelo e este gera um ou mais documentos. Os documentos principais são os modelos que cada disciplina gera: use-case, design, implementação e modelo de teste, como apresentado na Figura 9.

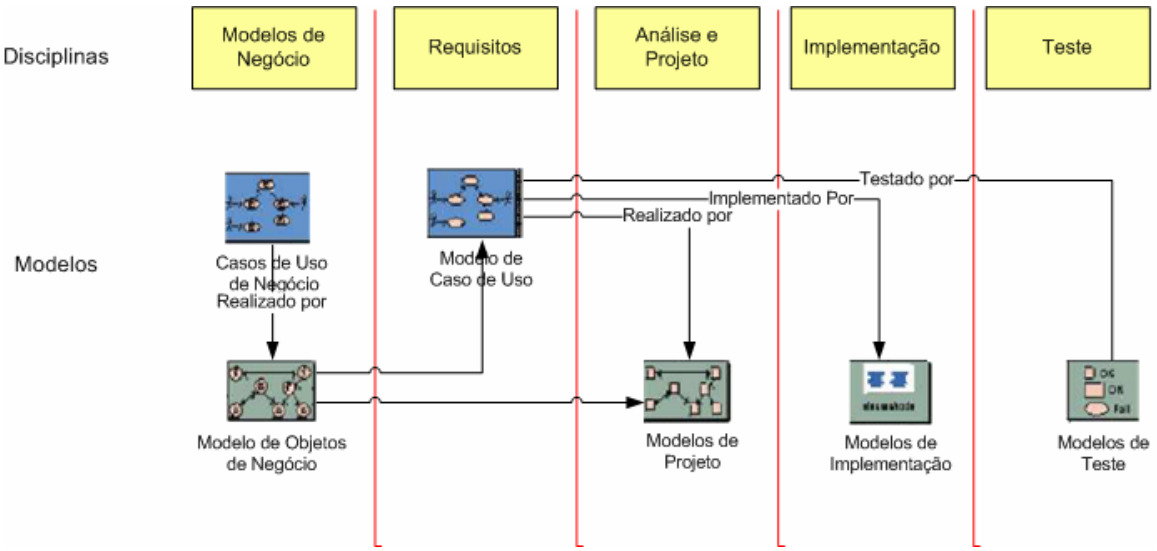

Figura 8: *Workflows* e Modelos.

Business Modeling (Modelagem de Negócio) – A intenção dessa disciplina é desenvolver um modelo de negócios. A idéia é melhorar o entendimento do negócio. Os documentos gerados são use cases de negócio e modelo de objetos de negócio.

- *Requirements* (Requisitos) – Possibilita um melhor entendimento dos requisitos do sistema. O intuito é definir um acordo com o cliente bem como oferecer uma orientação aos desenvolvedores. Nesse ponto é produzido um modelo de caso de uso detalhado e um protótipo do sistema.

- *Analysis and Design* (Análise e Design) – Os requisitos capturados na disciplina anterior são transformados em *design*. Modelos de analise e *design* são gerados.

- *Implementation* (Implementação) – Nessa disciplina, o design é transformado em codificação. A estratégia é desenvolver o sistema em camadas, particionando em subsistemas. O resultado final é componente testado que faz parte do produto final.

- *Test* (Teste) – Como o próprio nome diz, essa disciplina é responsável pela elaboração de testes além de verificar se todos os requisitos foram cumpridos. Os documentos gerados são os modelos e casos de testes.

- *Deployment* (Utilização) – Torna o produto disponível para o usuário final. Responsável pelo empacotamento do produto, instalação, treinamento ao usuário e distribuição do produto.

2.8.4. **Regras/Atividade/Artefatos**

Dentro da metodologia RUP existem papéis chamados de *Worker*, que define comportamento e responsabilidades de um indivíduo ou grupo de indivíduos trabalhando juntos como um time. Sendo assim, uma atividade é um trabalho de responsabilidade de um *Worker* que pode ser realizado a cada iteração, se necessário. Essas atividades são definidas através de regras abstratas.

Artefatos são produtos utilizados durante o desenvolvimento do projeto. Os artefatos podem ser documentos (relatório de riscos, plano de testes, plano de arquitetura), modelos (modelos de use case, modelos de projeto, modelos de arquitetura) e modelo de elementos (diagramas de classes, pacotes).

Como já visto, os artefatos são organizados por disciplinas, ou seja, cada disciplina produz um conjunto de artefatos.

A Figura 10 apresenta um modelo com os principais artefatos do RUP e o fluxo de informação entre eles.

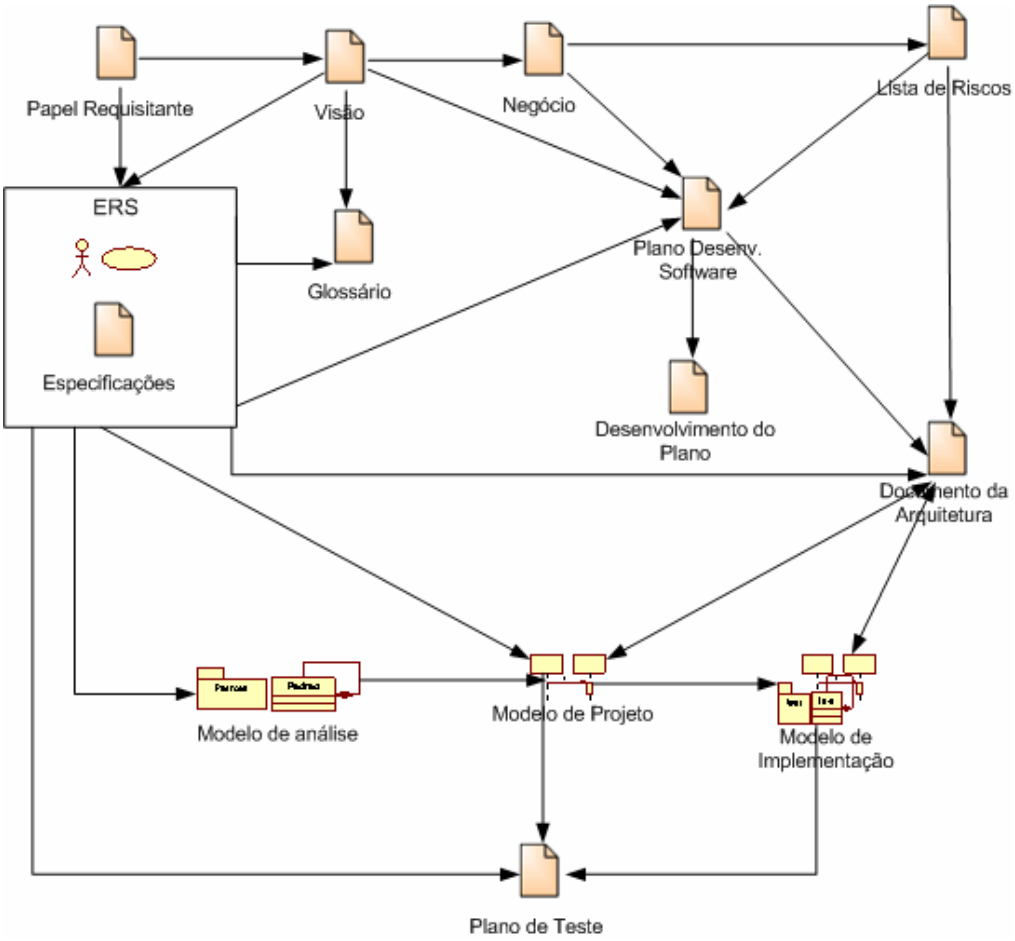

Figura 9: Fluxo de artefatos no RUP.

2.9. Web Services

Web Services podem ser vistos como componentes de software, independentes de implementação ou plataforma, que podem ser: descritos utilizando uma linguagem de descrição de serviços; publicados em um "diretório" de serviços; encontrados através de mecanismos de busca padronizados; que são invocados através de uma API, via rede; combinados com outros serviços [OELL 01, HANS 02, NEWC 02, HANS 03].

Segundo definição do W3C, um *Web Service* é: "uma aplicação de software identificado por um URI cujas interfaces e ligações são capazes de ser definidas, descritas e descobertas como artefatos XML. Um serviço *Web* suporta interações diretas com outros Agentes de software usando mensagens baseadas em XML, trocadas via protocolos baseados na Internet" [WORL 04].

De forma mais simplificada, pode-se dizer que um *Web Service* é um programa disponibilizado na *Web*, que pode ser utilizado por alguma aplicação que estejamos desenvolvendo. Para fazer a chamada do serviço, precisamos saber o endereço onde este programa está, os parâmetros de que ele necessita e o que ele pode retornar. Se já conhecemos essas informações, podemos escrever a chamada diretamente no código de nossa aplicação; caso contrário, podemos fazer com que nossa aplicação consulte uma base de serviços e descubra como utilizar o programa desejado.

Web Services combinam os melhores aspectos do desenvolvimento baseado em componentes na *We*b. Assim, como componentes de software, *Web Services* representam uma funcionalidade black box que pode ser reutilizada sem a preocupação com a linguagem e o ambiente utilizados em seu desenvolvimento [HANS 03].

Devido aos benefícios originados da XML, *Web Services* propiciam ligação dinâmica de serviços e aumentam a interoperabilidade entre linguagens e plataformas [FERR 03].

Web Services são construídos para serem acessados por outras aplicações [OELL 01]. Este acesso não é feito via protocolos específicos de modelos de objetos (DCOM, RMI, IIOP), mas sim via protocolos de transporte (HTTP, FTP SMTP) [HANS 02, HANS 03].

Uma forma mais primitiva de *Web Service* é mostrada em [NEWC 02]:

http://internal.iona.com:8080/iona/phonelist.jsp?search=vinoski

Neste tipo de serviço, dados de entrada são passados como parâmetro no endereço da página *Web* que implementa o serviço. O exemplo é de uma função que retorna um telefone e endereço de e-mail a partir do nome fornecido.

Hoje, *Web Services* implicam no uso de uma arquitetura, padrões de tecnologias e diversos recursos para desenvolvimento.

2.9.1. **Arquietura de Web Services**

A arquitetura de Web Services está baseada nas interações de três papéis: provedor, solicitante e registro de serviço [OELL 01, HANS 02, NEWC 02, FERR 03, HANS 03]. A Figura 11 ilustra as interações entre esses papéis.

Figura 10: Interações entre papéis na arquitetura de *Web Services*.

O provedor de serviço é a plataforma acessada na solicitação do serviço. É a entidade que cria o *Web Service*, sendo responsável por fazer sua descrição em algum formato padrão e publicar os detalhes em um registro de serviço central.

O registro de serviço é o local onde os provedores publicam as descrições dos serviços. Uma descrição de serviço torna possível descobrir onde está um *Web Service* e como invocá-lo: o provedor cria uma descrição que detalha a interface do serviço, ou seja, suas operações e as mensagens de entrada e saída para cada operação; uma descrição de ligação é então criada, mostrando como enviar cada mensagem para o endereço onde o *Web Service* está localizado.

O solicitante de serviço é uma aplicação que invoca ou inicializa uma interação com um serviço. Pode ser um browser ou um programa sem interface com o usuário como, por exemplo, outro *Web Service*. Um solicitante de serviço encontra uma descrição de serviço, ou consulta o registro de serviço para o tipo de serviço requerido, e obtém as informações de ligação da descrição do serviço durante a fase de desenvolvimento (ligação estática) ou em tempo de execução (ligação dinâmica).

A execução das interações entre os papéis acontece via rede, suportada por um conjunto de tecnologias padronizadas. A Figura 12 traduz esse cenário para um conjunto de camadas conceituais.

Figura 11: Camadas conceituais de *Web services*, adaptado de [HANS 02]

A rede é a camada base. Ela abrange os protocolos de transporte citados anteriormente (HTTP, FTP, SMTP) e pode ser utilizada para implementação de necessidades das aplicações, tais como: disponibilidade, performance, segurança e confiabilidade. Mensagem baseada em XML é a cama de comunicação. Ela utiliza o protocolo SOAP para realizar a troca de mensagens entre provedor, registro e solicitante. A descrição de serviços é feita com uma linguagem específica: a WSDL. As camadas de publicação e descoberta de serviços usam UDDI para tornar *Web Services* disponíveis [HANS 02, FERR 03, HANS 03].

2.9.2. **Tecnologias**

Por trás da estrutura básica de *Web Services*, existem algumas tecnologias padrão para sua construção: WSDL *(Web Services Description Language*), SOAP (*Simple Object Access Protocol*) e UDDI (*Universal Description, Discovery and Integration*). Estas tecnologias são baseadas em XML e permitem reutilização e chamada dos serviços sem que se conheça sua linguagem ou plataforma.

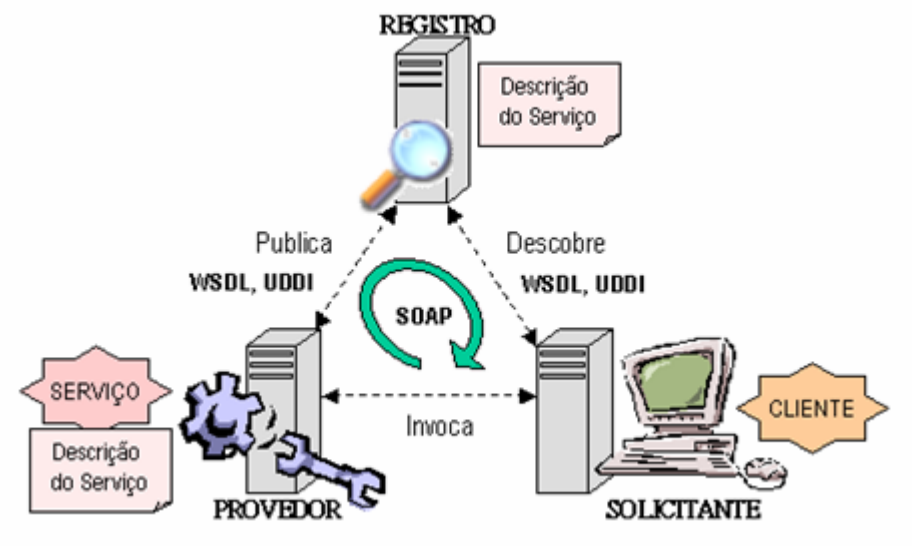

Figura 12: Papéis e tecnologias relacionadas.

A Figura 13 relaciona as tecnologias com os papéis previamente apresentados.

2.9.3. **WSDL (*Web Service Description Language)***

A linguagem WSDL descreve um serviço como uma coleção de operações que podem ser acessadas através de mensagens. O WSDL funciona como uma espécie de contrato que o cliente se compromete para utilizar o serviço. Utilizando seus métodos, é possível descrever um serviço de forma transparente e independente de implementação. As duas partes envolvidas em uma interação de *Web Services* precisam ter acesso à mesma descrição WSDL para conseguirem entender uma à outra [HANS 02, NEWC 02, HANS 03].

A descrição de um serviço consiste de duas partes, mostradas na Figura 13 definição da implementação do serviço e definição da interface do serviço. A WSDL também define protocolos de ligação e detalhes de rede. Ela apresenta descrições adicionais como contexto, QoS (*Quality of Service*) e relacionamento entre serviços. A separação entre as duas definições permite que elas sejam utilizadas separadamente [HANS 02, HANS 03].

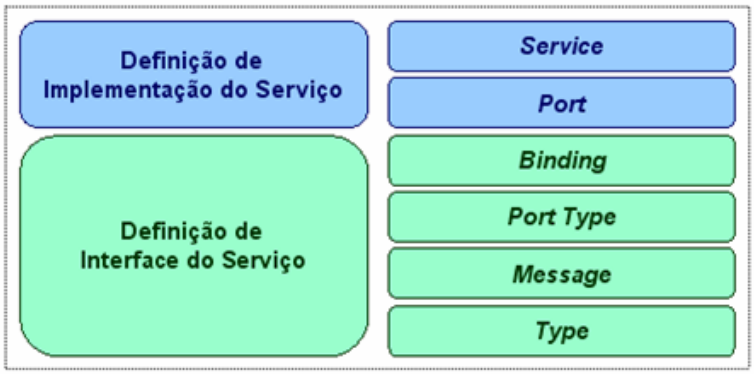

Figura 13: Detalhamento da camada de descrição de serviços, adaptada de [Hans 2002].

A parte de definição de interface do serviço descreve o *Web Service*, incluindo métodos que são invocados e parâmetros que são enviados. Pode ser usada, instanciada e referenciada por múltiplas definições de implementação de um serviço, e consiste de algumas diretivas [HANS 02, NEWC 02]:

- WSDL:*binding* – descreve protocolos, formato de data, segurança e outros atributos para uma interface (*portType*) em particular;
- WSDL:*portType* – informa elementos de operações do *Web Service*;
- WSDL:*message* – define entrada e saída de dados referentes a operações. Pode assumir a forma de um documento inteiro ou de argumentos que devem ser mapeados para invocação de métodos;
- WSDL:*type* – define tipos de dados complexos em uma mensagem.

A parte de definição de implementação do serviço descreve como uma interface de serviço é implementada por um provedor: onde o serviço está instalado e como pode ser acessado. A definição de um serviço (WSDL:*service*) contém uma coleção de elementos WSDL:*port* com um elemento WSDL:*binding*. Pode conter definições de extensibilidade. WSDL:port é a combinação de um WSDL:*binding* com um endereço de rede, fornecendo o endereço alvo para comunicação com o serviço [HANS 02, NEWC 02].

O trecho a seguir define um exemplo simplificado de uma descrição WSDL.

```
<message name="getTermRequest">
   <part name="term" type="xs:string"/>
</message>
```

```
    <message name="getTermResponse">
       <part name="value" type="xs:string"/>
    </message>

<portType name="glossaryTerms">
  <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
  </operation>
</portType>
```

Cada elemento *message* define as partes de uma mensagem e os tipos de dados associados, ou seja, define os elementos de dados de uma operação. No exemplo, temos a definição de uma mensagem nomeada *getTermRequest*, que possui um elemento chamado *term*, do tipo *string*. Temos, também, a definição de uma mensagem nomeada *getTermResponse*, que possui um elemento chamado *value*, do tipo *string*. Em comparação com a programação tradicional, é como se tivéssemos uma função *getTermRequest* com o parâmetro *term* e uma função *getTermResponse* com o parâmetro *value*.

O elemento *portType* define o *Web Service*, as operações que podem ser realizadas e as mensagens que estão envolvidas. No exemplo, *glossaryTerms* é definido como um elemento de operação do *Web Service*, onde *getTerm* é o nome de uma operação. A operação *getTerm* tem uma mensagem de entrada chamada *getTermRequest* e uma mensagem de saída chamada *getTermResponse* (definidas anteriormente no documento XML, através dos elementos *message*). Fazendo uma comparação com a programação tradicional, *glossaryTerms* seria uma biblioteca de funções, e a operação *getTerm* seria uma função com *getTermRequest* como parâmetro de entrada e *getTermResponse* como parâmetro de retorno.

A WSDL também especifica extensões relativas a protocolos e formatos de mensagem como SOAP, HTTP GET/POST e MIME (*Multipurpose Internet Mail Extensions*) [HANS 02].

2.9.4. **SOAP (*Simple Object Access Protocol*)**

O protocolo SOAP permite comunicação entre diversas aplicações em um ambiente distribuído e descentralizado. Como o formato das mensagens é baseado em XML, ele pode ser entendido por quase todas as plataformas de *hardware*, sistemas operacionais, linguagens de programação e equipamento de rede. Este protocolo é utilizado para publicar, localizar e invocar *Web Services*. Suporta RPC e pode trabalhar com protocolos como HTTP e SMTP. Ainda, possui definição de tipos de dados para as estruturas mais comumente utilizadas, como *string*, *integer*, *float*, *double* e *date* [HANS 02, HANS 03].

Um pacote SOAP consiste de quatro partes [HANS 02, HANS 03]:

- Envelope – guarda o conteúdo da mensagem, quem pode processá-la e o quão obrigatório é fazer esse processamento. É uma estrutura que encapsula elementos sintáticos da mensagem;
- Codificação – define mecanismos de serialização que podem ser utilizados para trocar instâncias ou tipos de dados definidos por uma aplicação;
- RPC – especifica como encapsular chamadas remotas de métodos e respostas dentro da mensagem;
- *Framework* de ligação e transporte – define um *Framework* abstrato para troca de envelopes SOAP entre aplicações utilizando um protocolo de transporte simples.

Outros conceitos importantes são apresentados em [HANS 02, HANS 03]:

- Servidor SOAP – responsável por executar uma mensagem SOAP, agir como um interpretador e realizar trocas de mensagens;
- Mensagem SOAP – consiste de um envelope, contendo cabeçalhos opcionais, e um corpo, contendo uma mensagem atual com seus parâmetros ou resultados. Esta estrutura e a navegação entre nodos são mostrados na Figura 15.

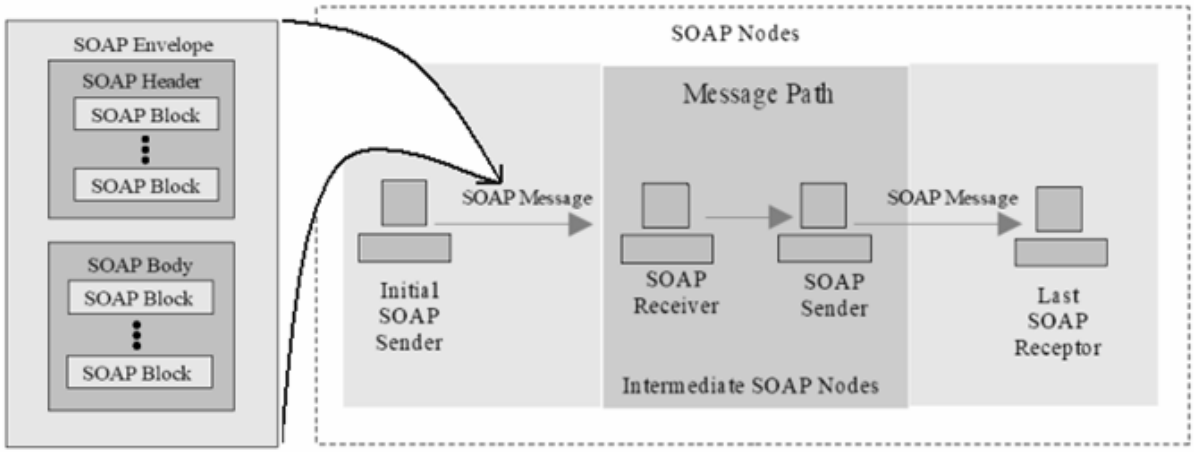

Figura 14: Mensagem SOAP e a navegação entre nodos, adaptado de [HANS 02].

A mensagem SOAP constitui, logicamente, uma única unidade computacional. O bloco é identificado por um elemento externo chamado *namespace*.. O cabeçalho SOAP é uma coleção de zero ou mais blocos, os quais podem ser direcionados para um determinado receptor SOAP dentro do caminho da mensagem. O corpo SOAP é uma coleção de zero ou mais blocos direcionados para o último receptor SOAP. O protocolo SOAP não garante roteamento, apenas sabe qual o nodo que criou a mensagem e qual deve ser o último receptor da mesma, através de zero ou mais nodos intermediários. Quando um nodo recebe a mensagem, deve processá-la e gerar as mensagens adequadas – de sucesso, falha, ou outras adicionais [HANS 02, HANS 03].

2.9.5. **UDDI (*Universal Description, Discovery And Integration*)**

Para publicar ou encontrar um serviço, é necessário acessar um registro UDDI, que roda em um servidor, funcionando como um grande catálogo de serviços.

A especificação UDDI é um trabalho conjunto para criação de um registro de serviços padronizado. Ela possui um componente central chamado UDDI *Project*, que manipula um registro global e público chamado *business registry*. A informação oferecida pelo *business registry* consiste de três componentes [HANS 02, NEWC 02, HANS 03]:

- *White pages* – endereço, contato e identificadores conhecidos;
- *Yellow pages* – categorização industrial;
- *Green pages* – informação.

A implementação UDDI é um servidor de registro que fornece um mecanismo para publicar e localizar serviços. Guarda informações categorizadas sobre empresas, serviços que elas oferecem e a associação com as especificações desses serviços (feitas em WSDL através do próprio registro) [HANS 02].

Registros podem ser públicos, acessados via *Internet*, ou privados, acessados em *Intranets* de empresas, por exemplo. Registros UDDI podem ser acessados tanto por aplicações, diretamente via código, quanto por pessoas, através de alguma interface. A Figura 16 mostra a interface *Web* do servidor de registro UDDI da IBM.

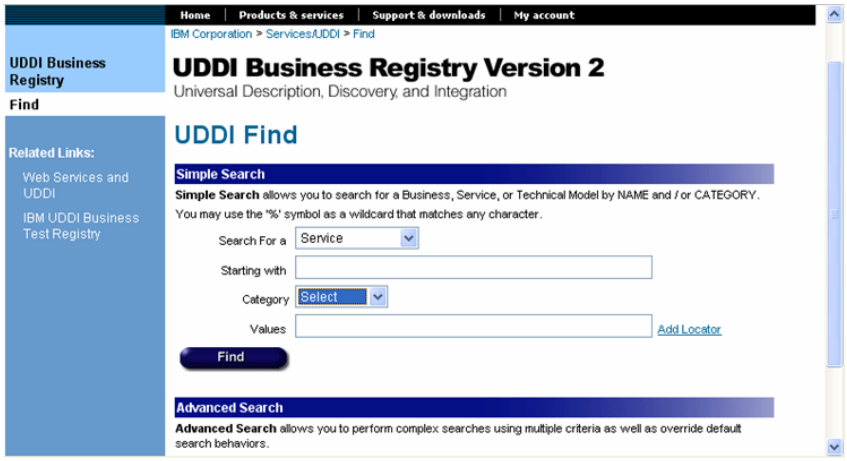

Figura 15: Interface *Web* do servidor de registro UDDI da IBM.

As classificações para categorização das informações em um registro UDDI incluem as seguintes [NEWC 02]:

- NAICS (*North American Industry Classification System*);
- UNSPSC (*Universal Standard Products and Services Classification*);
- ISO (*Internacional Organization for Standardization*).

O modelo de informação principal utilizado pelo registro UDDI é definido através de XML *Schema*, englobando quatro tipos de informação: sobre o negócio, sobre o serviço, de ligação e específica do serviço [HANS 02, HANS 03].

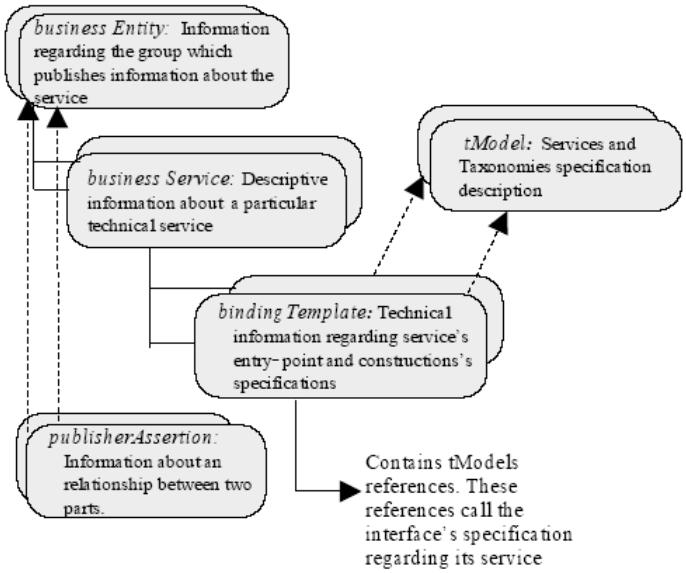

Figura 16: Estrutura UDDI [Hans 2002].

Alguns tipos de estruturas de dados, mostradas na Figura 17, também compõem o registro. São elas [HANS 02, NEWC 02, HANS 03]:

- *businessEntity* – estrutura de alto nível que representa toda a informação conhecida sobre uma empresa específica ou entidade que publica informação descritiva, bem como serviços;

- *businessService* – representa uma classificação lógica do serviço. Cada estrutura *businessService* pertence a uma única estrutura *businessEntity*;

- *bindingTemplate* – estruturas deste tipo são descrições técnicas sobre *Web Services* registrados. Fornecem suporte para que se possa acessar os serviços remotamente, e descrevem como a estrutura *businessService* utiliza várias informações técnicas;

- *tModel* – é representado através de metadados e tem por objetivo fornecer um sistema de referência;

- *publisherAssertion* – permite que se possa associar estruturas *businessEntity*, de forma a obter uma melhor identificação. Cada uma das partes relacionadas precisa concordar que o relacionamento entre elas é válido e precisam publicar exatamente a mesma informação, mantendo seu relacionamento visível.

2.10. Ontologia

Gruber foi quem introduziu o termo na área da Inteligência Artificial, motivado por razões como compartilhamento e reuso do conhecimento, tendo a idéia de especificação explícita de uma conceitualização [GRUB 93].

Segundo [BENJ 99][DECK 99][STAA 00], a construção de ontologias comuns tem sido proposta como abordagem promissora para a interoperabilidade de sistemas. Ontologia é, usualmente, definida como "*a especificação explícita de uma conceitualização*" [BENJ 99]. Uma ontologia comum é uma formalização compartilhada de um certo domínio de aplicação. Assim, por exemplo, uma formalização de conceitos sobre informações em ciência e tecnologia poderia permitir que diversos aplicativos deste domínio compartilhassem um vocabulário comum sobre o assunto.

A interoperabilidade é definida como a "interconexão efetiva de diferentes sistemas de computador, banco de dados ou redes com a finalidade de apoiar a computação distribuída e/ou o intercâmbio de informações". Pacheco [PACH 01] descreve que uma das barreiras da interoperabilidade é a incompatibilidade dos modelos de dados subjacentes à aplicações. Na área do desenho, por exemplo, é possível representar um arco circular de nove formas distintas. Dois aplicativos de desenho poderão trocar dados sobre arcos circulares somente se usarem a mesma forma de representação ou se dispuserem de uma forma explícita de tradução entre as duas formas distintas. Para resolver isso se faz necessário o desenvolvimento de uma ontologia para que se possa garantir a interoperabilidade.

Na Ciência da Computação, ontologia é um termo utilizado desde o início da década de 90 para representação computacional de conhecimento em áreas como engenharia de conhecimento e processamento de linguagem natural [CHAN 99].

2.10.1. **Por que desenvolver uma ontologia?**

Há alguns anos o desenvolvimento de uma ontologia tem focado o seu uso na Web. Em conseqüência, o W3C desenvolveu linguagens para codificação de conhecimento de maneira a torná-lo compreensível por agentes de software na busca de informação.

Muitas áreas têm desenvolvido ontologias padronizadas de domínios específicos objetivando o compartilhamento de informação.

Algumas razões para se desenvolver uma ontologia incluem:

- o compartilhamento e o entendimento de uma estrutura de informação por pessoas e agentes de *software*. Por exemplo, suponha diversos *Web Sites* médicos distintos contendo informações médicas ou provendo alguns serviços. Se estes sites compartilharem e publicarem sobre a mesma ontologia de termos, estes agentes poderão extrair e agregar informação de um site para outro;
- habilitar o reuso de um domínio de conhecimento.

A Figura 2 expressa o domínio de uma ontologia de parcelas, apresentando a idéia central do que se trata o domínio e assim começando o desenho da ontologia em classes e *slots*.

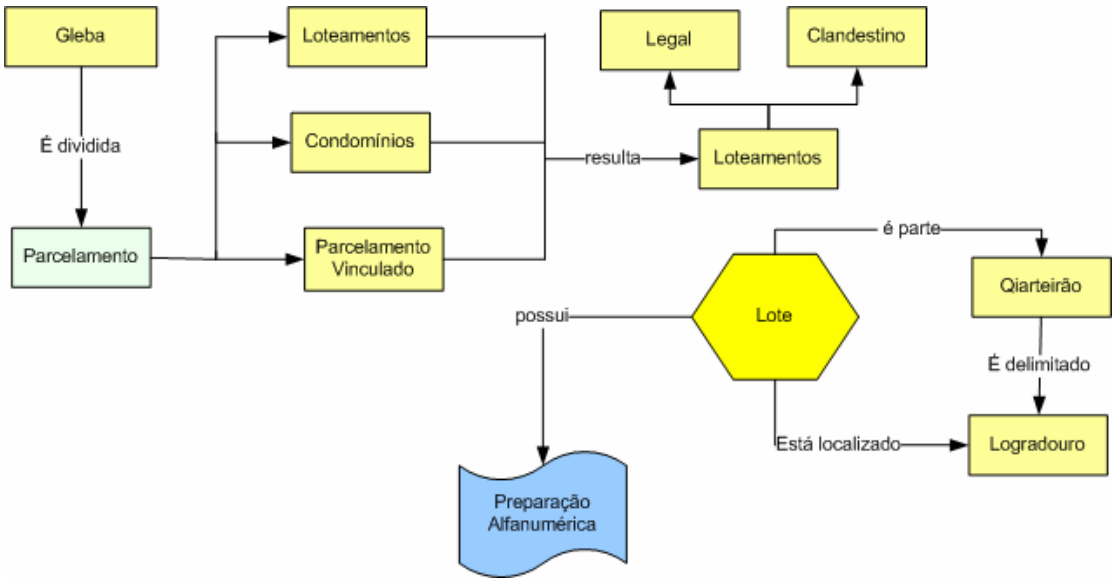

Figura 17: Exemplo de domínio da ontologia.

2.11. Detalhando uma Ontologia

Noy [NOYN 04] define uma ontologia como uma descrição formal de conceitos em um domínio de discurso, sendo composta por:

- **classes** (geralmente chamadas de Conceitos): descrevem conceitos de um domínio;
- ***slots*** (geralmente chamados de regras ou propriedades): descrevem as propriedades das classes e suas instâncias;

- **restrições*:*** definem propriedades específicas de um *slot*.

Na prática, o desenvolvimento de uma ontologia inclui:

- definir as classes da ontologia;
- organizar as classes em uma taxonomia hierárquica (subclasse – superclasse);
- definir os *slot*s e descrever quais os valores reservados para estes;
- definir valores para os *slot*s das instâncias.

As ontologias, geradas por um sistema, são resultados da integração de processos representados através de uma árvore, onde cada nó representa um conceito da ontologia, e cada relacionamento entre os nós pode ser um relacionamento taxonômico. Segundo Novello, taxonomias representam a maneira como se organizam classes e subclasses dentro de uma ontologia. Uma taxonomia é um sistema de classificação que agrupa e organiza o conhecimento num domínio usando relações de generalização/especialização através de herança simples/múltipla. Depois de criada, uma taxonomia apresenta a forma de uma árvore invertida em que o nó superior se chama raiz e os nós seguintes se chamam folhas. A relação existente entre um nó e os seus sub-nós é a de "tipo de" ou "é um". Assim, um possível nó ou uma subclasse do nó ou classe "Pessoas" seria "Estudante" ou "Professor" [NOVE 05].

2.12. Metodologia de desenvolvimento

Uma ontologia consiste no modelo da realidade de um domínio, portanto os conceitos nesta ontologia precisam refletir esta realidade. Noy [NATA 04] enumera alguns passos que devem ser seguidos no desenvolvimento de uma ontologia:

1. **Determinar o escopo:** esta etapa consiste em responder as seguintes questões: Qual o domínio que esta ontologia irá cobrir? Para que ela será utilizada? Para que tipos de perguntas as informações contidas nesta ontologia irão prover respostas? Quem usará e quem manterá esta ontologia? As respostas para estas questões podem mudar durante o processo de desenvolvimento da ontologia, mas com o tempo auxiliam na definição do escopo do modelo.
2. **Considerar o reuso de ontologias existentes:** reusar uma ontologia existente pode ser um requisito se o sistema precisa interagir com outras aplicações que já possuem uma ontologia ou um vocabulário controlado. Muitas ontologias já estão disponíveis no

formato eletrônico e podem ser importadas para o ambiente da ontologia que está sendo desenvolvido. Existem algumas bibliotecas de ontologias reutilizáveis na WEB e na literatura. Por exemplo, pode-se usar a biblioteca de ontologias Ontolingua (www.ksl.stanford.edu/*software*/ontolingua) ou a DAML (www.daml.org/ontologies).

3. **Enumerar os termos importantes desta ontologia:** inicialmente cria-se uma lista de termos sem se preocupar com a sobreposição de conceitos, relações entre seus termos ou propriedades que estes conceitos podem ter. Após este levantamento torna-se necessário desenvolver uma hierarquia de classes e definir as propriedades dos conceitos (*slot*s). Essas etapas são mais difíceis que a primeira e consistem na fase mais importante do processo de desenvolvimento de uma ontologia.
4. **Definir as classe e sua hierarquia:** para esta etapa combinam-se os processos de desenvolvimento *top-down* (inicia com a definição dos conceitos mais importantes do domínio e subseqüente especialização) e *botton-up* (definem-se classes mais específicas e depois se agrupam em conceitos gerais).
5. **Definir as propriedades das classes (*slot*s):** as classes sozinhas não provêem nenhuma informação que forneça as respostas levantadas durante o primeiro passo. Uma vez definidas algumas classes, é preciso definir a estrutura interna dos conceitos.
6. **Definir as restrições dos *slot*s:** um *slot* pode ter diferentes restrições que descrevem o tipo de valor, sua cardinalidade e outras características que os *slot*s podem ter. Por exemplo, o valor aceito para um *slot* "nome" é uma string. As restrições mais comuns descrevem:

    - cardinalidade: define quantos valores um *slot* pode ter;
    - tipo de valor: define que tipo de valor que o *slot* aceita, podendo ser: *string, number, boolean, enumerated* ou *instance;*
    - domínio e escala do *slot*: classes reservadas para *slot*s do tipo *instance* são freqüentemente chamadas de escala de um *slot*. As classes para qual um *slot* é anexado, são chamados de um domínio de um *slot*.

7. **Criar instâncias:** o último passo consiste na criação de instâncias individuais das classes na hierarquia. Definir uma instância individual de uma classe requer:
    - escolher a classe;
    - definir uma instância individual desta classe e;
    - definir os tipos de valores para os *slot*s.

3. TRABALHOS RELACIONADOS

Este capítulo apresenta trabalhos que possuem abordagens semelhantes ou que em algum aspecto trazem o conhecimento para que seja utilizado no trabalho desenvolvido.

3.1. Hidrologia [HIDR 02]

Hidrologia é um projeto realizado por um grupo de estudantes do Programa de Engenharia de Sistemas e Computação da Universidade de Federal do Rio de Janeiro (COPPE - Sistemas/UFRJ). O trabalho desenvolvido por eles tem o objetivo de obter gestão de conhecimento aplicando as técnicas de modelagem de domínio e ontologias. A motivação foi realizar a comparação dos modelos para organização e armazenamentos disponíveis nas técnicas já citadas.

Os estudantes realizaram análises envolvendo as duas técnicas, e avaliaram as abstrações oferecidas para a modelagem por cada abordagem. Estas buscam identificar fatores comuns que permitem propor componentes gerais de uma representação do conhecimento de domínio adequada para a manipulação através de programadores de computador [HIDR 02].

A perspectiva de comparação escolhida é a prática de modelagem através do uso de uma ferramenta adequada para cada abordagem. O estudo de caso apresentado, parte do princípio que a aquisição já foi realizada e que o indivíduo que realizará a modelagem já possui algum domínio sobre o conhecimento a ser modelado [HIDR 02]. A comparação dos estudantes se concentrou em determinar as abstrações disponíveis em cada abordagem destacando abstrações para representação de: Entidades de domínio, Atributos e Relacionamentos.

O trabalho concentrou-se na utilização dos dados do estudo de caso de um sistema já existente denominado *HidroWeb,* que fornece uma interface com consulta de dados com o propósito de apresentar esses dados de uma forma que seja reutilizado por outras entidades. A semântica do *HidroWeb* é definida por um glossário de termos com base na Classificação Decimal Universal (*Universal Decimal Classification – UDC*) que é um esquema hierárquico através de facetas.

O objetivo dos autores do trabalho era fazer com que existisse uma base de conhecimento único contento todas as relações de hidrologia do sistema *HidroWeb* que

utilizaram como pesquisa de dados. A Figura 18 apresenta a imagem da ontologia inicial do projeto Hidrologia que mostra as classes e atributos e seus devidos relacionamentos.

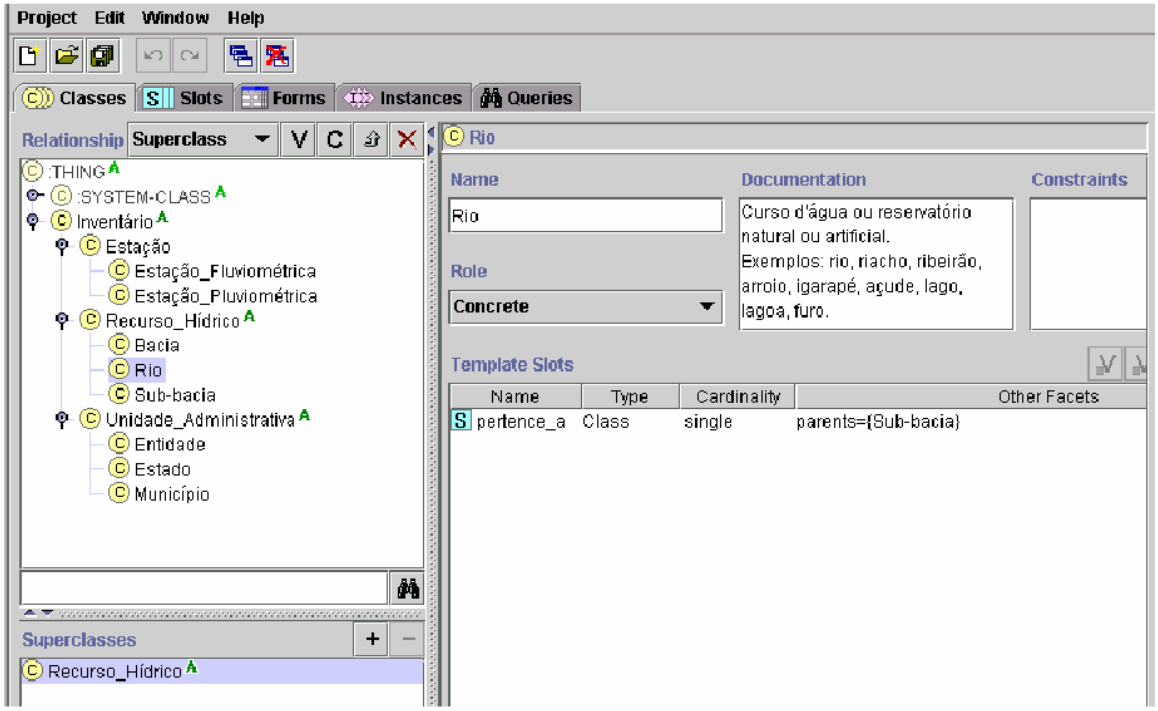

Figura 18: Ontologia do projeto Hidrologia.

O trabalho também apresenta um estudo voltado à modelagem de análise de domínio através dos diagramas de *features*, que são criados baseando-se em casos de uso do domínio. Os autores propõem-se modelar o conhecimento do domínio através de diagramas estendidos da UML, como diagrama de classes, casos de uso e seqüência, como pode ser visto na Figura 19.

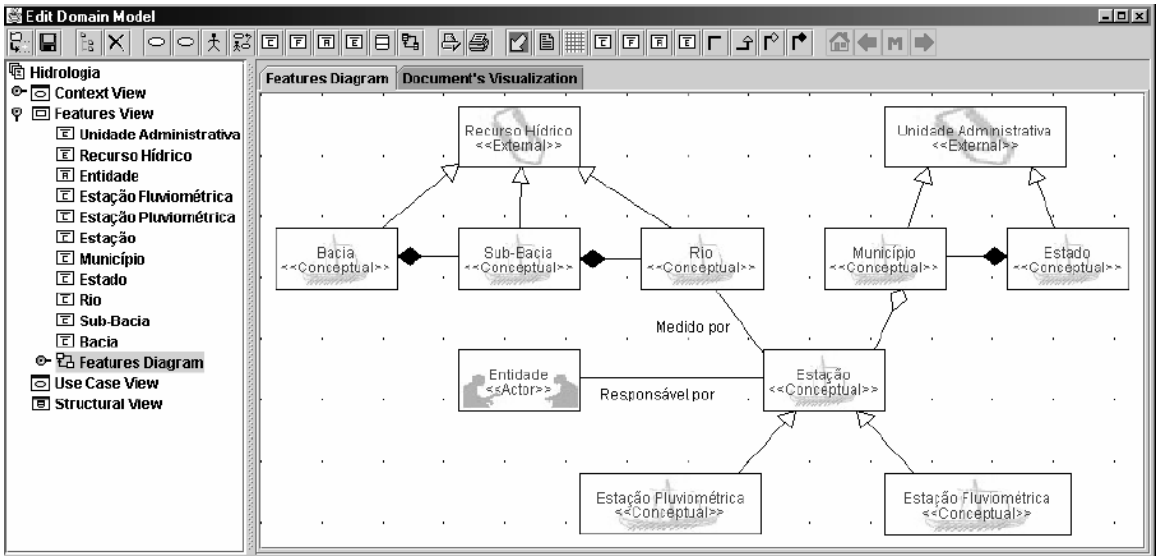

Figura 19: Modelo de *features* do domínio Hidrologia.

Após a finalização da etapa de modelagem de domínio os autores chegaram à conclusão que ambas as técnicas possuem muita semelhança para a criação de uma base de conhecimento de análise de domínio. Para [HIDR 02], "*a comparação das ferramentas evidencia necessidades práticas dos especialistas. A visualização do modelo sendo construído é a principal delas. A comunicação visual é favorecida nos modelos de domínio, principalmente a percepção dos relacionamentos. Por sua vez, a ontologia oferece facilidades para encontrar classes e slots através de consultas*".

3.2. IIMPAR [ANDR 03]

O projeto IIMPAR é uma dissertação de mestrado em Ciência da Computação da Universidade Federal do Ceará que tem como objetivo principal 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, foram utilizados modelos de padrões armazenados no formato XMI (*XML Metadata Interchange*). A Figura 20 ilustra a proposta para a reutilização de modelos de padrões de software.

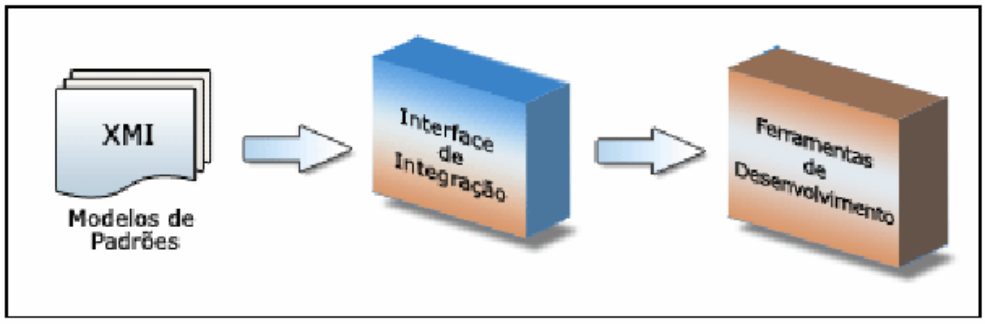

Figura 20: Proposta da reutilização dos modelos de padrões.

Segundo [ANDR 03], 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* [RATI 05].

A proposta abordou a criação de um repositório de padrões de *software* para suprir a carência de mecanismos que auxiliem no armazenamento e busca dos diferentes tipos de padrões de software existentes na literatura. O repositório criado pelos autores é capaz de armazenar os mais diversos tipos de padrões encontrados na literatura. Para a criação e armazenamento dos mesmos é necessário aplicar *templates* específicos para a criação dos padrões.

A Figura 21 apresenta 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. Além disso, 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 *Rational Rose*. Entretanto, a interface de integração é capaz de receber modelos XMI de padrões de qualquer origem.

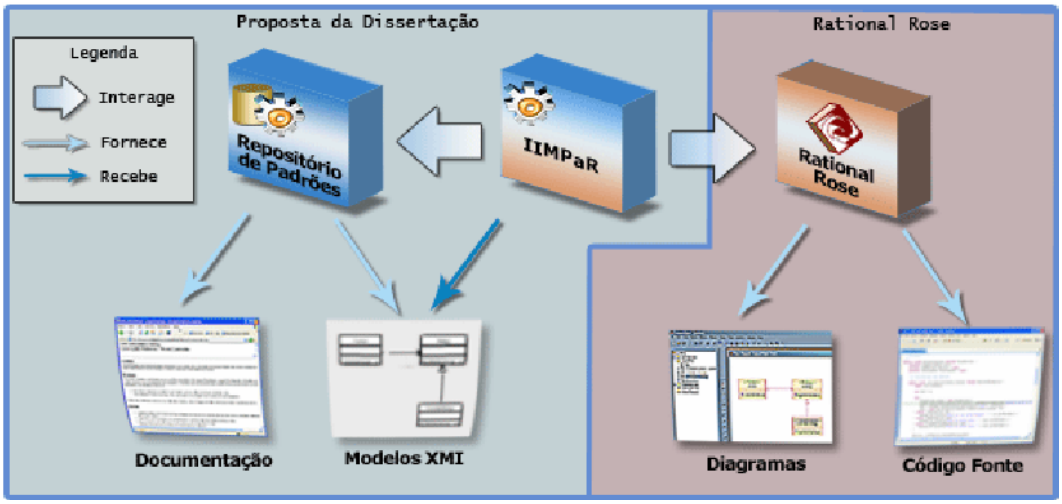

Figura 21: Proposta de Integração do Repositório de Padrões.

O repositório de padrões de software possui os seguintes mecanismos: criação dos padrões de software, criação de projetos e busca simples e avançada dos padrões de software. A ferramenta proposta possui a abordagem de manutenção dos padrões e geração dos padrões em formato XMI, e então aplicá-los em uma ferramenta de modelagem, como por exemplo, *Adapter, Observer e Composity*. A Figura 22 apresenta a tela de busca avançada do repositório com o resultado de padrões a ser apresentado.

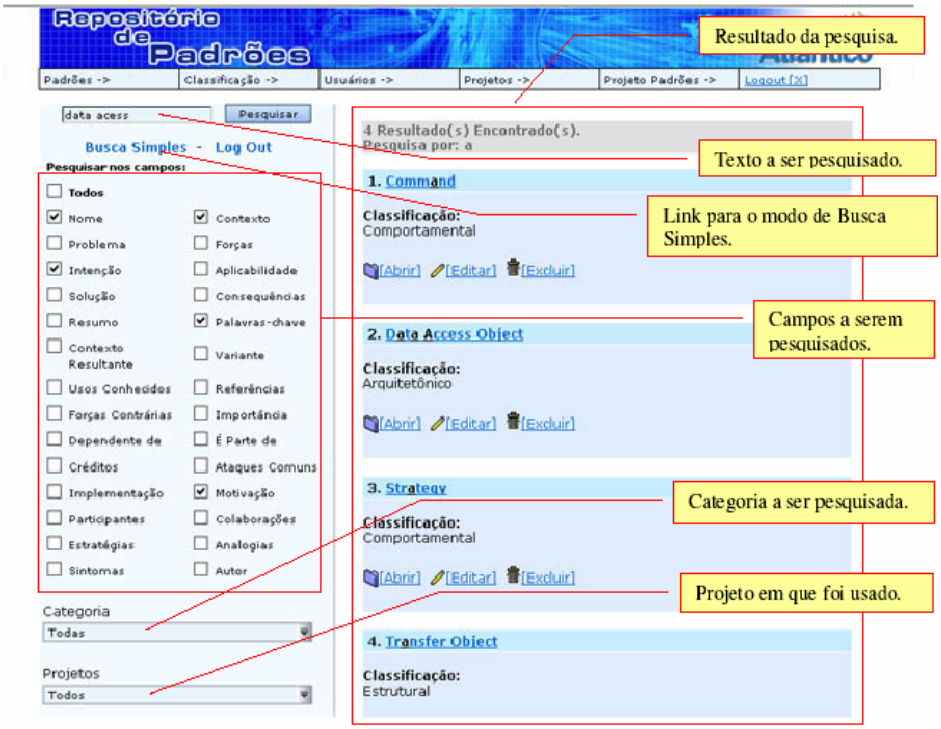

Figura 22: Ferramenta de busca avançada do Repositório de Padrões.

3.2.1. **Arquitetura do Repositório**

O repositório de padrões de software apresentado no trabalho relacionado fornece um requisito funcional denominado “visualizar dados”, 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 apresentados para o repositório.

As características citadas foram importantes na escolha da plataforma *J2EE* para o desenvolvimento do repositório. As tecnologias do J2EE (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, utilizam como padrão para a arquitetura do sistema o MVC (*Model-View-Controller*) que ajuda a estruturar os componentes em camadas. O padrão *Web Interceptor*, é um padrão usado como componente centralizador de requisições web e é usado como *Controller* da arquitetura MVC. O padrão *Web Interceptor* funciona como uma adaptação do padrão *Facade* para sistemas *Web*. Outros padrões de software são usados na implementação do repositório como o *Data Access Object* (DAO), o *Value Object* (VO) e o *Abstract Factory* [ANDR 03].

A Figura 23 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.

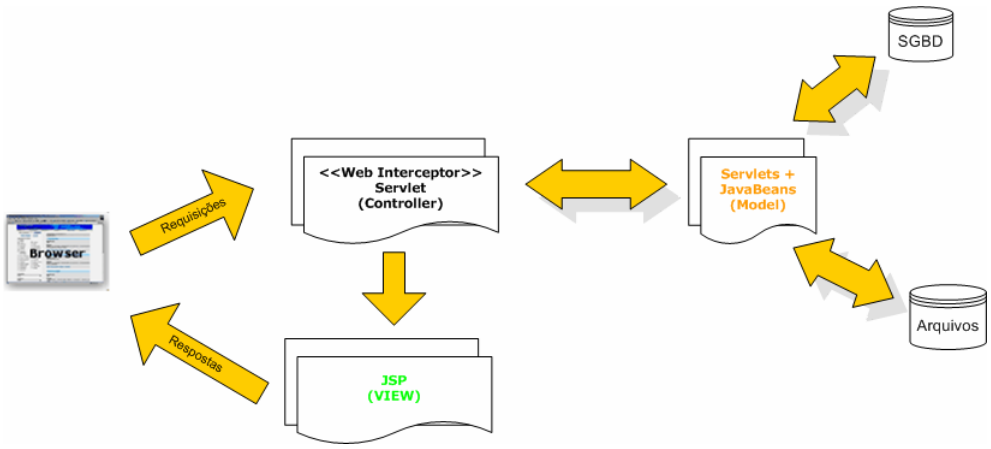

Figura 23: Arquitetura do Repositório de Padrões.

3.2.2. **Processo de Integração**

O processo de integração proposto por [ANDR 03], 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 24.

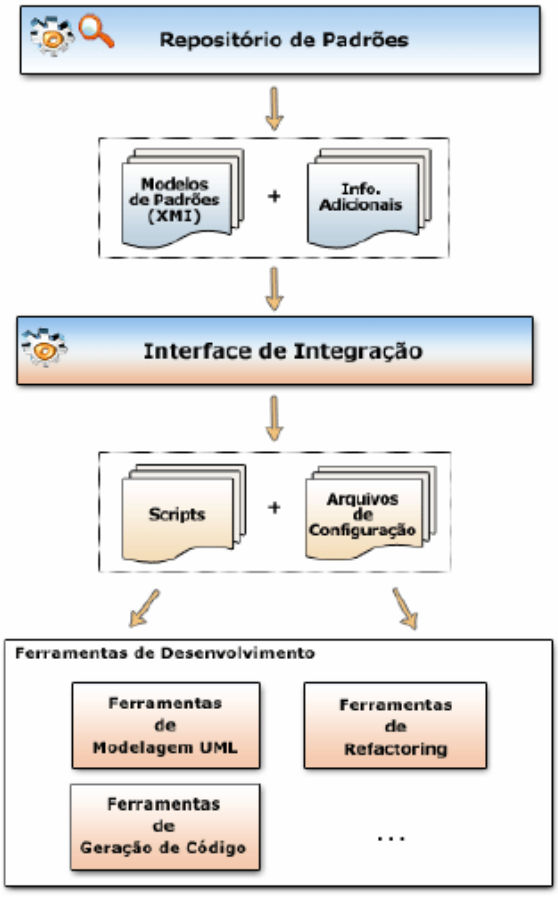

Figura 24: Cenário de Aplicação da Interface de Integração.

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 [ANDR 03]. 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.

Em conjunto com o repositório de padrões, 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 inserí-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.

3.3. Improving Analysis Pattern Reuse [HAMZ 02]

Este trabalho refere-se a um estudo de construção do conceito de padrões de análise em um formato ontológico. O autor baseia-se na identificação de padrões de análise através de quatro abordagens: *Direct Aprroach, Specialization Approach, Analogy Approach*, e *Stability Approach*. A Figura 25 apresenta os quatros desenvolvimentos das abordagens e dá exemplo de cada uma delas. Em *Direct Approach*, padrões de análise são identificados, mas estes padrões de análise não são generalizados ou abstratos. De acordo com esta abordagem não existe garantia de que os padrões abstratos serão aplicados com sucesso em outros contextos. Os exemplos do padrão de análise para esta categoria são os padrões de análise dados por Martin Fowler. Em *Specialization Approach,* os padrões de análise identificados são abstraídos de modo que possam ser utilizados em projetos similares e relacionados com projetos de mesmo conceito de especialização.

Em *Analogy Approach*, os padrões de análise são abstraídos para construir *templates* e então criar uma analogia entre a aplicação original e a aplicação nova, assim garantindo a ordem de adaptação dos *templates* para os demais padrões encontrados. Esta abordagem é amplamente utilizada para padrões de análise. Em *Stability Approach* são criados os padrões estáveis, cujo objetivo é desenvolver os modelos que capturem o conhecimento do núcleo do problema. Esta abordagem é responsável pela criação de camadas de desenvolvimento dos sistemas. As classes do sistema são divididas em três camadas: *Enduring Business Themes (EBTs), Business Objects (BOs)* e *Industrial Objects (IOs)*. EBTs são as classes do

conhecimento do negócio, os BOs traçam as classes EBTs para objetos mais concretos, e IOS mapeiam os BOs do sistema dentro de objetos físicos.

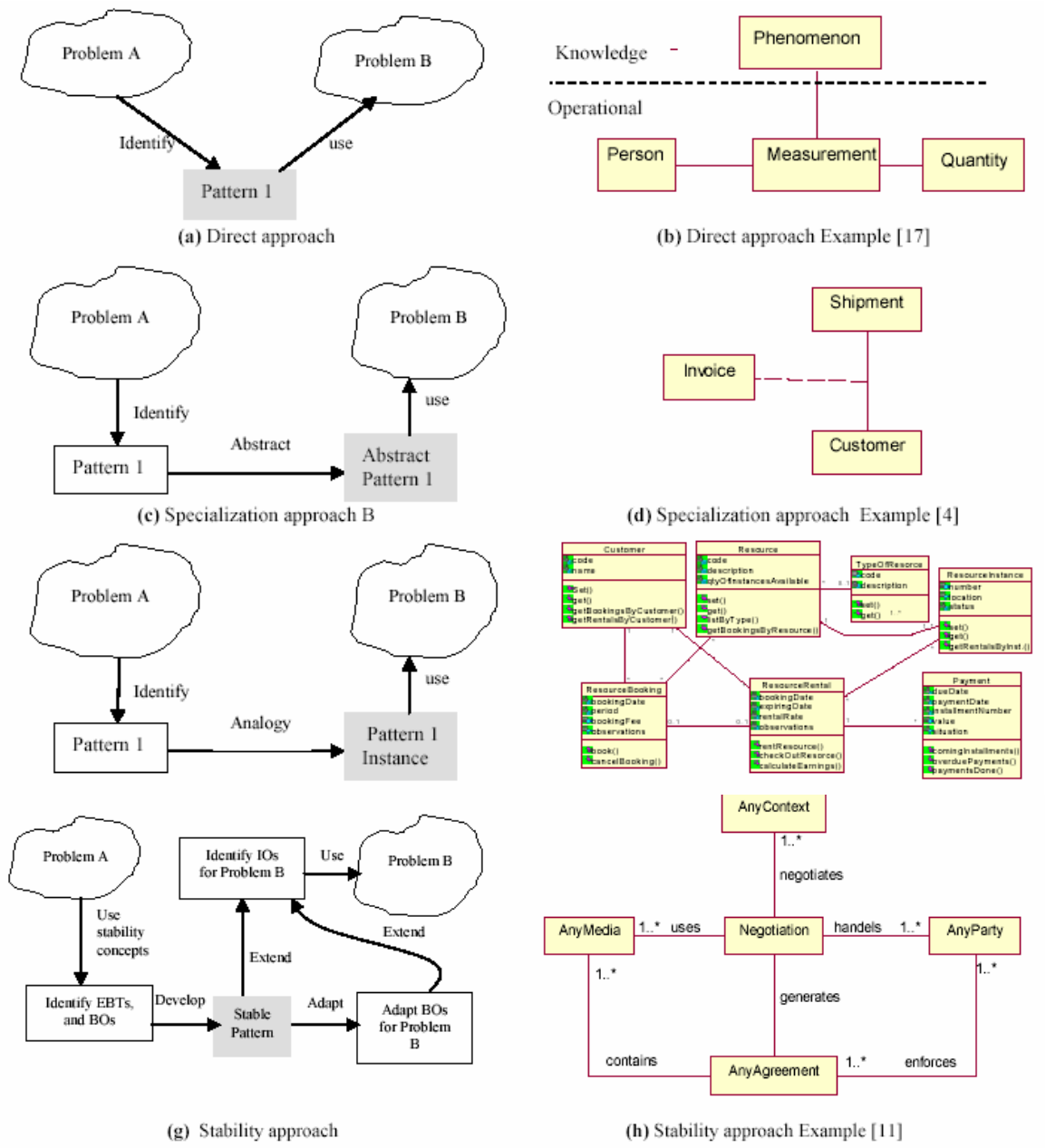

Figura 25: Abordagens de Padrões de Análise.

O trabalho apresenta alguns desafios que podem ser enfrentados para a descoberta de padrões de análise, bem como a pouca popularidade existente na continuação de trabalhos relacionados à descoberta de padrões de análise. Uma razão pela qual o autor justifica isso é a exigência quanto à qualidade de levantamento de requisitos dos projetos. Segundo [HAMZ 02], os padrões de análise necessitam ser gerais o bastante para que possam ser

utilizados por outros projetos, assim sendo eles reutilizáveis. Algumas outras fortes razões são apresentadas pelo trabalho e mostradas abaixo:

3.3.1. **Estrutura Inconsistente**

Os padrões de análise podem ter estruturas diferentes. Por exemplo, a Figura 26 (b) apresenta um padrão da análise com duas camadas: a camada do conhecimento e a operacional. Já a Figura 26 (d) e (e) mostra o padrão com uma única camada, e a Figura 26 (b) com duas camadas, citadas como a camada de EBT e a camada de BO. Esta inconsistência da estrutura pode limitar o reuso dos padrões, porque a reusabilidade da estrutura específica do padrão é limitada para aqueles padrões que seguem a mesma estrutura.

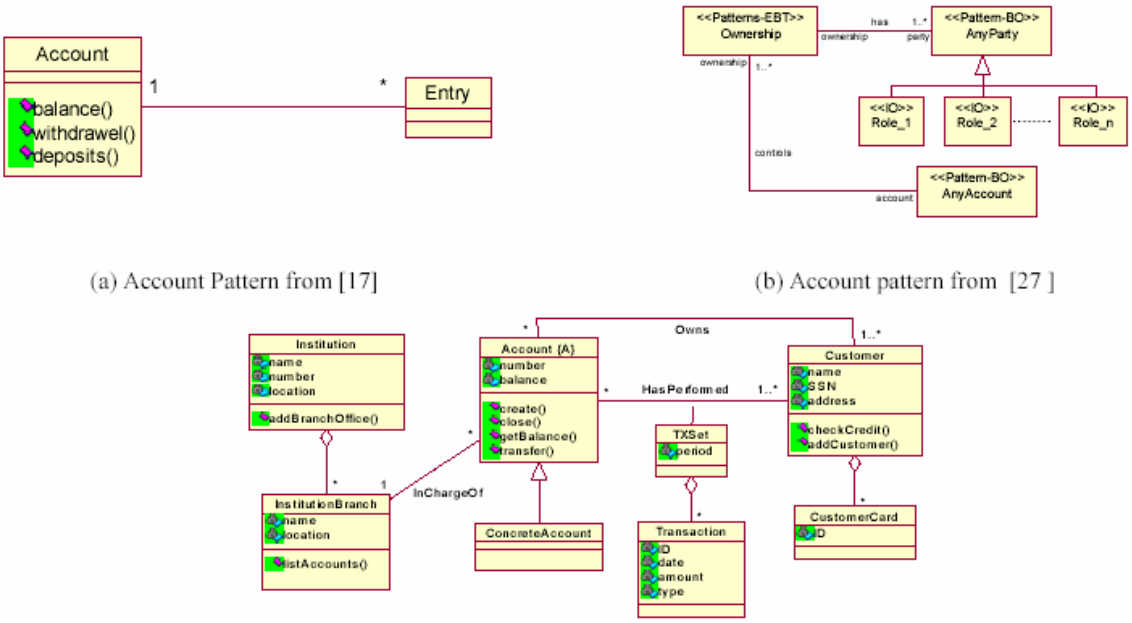

Figura 26: Diferente estrutura do padrão Conta.

3.3.2. **Padrão Redundante**

Este problema pode ser visto como uma das conseqüências já citada no item anterior. A inconsistência da estrutura complica o uso dos padrões de análise, podendo não beneficiar a utilização do reuso dos padrões de análise em estruturas diferentes. A figura 26 apresenta três padrões diferentes que foram propostos para modelar uma conta.

A Figura 27 mostra uma representação de alto nível de uma maneira de aplicar ontologias aos padrões de análise. A abordagem consiste em quatro fases principais: extração do conhecimento, desenvolvimento da ontologia, o reuso do conhecimento e aumento do

conhecimento. Na fase da extração do conhecimento uma coleção de padrões existentes e outras fontes do conhecimento, tais como modelos do domínio e modelos relevantes de outros projetos, são avaliados com cuidado. A extração do conhecimento pode fazer exame de diversos formulários baseados no domínio e no conhecimento disponível. O conhecimento extraído é alimentado então na fase do desenvolvimento da ontologia.

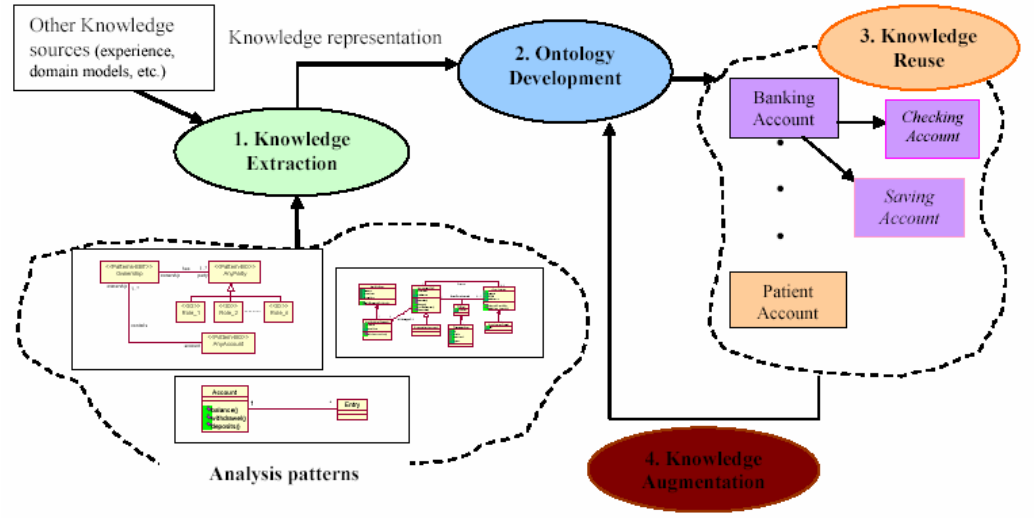

Figura 27: Uma abordagem alto nível de ontologias para padrões de análise.

3.3.3. **Considerações sobre os trabalhos relacionados**

A ferramenta proposta para este trabalho utiliza a junção das duas abordagens dos trabalhos relacionados. Um comparativo entre os três projetos pode ser conferido na Tabela 2 e posteriormente discutido a fim de justificar o desenvolvimento desta proposta.

Tabela 2: Quadro comparativo com trabalhos relacionados.

	HIDRO	IIMPAR	APPROACH	RPAO
Catálogo de padrões	X	X		X
Utilização de Ontologia como Base de Conhecimento	X		X	X
Apresentação dos dados através de uma aplicação específica de apresentação		X	X	X
Modelagem de domínio em Ontologia	X		X	X
Interoperabilidade entre agentes e base de conhecimento				X

A proposta atual RPAO procura englobar todas as características que as outras ferramentas utilizam, sendo assim deseja-se manter todo o conteúdo de modelos de domínio focado em uma base de conhecimento e que possa ser integrado através de uma ferramenta. A solução proposta para o trabalho será apresentada no capítulo seguinte em detalhes.

4. RPAO – Repositório de padrões de análise mapeados em Ontologias

Nos capítulos anteriores foram apresentados aspectos relacionados aos padrões de análise, padrões de projeto e ontologias. Este capítulo apresenta uma proposta para armazenar e pesquisar padrões de análise e mapeá-los em ontologias. Para que através disso seja possível apresentar padrões de análise que serão descobertos e, que poderão ser catalogados para que outros engenheiros de softwares possam identificá-los e aplicá-los de forma a reusar o modelo de um domínio.

Segundo Faria, "*duas abordagens complementares caracterizam o reuso de software: A engenharia de domínio ou desenvolvimento PARA o reuso e a Engenharia de aplicações ou desenvolvimento COM o reuso*". A engenharia de domínio busca a construção de abstrações de software reutilizáveis que serão usadas na criação de aplicações específicas na engenharia de aplicações [FARI 03].

Em padrões de análise, é essencial a engenharia de domínio para obter o reuso e segundo [FOWL 97], "*padrões não são usados para normalizar o sistema de notação, mas sim a forma de pensar sobre o negócio"*. Fowler também ressalta que o reuso pode ser eficiente com a identificação e documentação dos padrões de análise. O entendimento destas similaridades permite ao analista construir um modelo base, o qual poderá ser ajustado às necessidades específicas.

4.1. Tecnologias Utilizadas

O repositório de padrões de análise apresentado nesta dissertação é baseado em um ambiente *web*, em virtude do intuito de aumentar a sua disponibilidade para todos os tipos de usuários. Outras características como manutenibilidade, escalabilidade e suporte a concorrência são requisitos não-funcionais desejados para o repositório.

Esses requisitos foram importantes na escolha da tecnologia Microsoft .NET (leia-se dot net) para o desenvolvimento do repositório. Além da plataforma possuir uma característica de agilidade no desenvolvimento. A utilização da ferramenta Microsoft Visual Studio 2005 *Professional* nos fornece uma coleção de linguagens e bibliotecas unificadas ao *framework* da Microsoft bem como o servidor de aplicação IIS (*Internet Information Service*). A opção do desenvolvimento com o *framework* da Microsoft veio ao encontro as soluções

proposta pela empresa que adotaria o sistema como um piloto, conforme é apresentado no capítulo 6.

Para o armazenamento de dados relacionais, optou-se pelo SGBD (Sistema de Gerenciamento de Base de Dados) Microsoft SQL Server 2005 pela facilidade de administração dos dados e, principalmente, pela fácil conectividade entre os objetos de negócio e a camada de persistência do framework ADO.NET.

Para o desenvolvimento da estrutura para as ontologias foi utilizada a ferramenta Protege que fornece um ambiente gráfico e ágil para a construção de classes, subclasses e propriedades.

Os *WebServices* servem de comunicação entre a ferramenta RPAO e os sistemas externos que buscam reusar as ontologias geradas pelo sistema. Na seção seguinte será apresentada a arquitetura proposta para a ferramenta RPAO com base nas tecnologias escolhidas.

4.2. Arquitetura

A arquitetura para a ferramenta RPAO, como já introduzido na seção anterior, é baseado no framework .NET. Além disso, é utilizado como padrão para a arquitetura do sistema o MVC (*Model View Controller*) que ajuda a estruturar os componentes em camadas. O modelo do padrão MVC representa os dados da aplicação e as regras do negócio que governam o acesso e a modificação dos dados. O modelo mantém o estado persistente do negócio e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo. A Figura 28 apresenta o modelo UML da utilização do padrão MVC dentro do contexto da aplicação. A página aspx.net renderiza o conteúdo de uma parte particular do modelo (PadraoAnaliseDAO) e encaminha para o controlador as ações do usuário acessando os dados do modelo via controlador, assim definindo como esses dados serão apresentados. Por sua vez, o controlador define o comportamento da aplicação, é ele que interpreta as ações do usuário e as mapeia para chamadas do modelo. As ações dos usuários interpretadas no ambiente *web* são cliques de botões ou seleções de menus.

As ações realizadas pelo modelo incluem ativar processos de negócio ou alterar o estado do modelo. Com base na ação do usuário e no resultado do processamento do modelo, o controlador seleciona uma visualização a ser exibida como parte da resposta à solicitação do requisitante. Normalmente, há um controlador para cada conjunto de funcionalidades relacionadas.

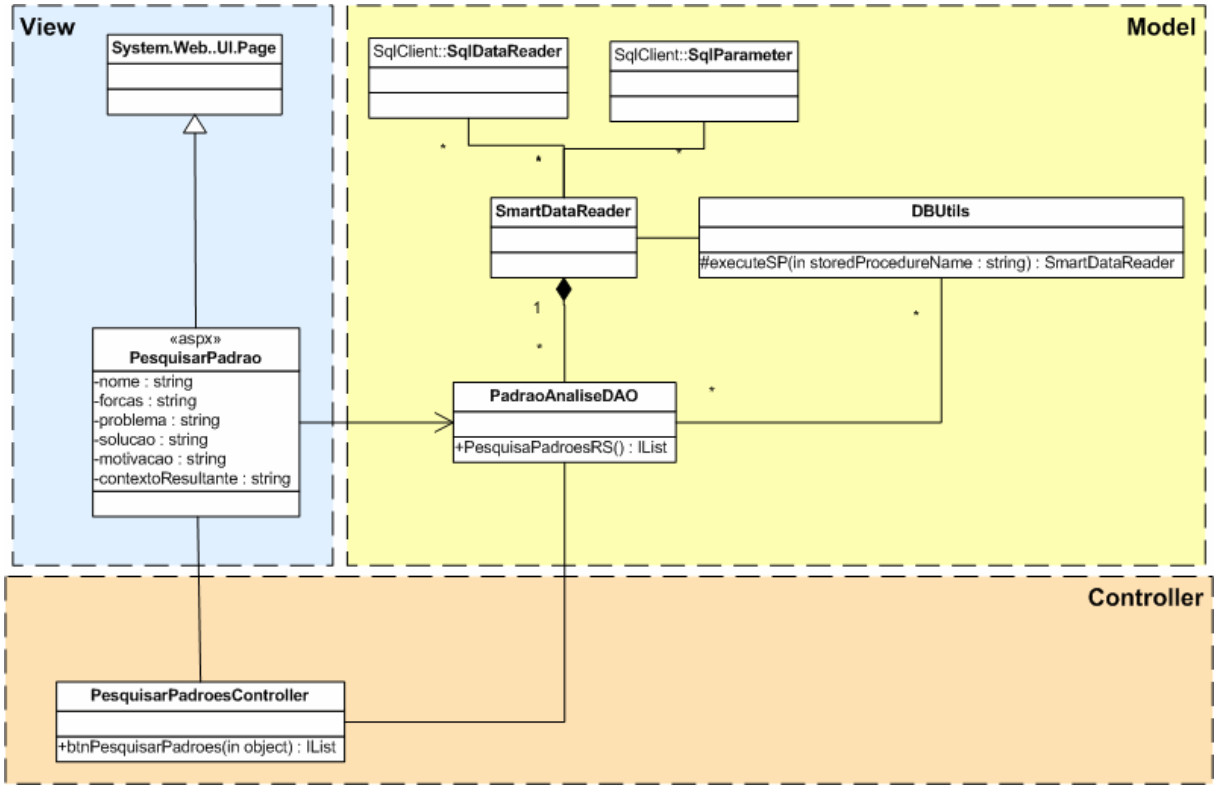

Figura 28: Padrão MVC para o RPAO.

Além do padrão MVC, a arquitetura implementa uma camada de negócio designada BO (*Business Object*) onde são implementadas as regras de negócio da ferramenta, entre elas: as importações de arquivos interoperáveis XMI e a geração de ontologias. Foram utilizados, durante a implementação desta camada da arquitetura, os seguintes padrões de projetos:

- *Façade*: O padrão Facade fornece uma interface unificada para um conjunto de interfaces em um subsistema. O façade foi utilizado para a importação de XMI e também para a geração do gráfico de reuso do sistema.

- *Singleton*: utilizada para assegurar que uma classe possua uma única instância e prover um ponto de acesso global a esta instância. Foi implementado na camada BO para gerenciar o controle de acesso.

A camada de persistência é representada pelas classes nativas da Microsoft ADO.NET. Como já apresentado na seção anterior, a ferramenta utiliza inicialmente conexão com o banco de dados Microsoft SQL Server, porém a arquitetura da ferramenta já provê, através do ADO.NET, conexões com outras bases de dados como Oracle ou DB2. O padrão *Factory* é implementado para esta camada a fim de fornecer conexões com outras bases de dados.

A camada de interoperabilidade, implementada através de *Web Services*, fornece comunicação entre sistemas. A comunicação entre os serviços é padronizada, possibilitando a independência de plataforma e de linguagem de programação. Por exemplo, um sistema desenvolvido em Java e rodando em um servidor Linux pode acessar, com transparência, o serviço feito em .NET da aplicação RPAO que retorna ontologias geradas. A Figura 29 apresenta o modelo conceitual da arquitetura implementada para a ferramenta.

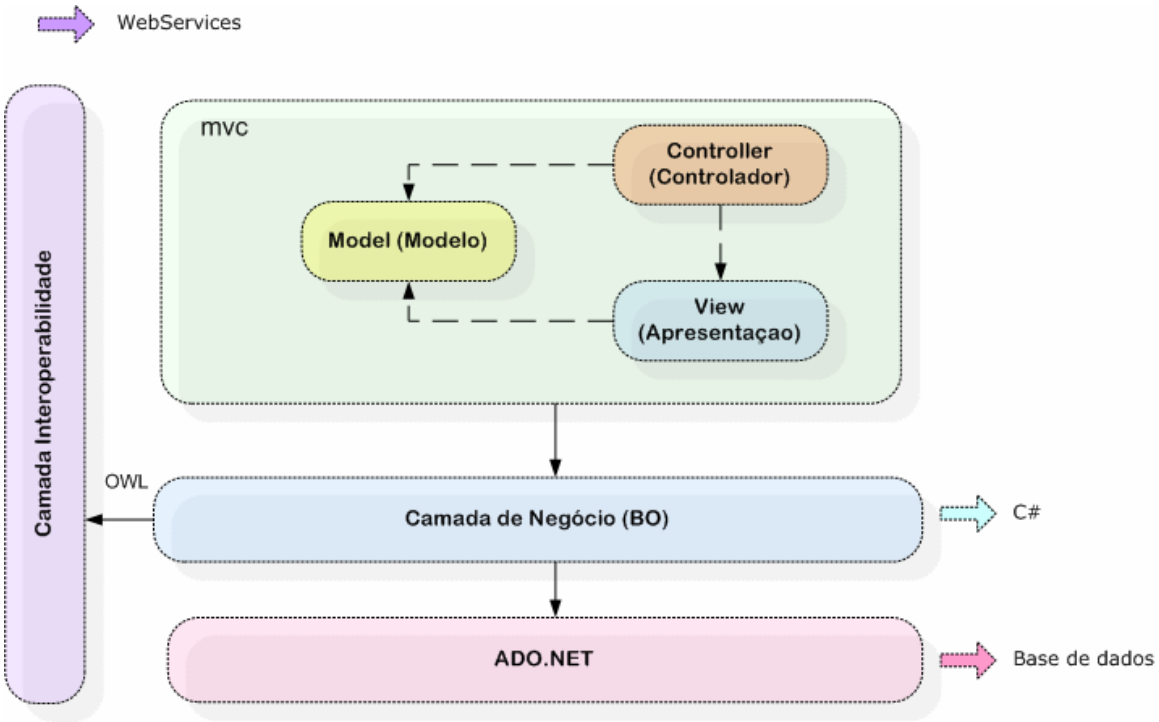

Figura 29: Arquitetura da Ferramenta RPAO

4.3. Especificação do RPAO

Nesta Seção serão apresentadas as principais funcionalidades do repositório de padrões de software desenvolvidos nesta dissertação. A seção está dividida na explicação dos

casos de uso do sistema e após os artefatos (diagrama de classe e seqüência), que apresentam como os casos de uso se transformaram em objetos no sistema. Após são apresentadas seções que explicam o mecanismo de busca e implementação da solução.

Os casos de uso diagnosticados no sistema estão divididos em casos de uso simples e complexos. Os casos de uso simples são destinados aos usuários restritos que apenas entram no sistema, realizam pesquisa de padrões e saem do mesmo. Na Tabela 3 são apresentados os casos de uso simples visíveis através da letra "S" na última coluna. Os casos de uso complexos, representados pela letra "C", são os casos de uso relacionados aos engenheiros de *software* que podem criar projetos e padrões e associá-los a outros padrões.

Tabela 3: Tabela de Casos de uso do Sistema

Use Cases	Descrição	S/C
Pesquisar Padrões de Análise	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 5.3.1 serão apresentados mais detalhes dos mecanismos de busca adotados.	C
Logar no sistema	Destina-se a logar o usuário do sistema dando permissões específicas para ele, como por exemplo, se o usuário é ou não engenheiro de software.	S
Sair do sistema	Destina-se a realizar a saída do usuário do sistema	S
Definir Domínio	Tem como função criar novos domínios de negócio através da aplicação.	C
Manter Usuários	Destina-se a criar o usuário e suas permissões para manipulação da aplicação.	S
Manter padrões de análise	Caso de uso que tem como finalidade a criação de padrões de análise dentro do repositório tendo em vista o *template* e os campos necessários para compor o padrão.	C
Gerir Requisitos	Caso de uso que se destina a relacionar requisitos que se tornaram casos de uso do sistema, e que possuem documentos e artefatos que apresentem histórico do padrão.	C
Gerir Projetos	Destina-se a criar projetos aos quais um padrão de análise estará relacionado.	C
Gerir Artefatos	Obter a manutenção de artefatos armazenados para cada padrão de análise e apresentá-los de forma visível ao engenheiro de software.	C
Gerenciar Ontologias	Para cada padrão diagnosticado é necessário persistir e gerar a ontologia do padrão, à qual este caso de uso se propõe.	C
Importar XMI	Realizar a importação dos casos de uso de um modelo externo para dentro da ferramenta RPAO	C

O principal objetivo do "*Engenheiro de Software"* é manter e aplicar um ou mais padrões no desenvolvimento de um sistema. Neste sentido, a primeira funcionalidade importante para o engenheiro é "*Pesquisar Padrões de Análise"* que realiza a pesquisa e a recuperação dos dados dos padrões armazenados na base de dados e a geração dos mesmos para ontologias através da "*Gerênciar Ontologias"*. Posteriormente, o sistema externo executará os *WebServices* que comunicam-se com o sistema RPAO através do caso de uso "*Exportar Ontologia"*, retornando à ontologia requisitada pelo sistema externo. O caso de uso "*Importar XMI"* refere-se à função de importação de casos de uso para a ferramenta, a descrição deste caso de uso é apresentada na seção 5.3.5.

Os principais atores do sistema, representando os usuários e as entidades externas, são apresentados na Tabela 4 e ilustrados no diagrama de caso de uso da Figura 30.

Tabela 4: Tabela de Usuários do Sistema

Atores	Descrição
Protégé	Ator representado perante uma ferramenta de ontologia.
Engenheiro do *software*	Está envolvido com os casos de usos complexos e com a manipulação do sistema como inclusão de padrões e manipulação de ontologias.
Gerador Ontológico	É representado por um ator, pois será uma ação submetida através de qualquer alteração da base então gerada em uma Ontologia
Sistema Externo	Qualquer sistema desenvolvido que realize acesso aos *web services* da ferramenta RPAO para se comunicar com as ontologias.

O ator E*ngenheiro de Software*, por possuir maiores privilégios no sistema além de suas permissões, herda todas as funcionalidades que o ator *usuário de pesquisa* possui. No diagrama da Figura 31 é ilustrado o envolvimento do *engenheiro de software* com a manutenção do repositório, e os casos de uso que gerem informações relevantes para a base de conhecimento dos padrões de análise.

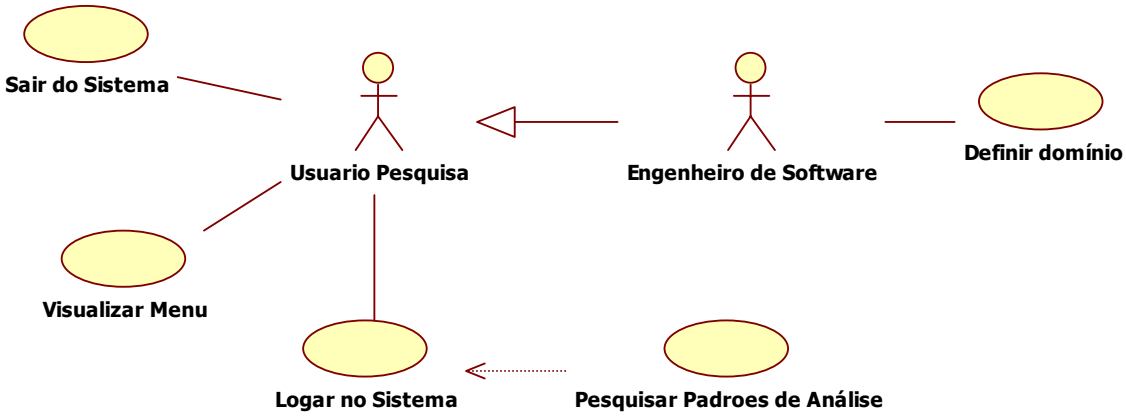

Figura 30: Casos de uso do sistema

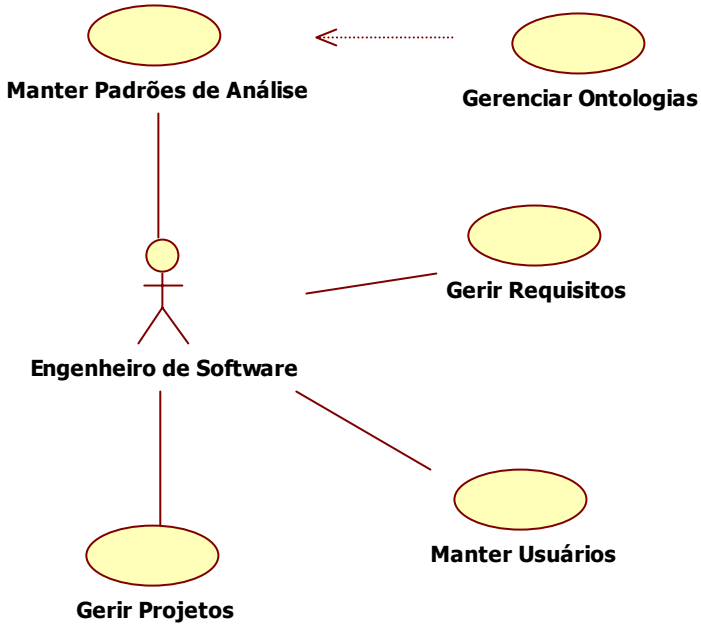

Figura 31: Casos de uso para gerenciamento do sistema

As seções a seguir apresentarão a realização dos casos de uso principais que compõem a arquitetura do repositório e o cenário de aplicação da interface de integração que foram ilustrados nas Figuras 23 e 24.

4.3.1. **Caso de uso Manter Padrões de Análise**

O diagrama de caso de uso da Figura 32 representa o refinamento do caso de uso "*Manter padrões de análise*".

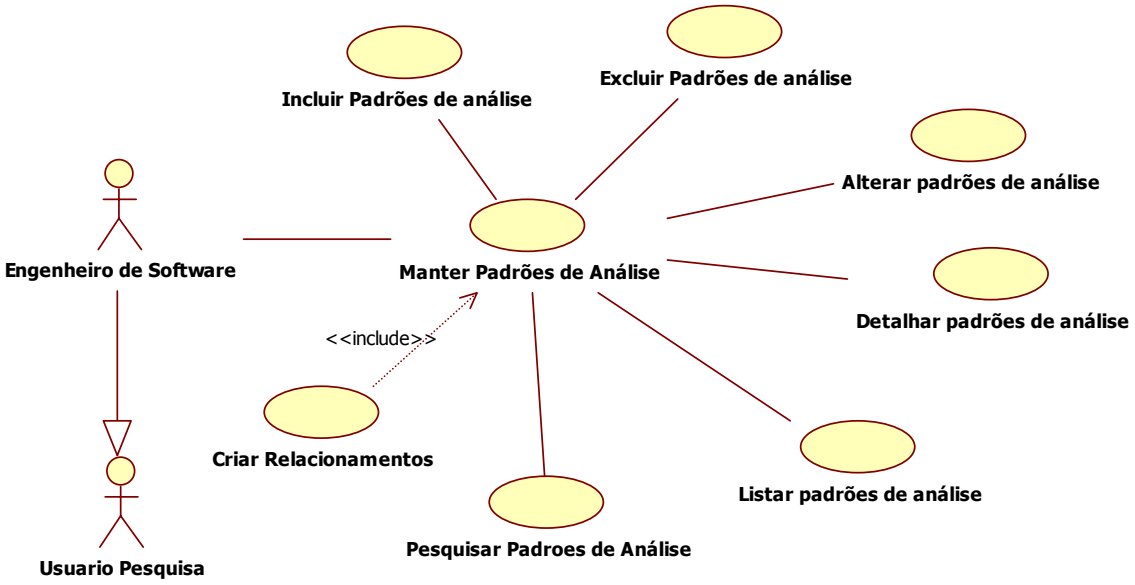

Figura 32: Diagrama de caso de uso manter padrões de análise.

O engenheiro de software possui permissão para total gerenciamento dos padrões de análise identificados que deverão ser apresentados através de um *template* de campos para realização do gerenciamento. O diagrama de caso de uso apresentado na Figura 32 é expandido em outros casos de uso como, "*Incluir Padrões de Análise"* que possui a função de incluir os padrões dentro da ferramenta. A exclusão dos padrões existentes é realizado através do caso de uso "*Excluir Padrões de Análise*" que devem ser apresentados em uma listagem expressados pelo caso de uso "*Listar padrões de Análise"*. A utilização de "*Listar padrões de análise"* também é importante para os casos de uso "*Alterar padrões de análise" e "Detalhar Padrões de análise"*, ambos são executados através da listagem dos padrões.

A Figura 33 apresenta o protótipo de tela da manutenção de padrões de análise e também os padrões relacionados.

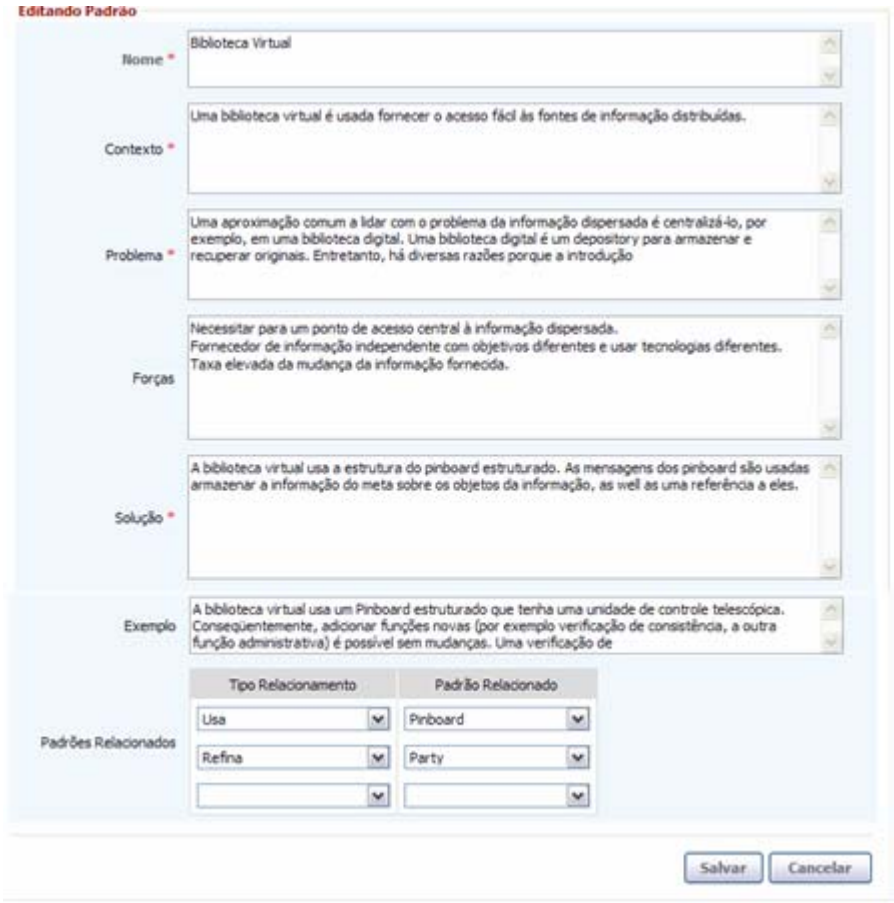

Figura 33: Protótipo de tela manter padrões de análise.

Para descrição do diagrama de casos de uso demonstrado pela Figura 33 optou-se pelo modelo casual, conforme o anexo 1. No anexo é apresentada a descrição de cada um dos casos de uso refinados.

A pesquisa permite que sejam selecionados múltiplos campos do *template* do padrão para a consulta. Pode-se ainda refinar a pesquisa selecionando um dos projetos cadastrados pelo sistema que estejam relacionados ao padrão. A Figura 34 mostra a janela proposta para a realização de busca avançada no repositório de padrões. Na esquerda da janela estão 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 e do botão que aciona a busca e de um link para o modo de busca simples. Os resultados serão listados no lado direito da janela.

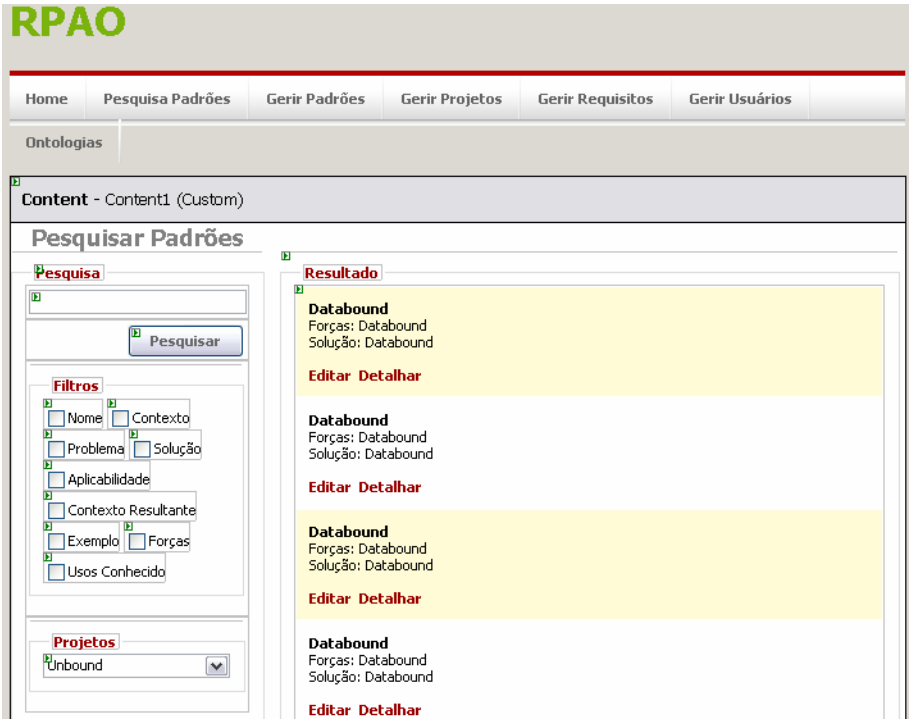

Figura 34: Protótipo de tela Pesquisar Padrões de Análise.

4.3.2. **Caso de uso Gerir Projetos**

O diagrama de caso de uso da Figura 35 representa o refinamento do caso de uso "*Gerir Projetos*". A gestão de projetos tem a finalidade de criar projetos que um padrão de análise possui relacionamento, assim facilitando a observação de reuso.

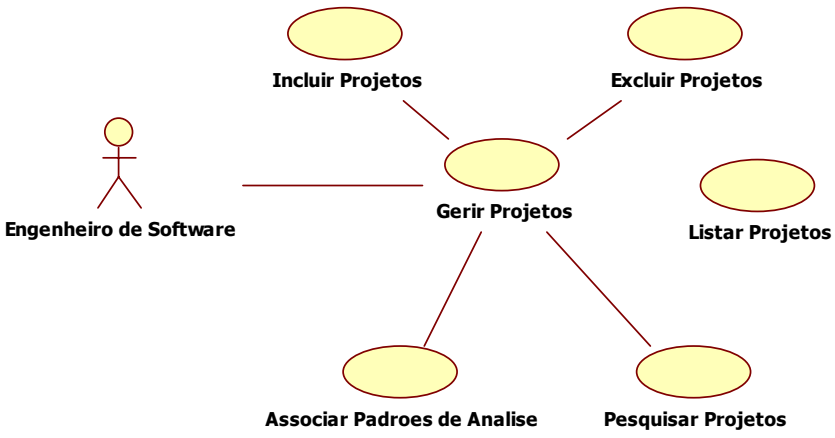

Figura 35: Diagrama de Caso de uso gerir projetos.

Segue abaixo, na Figura 36, o protótipo realizado para gerir projetos e as descrições dos casos de uso referentes aos refinamentos: *Incluir Projetos, Excluir Projetos, Associar Padrões de Análise, Pesquisar Projetos*. No Anexo 2 deste trabalho é apresentada a descrição textual destes casos de uso. No topo da tela de gestão de projetos é apresentada a listagem dos projetos já inseridos, e abaixo da listagem um botão para novos projetos com os campos de inserção de projetos e associação a padrões de análise.

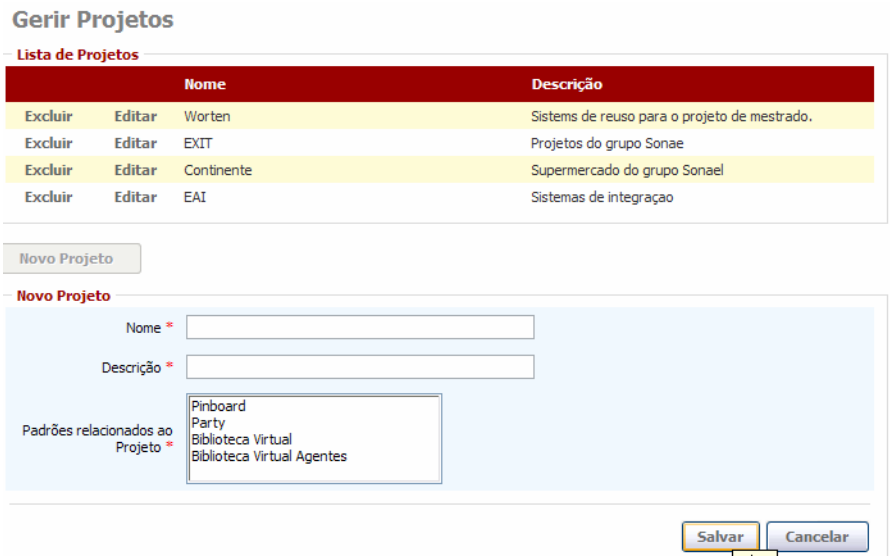

Figura 36: Protótipo gerir projetos.

4.3.3. **Caso de uso Gerir Requisitos**

A ferramenta, além de realizar o armazenamento dos padrões de análise, também se propõe a relacionar estes padrões aos requisitos, pois geralmente um padrão de análise surge de uma necessidade diagnosticada por um requisito que um analista de software encontrou através de um problema junto ao cliente. Foi considerado interessante gerar um *template* que abrangesse requisitos e artefatos encontrados, assim se tornou possível o armazenamento do histórico e, também, dos artefatos gerados para este requisito. O diagrama de caso de uso expresso na Figura 37 e o protótipo na Figura 38 apresentam o gerenciamento de requisitos.

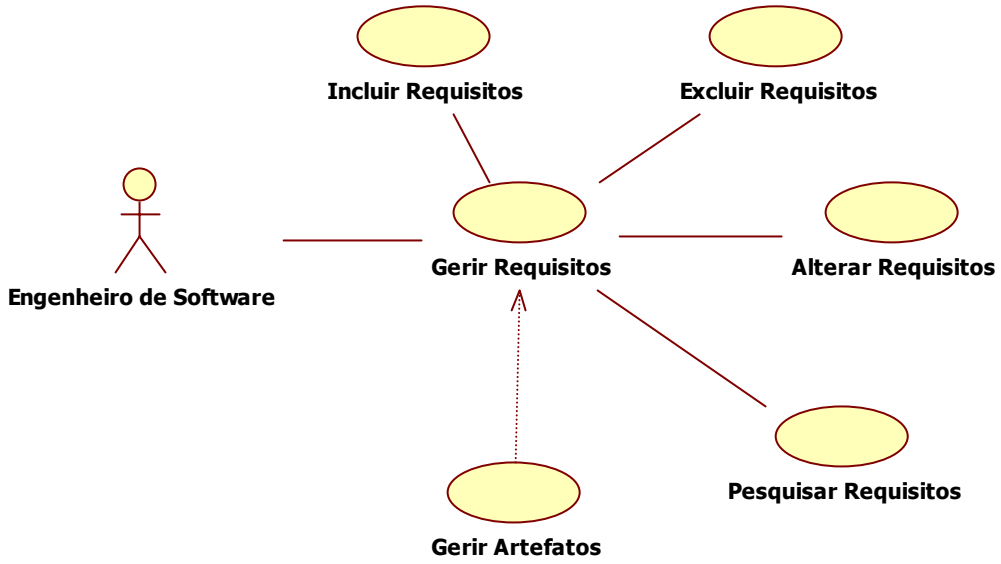

Figura 37: Diagrama de caso de uso gerir requisitos

Ao usar requisitos como ponto de referência do padrão de análise, pode-se gerenciar, dinamicamente, os projetos e identificar os artefatos produzidos para os mesmos. O sistema deve permitir a inclusão de quatro tipos de artefatos (Diagramas: Caso de uso, Atividades, Classes e Seqüência), bem como, o documento de especificação funcional vinculado ao padrão.

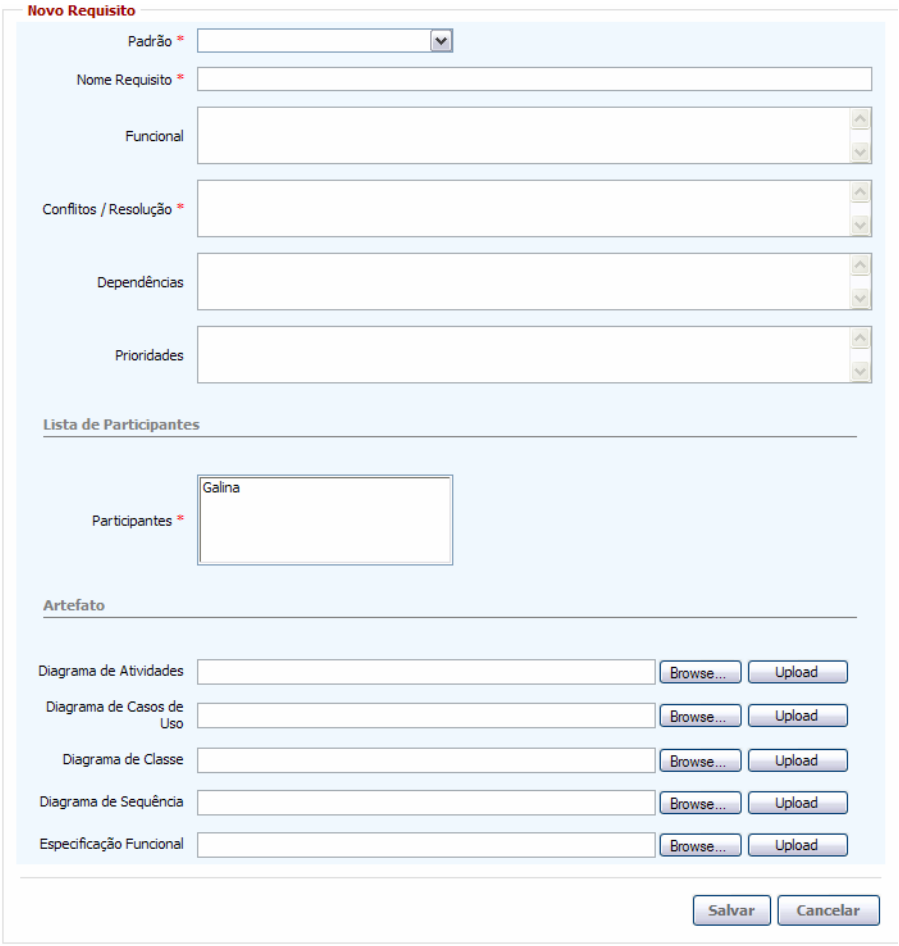

Figura 38: Protótipo de tela Gerir Requisitos

As descrições dos casos de uso refinados acima estão apresentadas no Anexo 3.

4.3.4. **Caso de uso Gestão de Ontologias**

Os casos de uso referentes à estrutura do sistema e à manutenção já foram definidos, restando ainda os casos de uso de ontologias que irão servir para promover a interoperabilidade para outros usuários que não possuírem acesso ao repositório e queiram estar integrados com os padrões catalogados. Estes usuários externos poderão obter as ontologias através do caso de uso "*Exportar Ontologias"* que estará acoplado na camada de interoperabilidade executados por *WebServices* já mencionados no capítulo 5.2 A Figura 39 apresenta os casos de uso de geração de ontologia e a definição da estrutura do domínio e a interação entre o *WebService* e as ontologias geradas.

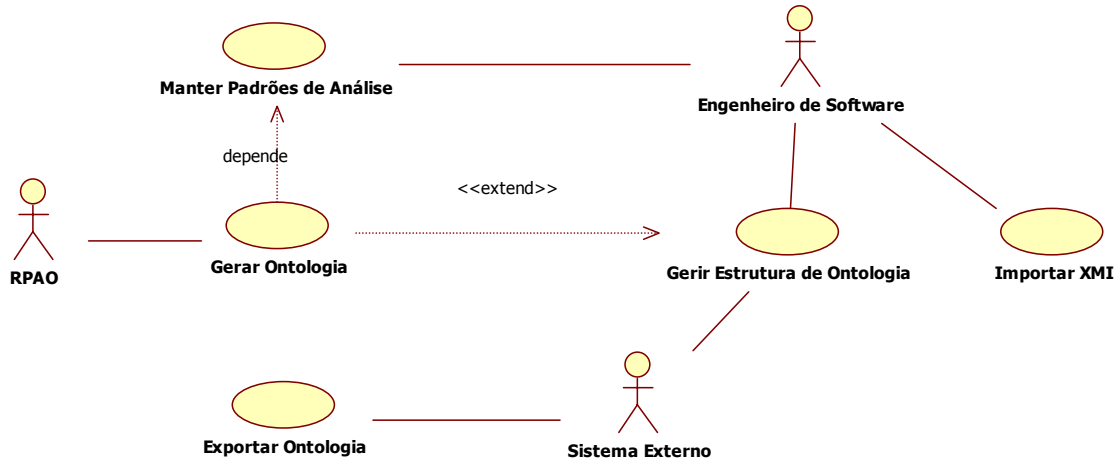

Figura 39: Diagrama de caso de uso Gerar Ontologias

A criação de ontologias estará vinculada a cada ação ocorrida para a entidade "*PadraoAnalise*", ou seja, ao incluir, alterar e excluir um padrão de análise, a operação é refletida automaticamente na manipulação da ontologia. A Figura 40 apresenta o protótipo da listagem de ontologias com um link para visualização do arquivo (owl) em *browser*.

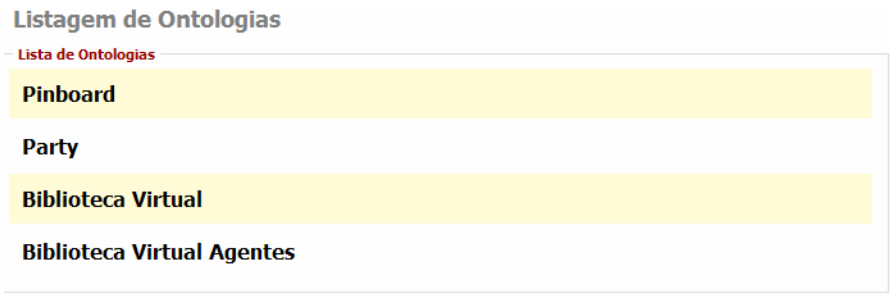

Figura 40: Protótipo de tela listagem de ontologias.

No Anexo 4 é apresentada a descrição dos casos de uso da Figura 40, que detalha o fluxo das atividades entre o caso de uso, sistema e ator.

4.3.5. **Caso de uso Importar XMI**

O sistema comporta também a importação de arquivos XMI que são elementos UML expressados em um arquivo XML. A Figura 41 apresenta o diagrama de caso de uso onde o engenheiro de *software* realiza a importação do arquivo XMI, e após isso, o gerenciamento de requisitos utiliza os dados para a relação entre requisitos e padrões de análise.

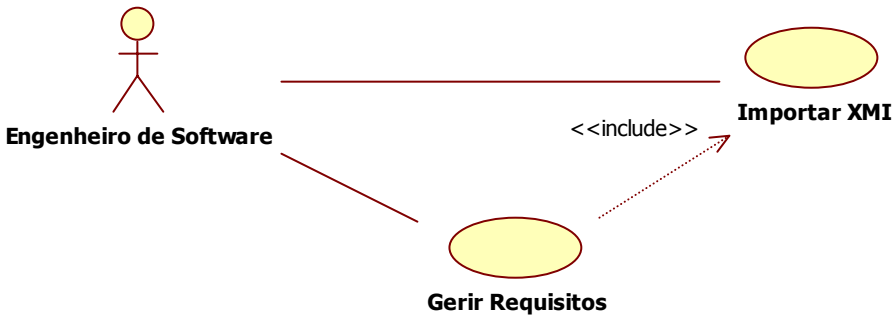

Figura 41: Diagrama de caso de uso importar XMI

A Figura 42 apresenta o protótipo de importação de arquivo XMI. O RPAO deverá processar os arquivos e coletar os casos de uso existentes no XMI exportado, para após apresentar em uma lista todos os elementos encontrados no arquivo. Assim é apresentado na lista o status do requisito importado identificando se já foi relacionado a um requisito ou não.

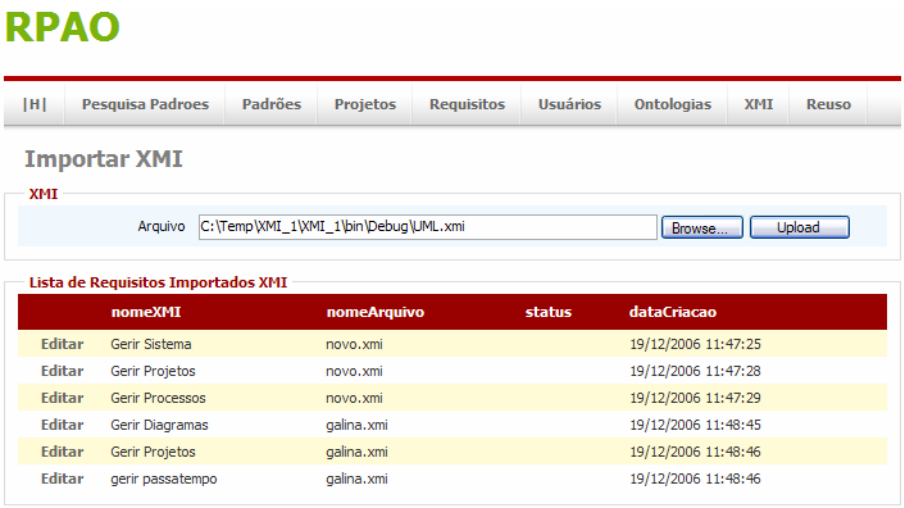

Figura 42: Protótipo de importação de arquivo XMI

4.3.6. **Diagrama de Classe de Análise RPAO**

Os diagramas de caso de uso apresentados na seção anterior transformaram-se em modelos de classe para apresentação do desenvolvimento dos métodos e atributos apresentados na Figura 43.

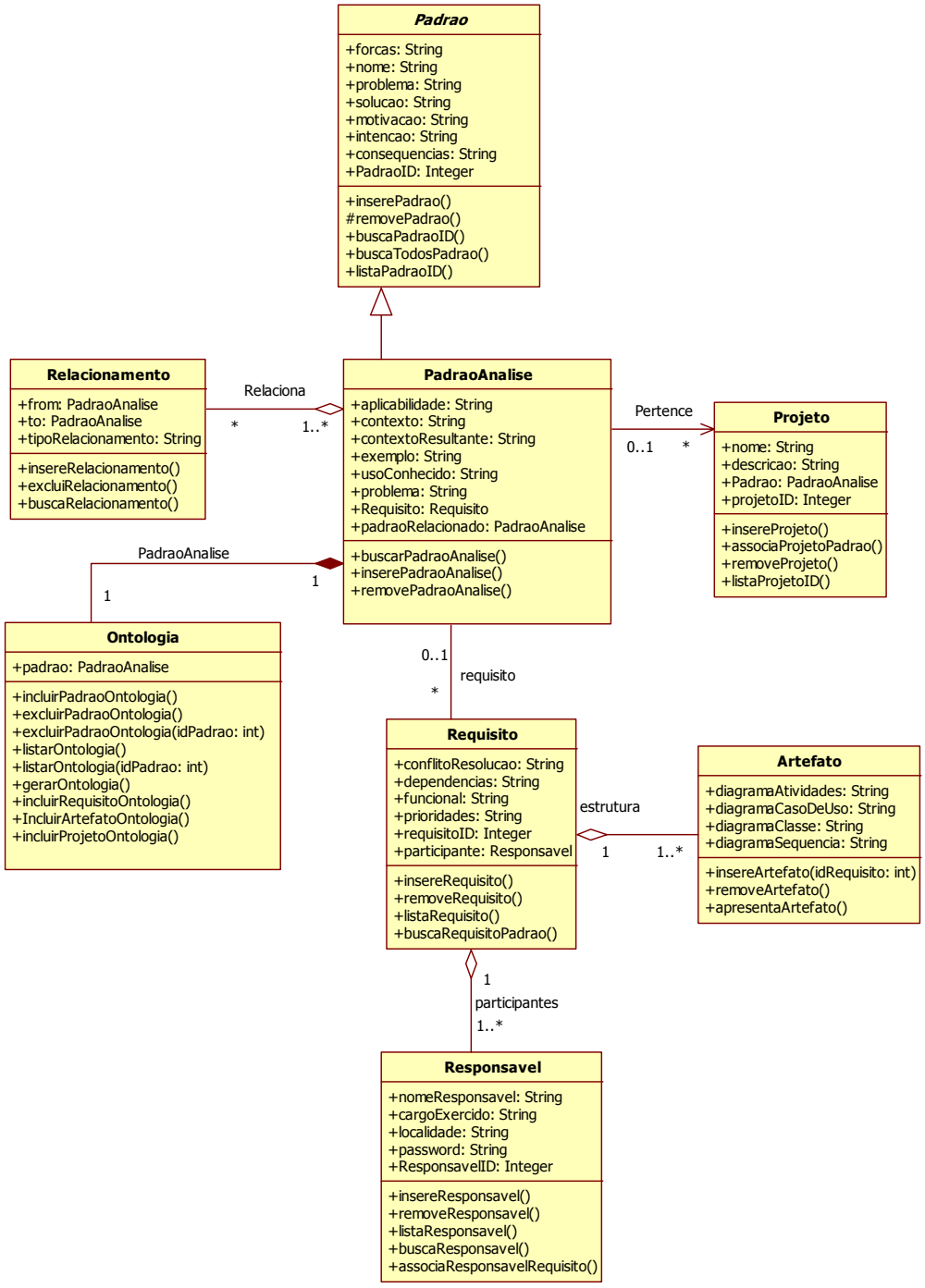

Figura 43: Diagrama de Classes de Análise RPAO.

O diagrama de classe apresentado foi modelado de forma a colocar uma estrutura de padrões genérica, assim para trabalhos futuros ela pode servir de repositório para *design patterns*, pois a classe *"Padrao*" é uma classe abstrata. A classe "*PadraoAnalise*" herda atributos e operações da classe "*Padrao*". Os casos de uso refinados, já descritos em *Manter Padrões de Análise*, se tornaram métodos da classe "*Padrao*" para a realização das operações do sistema.

Entre as classes "*Projeto*", "*PadraoAnalise*" e Requisito, existe uma associação simples, representando que um padrão de análise pertence a zero ou mais projetos, e padrões de análise possuem zero ou mais requisitos relacionados. A classe "*Relacionamento*" destina-se a expor os relacionamentos que a classe "*PadraoAnalise*" cria para relacionar com demais padrões de análise.

A classe "*Ontologia*" serve para tratar toda e qualquer manipulação com ontologias dando destaque para os seguintes métodos:

gerarOntologia() - Para cada novo evento realizado pela classe "*PadraoAnalise*" que persista em base de dados, é invocado este método para que se realize a criação e ou a alteração da ontologia que expressa o padrão de análise.

listarOntologia() - Para toda a requisição que o usuário fizer para procura de um padrão em ontologia é chamado este método.

As demais classes "*Artefatos*" e "*Responsavel*" são classes simples contendo métodos que surgiram da descrição do refinamento de cada caso de uso já apresentado.

4.3.7. **Diagrama de Seqüência RPAO**

Inicialmente, o engenheiro de software incluirá padrões de análise encontrados no sistema. Após o preenchimento dos dados dos padrões de análise, o engenheiro realiza a operação de inclusão e executa o método *inserePadrao* da classe "*PadraoAnalise*". A chamada do método é reconhecida pelo sistema que instancia a classe "*Ontologia*" através do método *incluirPadraoOntologia*, retornando para a classe "*PadraAnalise*" a mensagem de sucesso de inclusão.

O engenheiro recebe o retorno de inclusão do novo padrão de análise, e então através do menu escolhe a opção de “Gerir Projetos” que foi apresentada no protótipo da Figura 34. O engenheiro através da classe “*Projeto*” executa o método “*InsereProjeto*” e, logo após, realiza a associação dos padrões que deseja-se relacionar executando o método *associarPadraoProjeto*. A classe “*Projeto*” instancia o objeto *Ontologia* e executa o método *incluirProjetoOntologia* que incluirá na ontologia já criada o(s) projeto(s) vinculado(s) ao padrão.

Depois de finalizadas todas as inserções, o ator do sistema RPAO, apresentado no diagrama de seqüência de inclusão de dados da figura 44, executará o método de geração de ontologia, o qual publicará a ontologia criada em um diretório virtual. A ontologia estará publicada em um servidor de aplicações podendo ser executada por qualquer ator externo do sistema RPAO, e é gravada uma referência na base de dados do caminho da criação da ontologia.

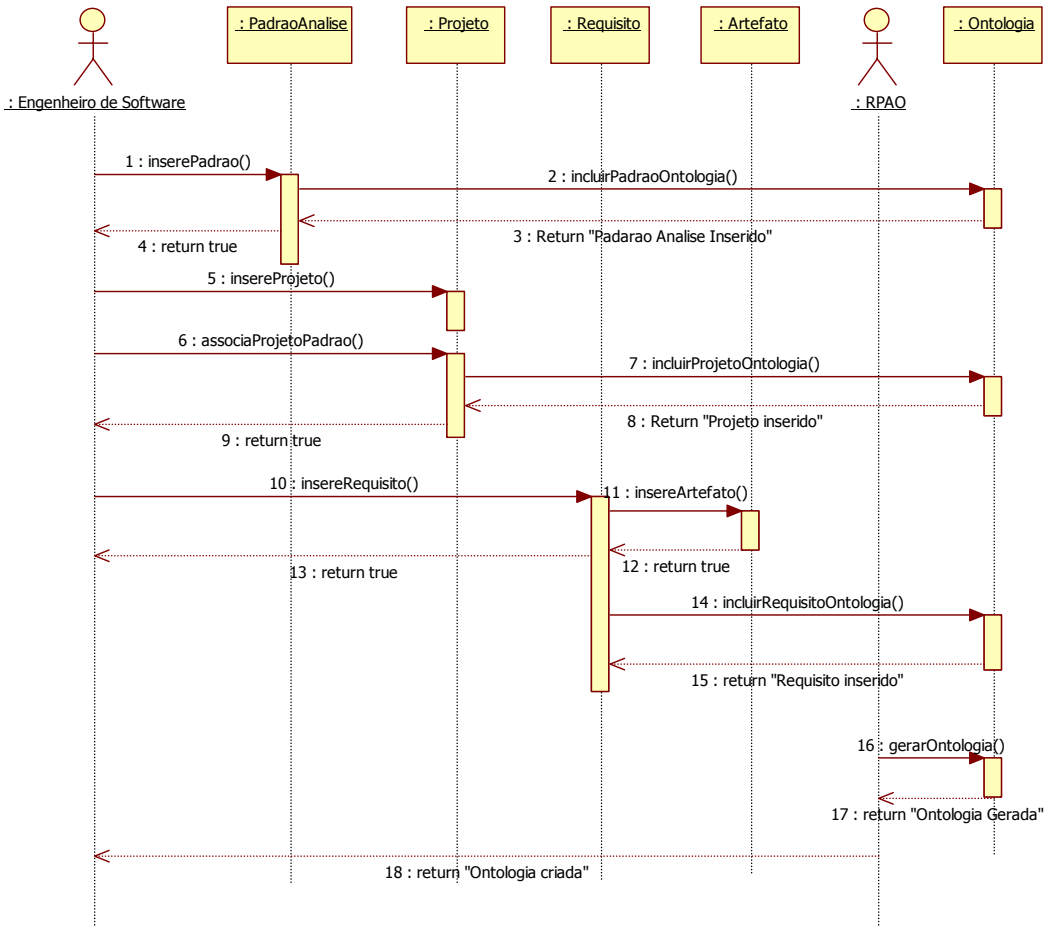

Figura 44: Diagrama de seqüência inclusão de Padrão de Análise.

4.3.8. **Modelo ER (Entidade Relacionamento)**

Com a necessidade de persistência em base de dados, na fase de elaboração modelou-se, juntamente com os demais artefatos, o modelo relacional para a devida construção e geração de scripts na fase de construção. A Figura 45 apresenta o modelo relacional.

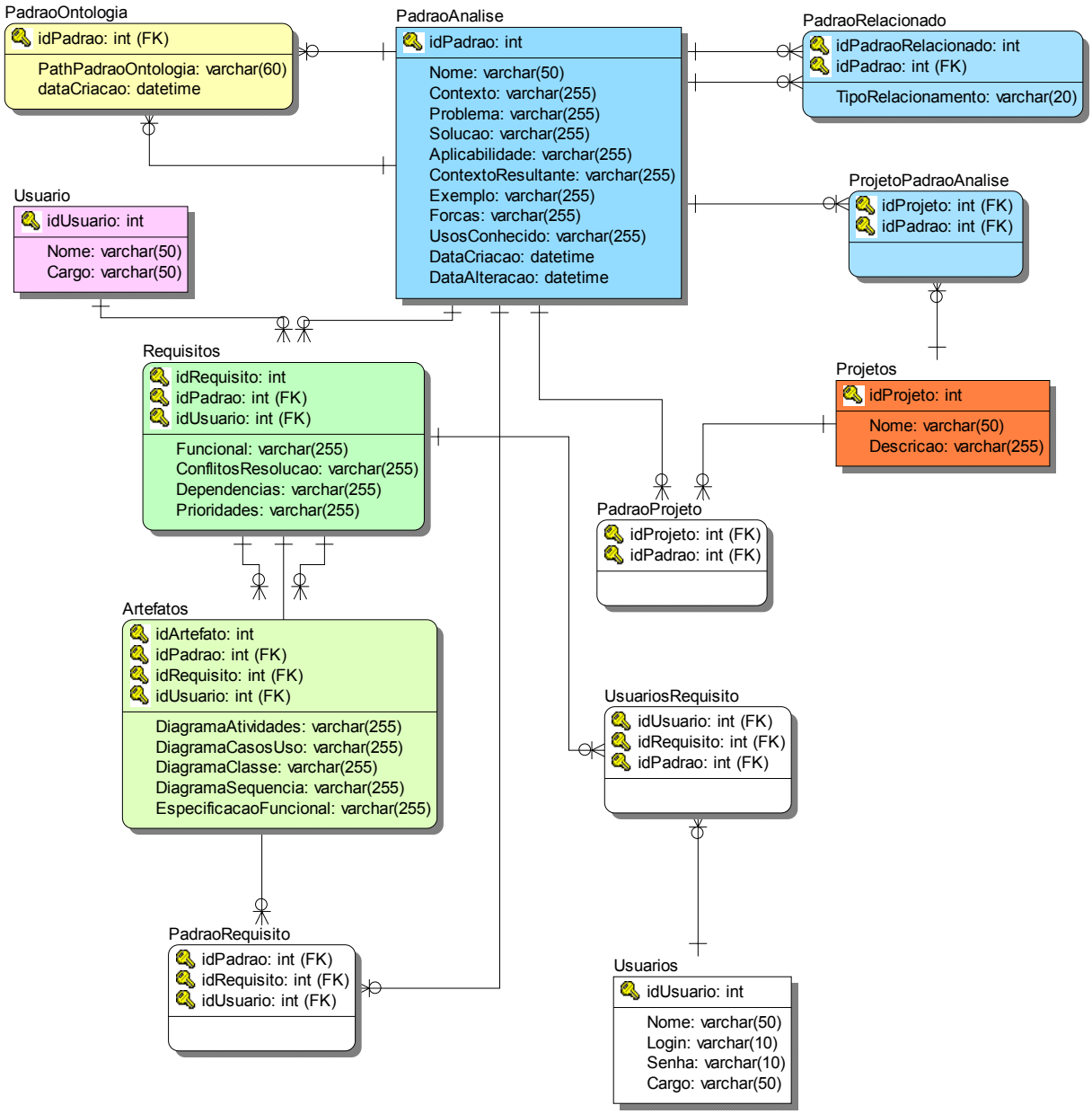

Figura 45: Modelo Relacional (ER).

4.4. Camada de Negócio do RPAO

Os mecanismos de busca do repositório consistem de pesquisas realizadas sobre os campos que compõem os padrões. O formalismo para a entrada da pesquisa é semelhante à maioria dos sistemas de busca da *web*, conhecidos como *search engines* [SECU 07]. Operadores lógicos "e" e "ou" podem ser 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 é gerada uma pilha de operadores e uma pilha de frases (palavras ou frases delimitadas por aspas). O parser é responsável 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 consulta SQL que será usada na consulta ao banco de dados.

A solução descrita acima é realizada através da classe "*PadraoAnalise*" que utiliza um objeto *SmartDataReader*, o qual referencia a base de dados nativa viabilizando o mapeamento para cada campo especificado pelo *Stored Procedure*. A Tabela 5 apresenta a assinatura do método PesquisarPadroes. O método implementado é estático com retorno para uma coleção de dados, representado pela classe de sistema c# <*System.Collections.Generic.List*>, e espera todos os parâmetros da classe para consulta na *Stored Procedure.*

Tabela 5: Assinatura do método PesquisarPadroes.

```
public static List<Padrao> PesquisarPadroes(
int projetoID,
string nome,
string contexto,
string problema,
string solucao,
string aplicabilidade,
string contextoResultante,
string exemplo,
string forcas,
string usosConhecido)
```

Na Tabela 6 é apresentado o trecho do método onde é realizada a criação de um objeto *List* da classe "*Padrao*". Seguem abaixo algumas explicações para o trecho de código:

- ***Reader:*** Objeto que armazena uma coleção de dados retornada de um *stored procedure,* onde são definidos parâmetros que o usuário escolherá para pesquisa.

- ***SqlParameter:*** a execução do método *ExecuteSP* aguarda parâmetros para que o *stored procedure* execute a cláusula de consulta. O comando S*qlParameter* serve para expressar que um novo parâmetro estará sendo criado dentro do método.

Tabela 6: Código Método PesquisarPadroes da classe PadraoAnalise.

```
. . . .

     List<Padrao> padroes = new List<Padrao>();

      SmartDataReader reader =
DBUtils.ExecuteSP("proc_PadroesAnalise_Search",
      new SqlParameter("@idProjeto", (projetoID == 0 ? 0 : projetoID)),
      new SqlParameter("@Nome", (nome == "" ? null : nome)),
      new SqlParameter("@Contexto", (contexto == "" ? null :
contexto)),
      new SqlParameter("@Problema", (problema == "" ? null :
problema)),
      new SqlParameter("@Solucao", (solucao == "" ? null : solucao)),
      new SqlParameter("@Aplicabilidade", (aplicabilidade == "" ? null
: aplicabilidade)),
      new SqlParameter("@ContextoResultante", (contextoResultante == ""
? null : contextoResultante)),
      new SqlParameter("@Exemplo", (exemplo == "" ? null : exemplo)),
      new SqlParameter("@Forcas", (forcas == "" ? null : forcas)),
      new SqlParameter("@UsosConhecido", (usosConhecido == "" ? null :
usosConhecido)));

        while (reader.Read())
        {
            Padrao padrao = new Padrao();
            padrao.PadraoID = reader.GetInt32("idPadrao");
            padrao.Nome = reader.GetString("Nome");
            padrao.Contexto = reader.GetString("Contexto");
            padrao.Problema = reader.GetString("Problema");
            padrao.Solucao = reader.GetString("Solucao");
            padrao.Aplicabilidade = reader.GetString("Aplicabilidade");
            padrao.ContextoResultante =
reader.GetString("ContextoResultante");
            padrao.Exemplo = reader.GetString("Exemplo");
            padrao.Forcas = reader.GetString("Forcas");
            padrao.UsosConhecido = reader.GetString("UsosConhecido");

            padroes.Add(padrao);
        }

        return padroes;
    }
```

Após o objeto *Reader* obter a coleção de dados referente à pesquisa retornada pelo *Stored Procedure* é necessário realizar os *Set* dos valores da classe "*PadraoAnalise*" para a

visualização na camada de apresentação. A Tabela 7 demonstra a construção da pesquisa dos padrões de análise dentro de uma *Stored Procedure*.

Tabela 7: *Stored Procedure (*proc_PadroesAnalise_Search)

```
CREATE PROCEDURE dbo.proc_PadroesAnalise_Search
(
      @idProjeto              INT = NULL,
      @Nome                         VARCHAR(50) = NULL,
      @Contexto               VARCHAR(255) = NULL,
      @Problema               VARCHAR(255) = NULL,
      @Solucao                VARCHAR(255) = NULL,
      @Aplicabilidade   VARCHAR(255) = NULL,
      @ContextoResultante VARCHAR(255) = NULL,
      @Exemplo                VARCHAR(255) = NULL,
      @Forcas                 VARCHAR(255) = NULL,
      @UsosConhecido          VARCHAR(255) = NULL
      )
      AS

      SELECT
      a.idPadrao,
      a.Nome,
      a.Contexto,
      a.Problema,
      a.Solucao,
      a.Aplicabilidade,
      a.ContextoResultante,
      a.Exemplo,
      a.Forcas,
      a.UsosConhecido

      FROM dbo.PadraoAnalise a
            LEFT JOIN dbo.PadraoProjeto p ON
                  p.idPadrao = a.idPadrao

      WHERE
      (@idProjeto = 0 OR p.idProjeto = @idProjeto) AND
      (@Nome IS NULL OR a.Nome LIKE '%' + @Nome + '%') AND
      (@Contexto IS NULL OR a.Contexto LIKE '%' + @Contexto + '%') AND
      (@Problema IS NULL OR a.Problema LIKE '%' + @Problema + '%') AND
      (@Solucao IS NULL OR a.Solucao LIKE '%' + @Solucao + '%') AND
      (@Aplicabilidade IS NULL OR a.Aplicabilidade LIKE '%' +
@Aplicabilidade + '%') AND
      (@ContextoResultante IS NULL OR a.ContextoResultante LIKE '%' +
@ContextoResultante + '%') AND
      @Exemplo IS NULL OR a.Exemplo LIKE '%' + @Exemplo + '%') AND
      (@Forcas IS NULL OR a.Forcas LIKE '%' + @Forcas + '%') AND
      (@UsosConhecido IS NULL OR a.UsosConhecido LIKE '%' +
@UsosConhecido)

GO
```

A implementação se preocupou com a interoperabilidade dos dados não armazenando o conteúdo das informações somente em uma base restrita ao sistema RPAO. O sistema se propõe a gerar dados em ontologias, que já foram descritas anteriormente.

A construção da ferramenta focou-se em apresentar resultados aos engenheiros que a utilizarão. Foi implementado um gráfico de uso dos padrões de análise em projetos que apresenta os resultados do reuso dentro da ferramenta. A Figura 46 mostra a tela de geração de gráfico de reuso.

Figura 46: Gráfico de reuso Padrões de Análise X Projetos

A classe "*PadraoAnalise"* possui um método o qual retorna uma Coleção de dados do tipo *List*, o qual é criado da mesma forma que o método já apresentado em "*pesquisarPadroes*". A Tabela 8 apresenta o código de criação do método que retorna através do objeto "*PadroesProjeto* à coleção dos dados do *storedProcedure*.

Tabela 8: Método Estático ListarPadoresProjetos

```
public static List<PadraoProjeto> ListaPadroesProjetos()
    {
        List<PadraoProjeto> padroesProjeto = new
List<PadraoProjeto>();

        SmartDataReader reader =
DBUtils.ExecuteSP("proc_ProjetoPadrao");

        while (reader.Read())
        {
            PadraoProjeto padraoProjeto = new PadraoProjeto();
            padraoProjeto.PadraoID = reader.GetInt32("idPadrao");
            padraoProjeto.ProjetoID = reader.GetInt32("idProjeto");
            padraoProjeto.Nome = reader.GetString("nome");

            padroesProjeto.Add(padraoProjeto);
        }

        return padroesProjeto;
    }
}
```

A Figura 47 representa o diagrama de classes para a geração do código onde foi aplicado um padrão de projeto chamado *Façade,* conforme definição vista na seção 5.2. Desta forma, o objetivo da classe "*GeradorGraficoFacade*" é ser uma classe "fachada" para acessar as demais classes no subsistema. Tal classe possui a função de apresentar os valores que serão visualizados no gráfico, como cor, nome de eixos e dados os quais precisam ser apresentados para a geração do gráfico.

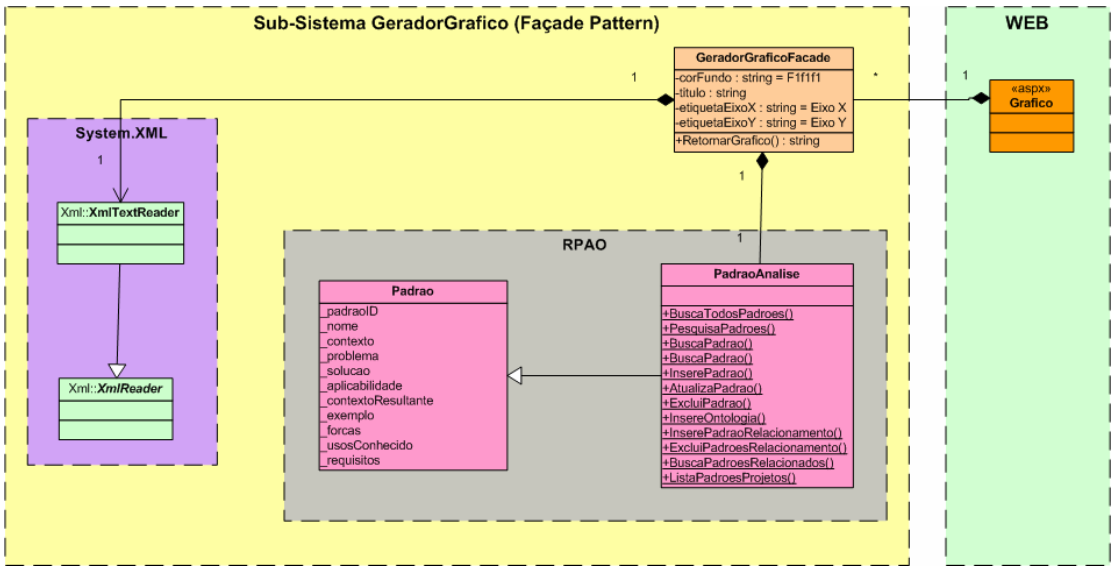

Figura 47: Gráfico de reuso Padrões de Análise X Projetos

4.5. Camada de Interoperabilidade do RPAO

No contexto desta pesquisa, a camada de interoperabilidade é responsável por duas funcionalidades: a primeira consiste em permitir que agentes de software ou sistemas externos possam obter informações a respeito dos padrões de análise gerados pela ferramenta, e a segunda é servir de base a analistas que pretendem desenvolver ou melhorar uma aplicação que possa utilizar os dados descobertos e catalogados no RPAO.

A ontologia fornece a estrutura dos padrões de análise descobertos e catalogados. Assim, o agente externo através da conexão via *web service* deve requisitar a ontologia que deseja para visualizar e usufruir em seu sistema. A ferramenta importa arquivos XMI para interoperar modelos UML. As seções a seguir detalharão as funcionalidades da Ontologia, do XMI e do *Web Service* na camada de interoperabilidade da arquitetura desenvolvida.

4.5.1. Utilizando XMI para importação de Modelos

Diante do processo de reuso, a ferramenta se propõe ainda a reusar o modelo UML desenhado pelos projetistas. A ferramenta trabalha com padrões de análise e requisitos, portanto, todo o requisito é transformado em caso de uso. O engenheiro pode transformar o seu modelo em um formato universal XMI e, assim, a ferramenta poderá ler este modelo e apresentar os dados em uma listagem para que o engenheiro possa relacioná-lo a requisitos e continuar o processo de cadastro.

Para a solução de importação de um arquivo XMI foi necessário implementar um *Parser*, apresentado na Tabela 9 pelo método "*NodeParse*". O método recebe um caminho do arquivo e então utiliza o objeto nativo do .NET *XmlTextReader*, o qual possui implementação para leitura de arquivos do tipo XML. Após a instanciação é realizada uma pesquisa pelos nodos que, para o caso, será necessário que o nodo seja um elemento do tipo *Model* e que o *LocalName* seja a palavra *UseCase*.

Tabela 9: Método leitura XMI

```
public NodeParse(string URLParse)
            {
                  _nodes = new Hashtable();
                  _readXMI = new XmlTextReader(URLParse);
                  while (_readXMI.Read())
                  {
                        if (_readXMI.NodeType == XmlNodeType.Element)
                        {
                              switch (_readXMI.LocalName)
                              {
                                    case "UseCase":
                                          AddNode(_readXMI);
                                          break;
                              }
                        }
                        else if (_readXMI.NodeType ==
XmlNodeType.Attribute)
                        {
                        }
                  }
            }
```

A Tabela 10 apresenta com tarjas, o formato do arquivo XMI e o que a ferramenta interpreta para a importação dos dados.

Tabela 10: Casos de uso em arquivo XMI

```
<UML:Model xmi.id="UMLProject.1">
- <UML:Namespace.ownedElement>
- <UML:Model xmi.id="UMLModel.2" name="Use Case Model" visibility="public"
isSpecification="false" namespace="UMLProject.1" isRoot="false" isLeaf="false"
isAbstract="false">
- <UML:Namespace.ownedElement>
  <UML:Actor xmi.id="UMLActor.3" name="Engenheiro Software" visibility="public"
isSpecification="false" namespace="UMLModel.2" isRoot="false" isLeaf="false" isAbstract="false"
participant="UMLAssociationEnd.8 UMLAssociationEnd.11 UMLAssociationEnd.14" />
  <UML:UseCase xmi.id="UMLUseCase.4" name="Manter Padroes" visibility="public"
isSpecification="false" namespace="UMLModel.2" isRoot="false" isLeaf="false" isAbstract="false"
participant="UMLAssociationEnd.9" />
  <UML:UseCase xmi.id="UMLUseCase.5" name="Manter Usuário" visibility="public"
isSpecification="false" namespace="UMLModel.2" isRoot="false" isLeaf="false" isAbstract="false"
participant="UMLAssociationEnd.12" />
  <UML:UseCase xmi.id="UMLUseCase.6" name="Manter Viagens" visibility="public"
isSpecification="false" namespace="UMLModel.2" isRoot="false" isLeaf="false" isAbstract="false"
participant="UMLAssociationEnd.15" />
```

Na Figura 48 é apresentada a implementação da importação de arquivos XMI para reuso de informações já contidas em modelos UML, restringidas em diagramas de casos de uso. A tela apresenta um campo caminho do arquivo XMI, e também um botão para executar a importação do arquivo e, após a finalização da importação, são apresentados na lista os

casos de uso encontrados no arquivo. A ferramenta permite que o engenheiro realize a edição destes dados, assim direcionando o caso de uso para o cadastro de requisitos.

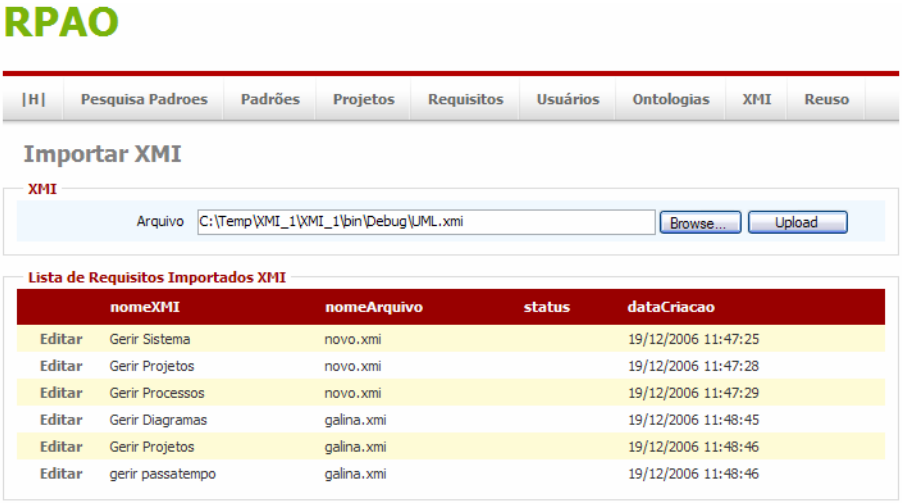

Figura 48: Importação XMI

Na próxima subseção é apresentado o projeto de construção da ontologia que é unida à geração que o RPAO faz com a ontologia pré-definida.

4.5.2. **Ontologia dos Padrões**

A ontologia a ser desenvolvida representa o conhecimento sobre padrões de análise estendendo aos demais padrões de sistemas. Por meio da ontologia proposta são explicitados os conceitos comumente utilizados no que se refere à descrição e representação de sistemas de padrões, bem como as restrições aplicadas a esses conceitos, provendo dessa forma, um vocabulário comum entre os participantes do projeto. Ela também permite a especificação de padrões isolados, ou seja, padrões que não estão vinculados a um sistema de padrões específico.

O processo de construção da ontologia seguiu o método proposto por Bechhofer [BECH 06] e consiste em duas fases: a especificação e o projeto da ontologia. Na fase de especificação é utilizado o conhecimento sobre padrões e sistemas de padrões para a

construção da rede semântica. Na fase de projeto é realizado o mapeamento da rede semântica em uma ontologia baseada em *frames*, gerando assim a *Onto-APattern*. Na Figura 49 é ilustrado este processo.

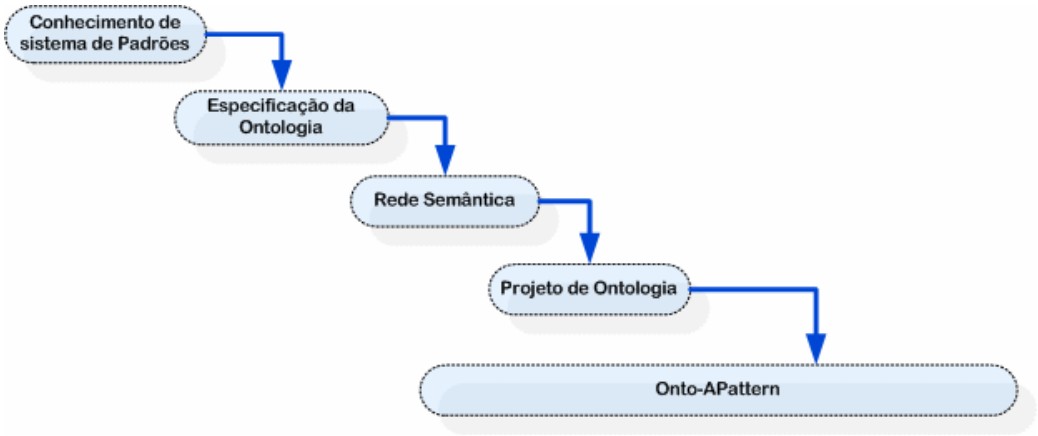

Figura 49: Processo da construção da Onto-APattern (ontologia de padrões de análise).

Já na Figura 50 é mostrada a parte da rede semântica que representa os atributos de descrição de um padrão e a sua classificação. O formato de descrição de padrões adotado é constituído dos atributos mais comumente utilizados: nome, problema, contexto, forças, solução, exemplo, usos conhecidos, contexto resultante, padrões relacionados, sendo obrigatória a especificação dos cinco primeiros atributos. Um sistema de padrões é composto por um nome, uma descrição por um conjunto de padrões e pelos seus relacionamentos.

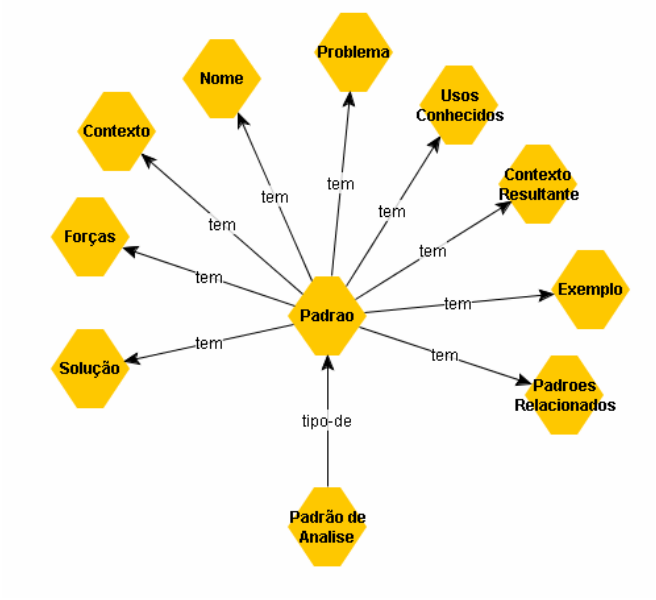

Figura 50: Rede semântica representando os atributos e calcificações dos padrões.

A Figura 51 mostra parte da rede semântica representando um sistema de padrões que será expressa em ontologias através de classes e atributos.

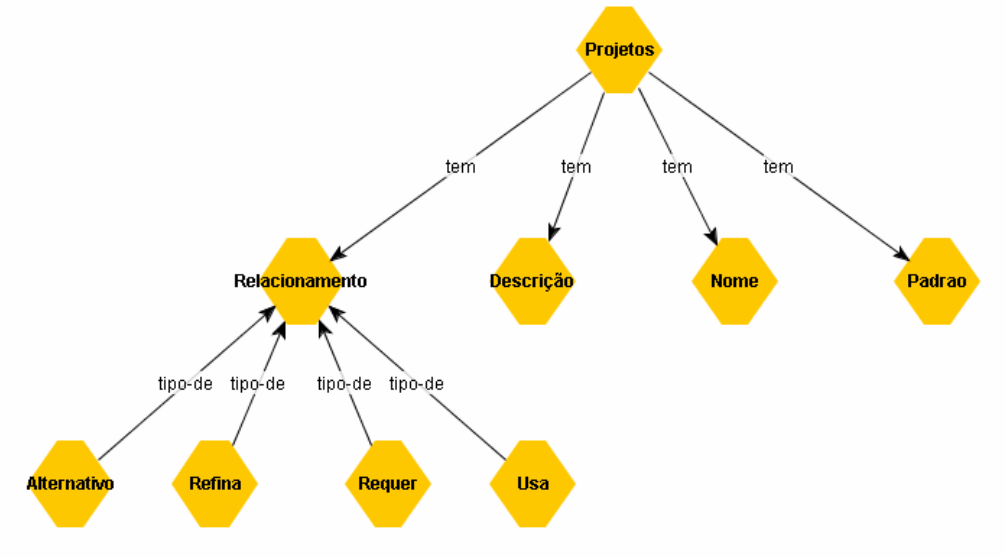

Figura 51: Rede semântica representando o sistema de padrões da Ontologia.

A figura 52 apresenta a estrutura da ontologia para as classes Requisito e Artefatos que surgiram para relacionarem-se com a classe PadraoAnalise.

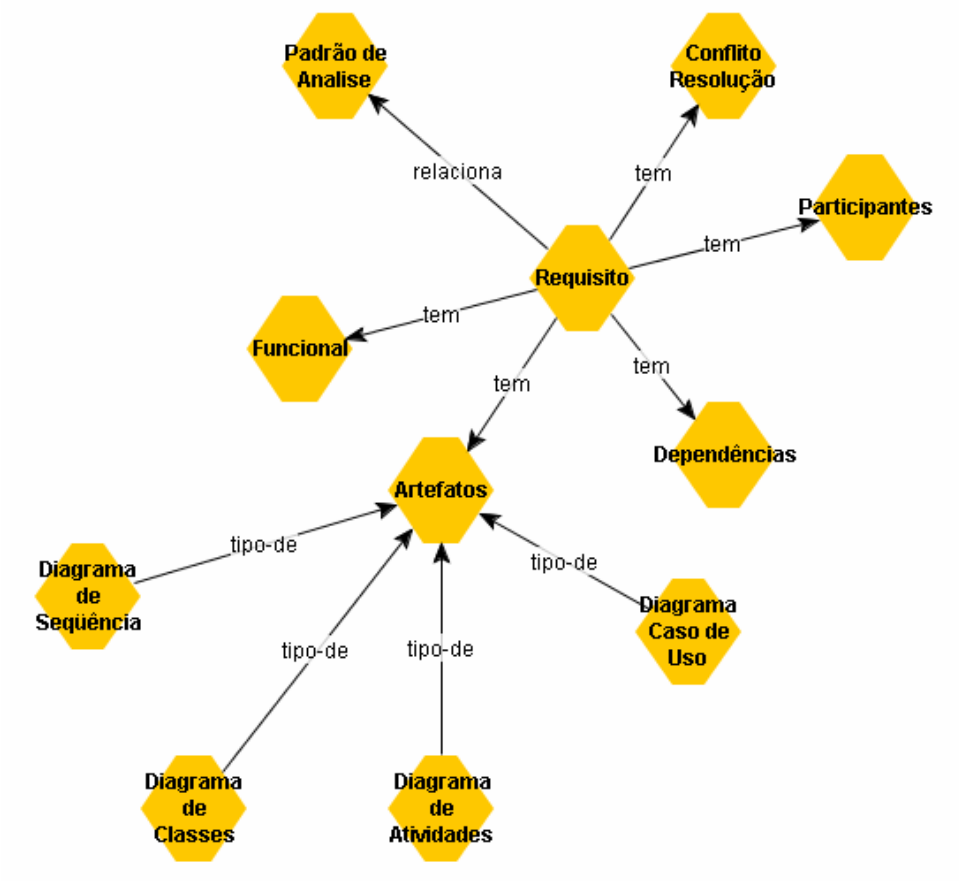

Figura 52: Rede semântica evolvendo requisitos e artefatos

A ferramenta proposta para o desenvolvimento da ontologia é o *Protégé* [PROT 06]. O *Protégé* é um sistema integrado de desenvolvimento e gerenciamento de bases de conhecimento. Uma ontologia no *Protégé* consiste basicamente de:

- Classes: conceitos do domínio abordado que constituem uma hierarquia taxonômica;
- *Slots*: descrevem propriedades de classes e instâncias;
- Facetas: descrevem propriedades de *slots* e permitem a especificação de restrições nos valores dos *slots*;

Na fase do projeto da ontologia, os conceitos e relacionamentos da rede semântica são mapeados para a ontologia. Os nós são mapeados como classes. Os nós relacionados por um link do tipo “tipo-de” são mapeados em uma hierarquia de subclasses e superclasses. Outros

tipos de relacionamentos são mapeados para *slots* das correspondentes classes. Cada *slot* é associado a uma faceta adequada como tipo e cardinalidade.

A Figura 53 apresenta a classe Padrao da Onto-APattern (nome da ontologia neste trabalho) que possui como subclasses padrão arquitetural, padrão de análise, padrão de projeto, padrão de linguagem e padrão conceitual.

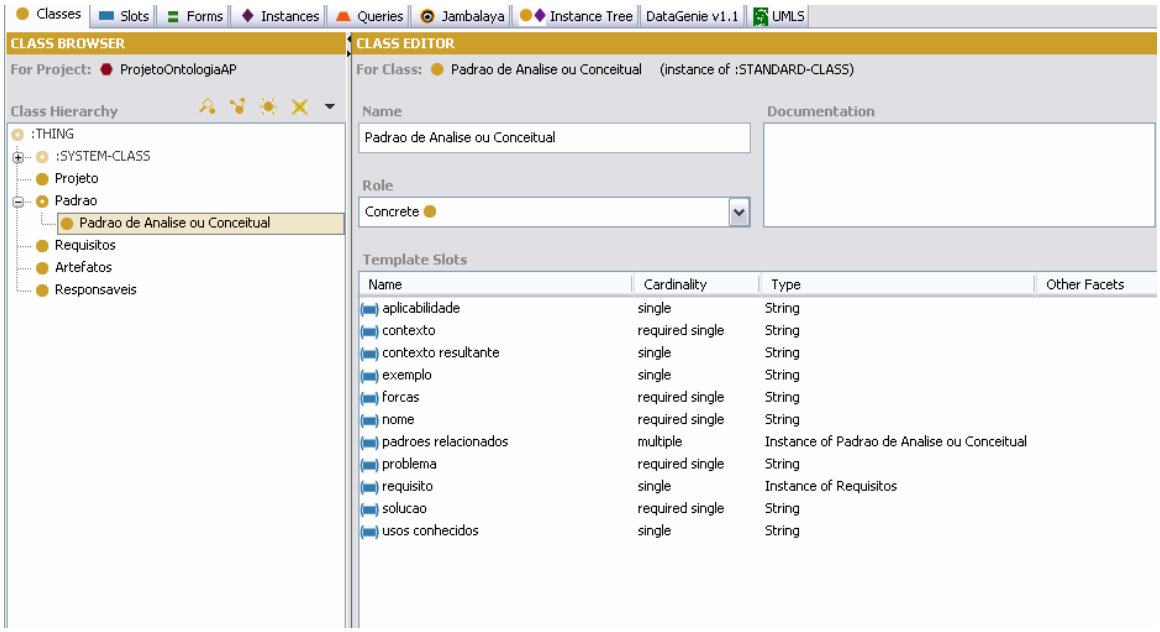

Figura 53: Hierarquia de Classes da Onto-APattern.

Os relacionamentos entre os padrões pertencentes a um mesmo sistema de padrões foram representados por meio da instanciação das subclasses de relacionamento (Figura 52) que são: Substitui, Refina, Requer, Usa, de acordo com o formalismo apresentado na seção de *design-patterns.*

Os *slots* que fazem parte da classe "*Padrão*": usos conhecidos, nome, problema, forças, exemplo, contexto, contexto resultante, solução, aplicabilidade e padrões relacionados são mostrados na figura abaixo.

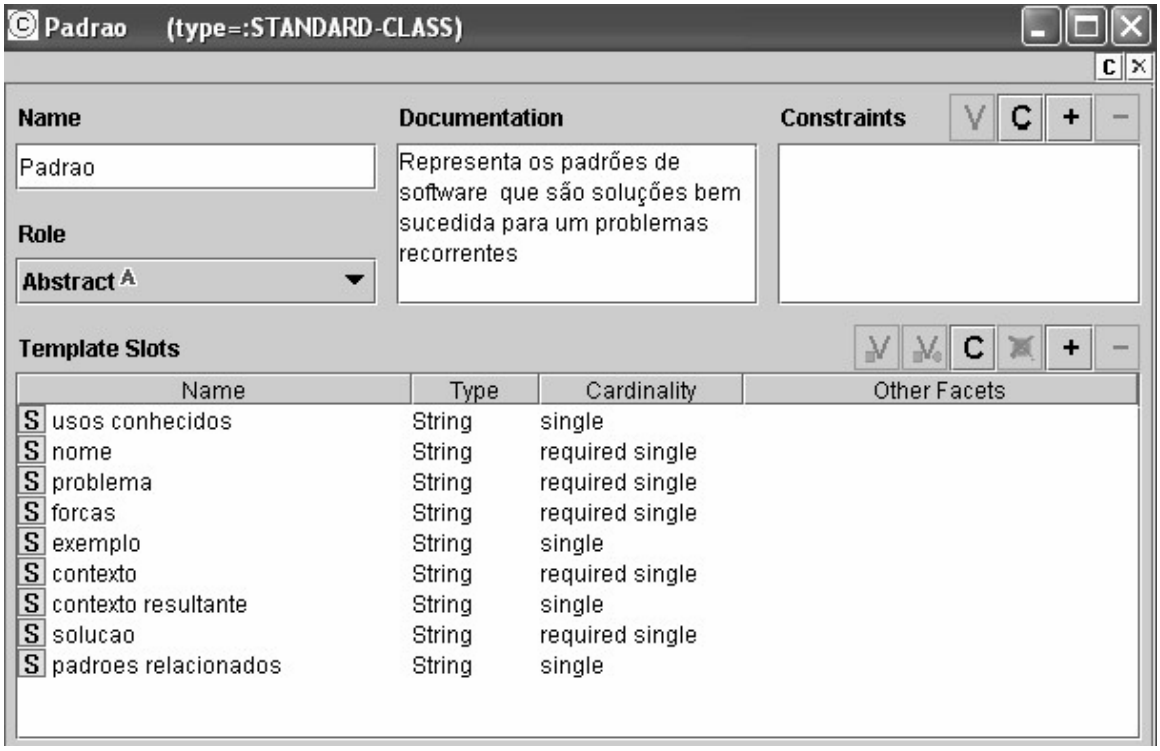

Figura 54: *Slots* da classe Padrão.

É apresentado na Tabela 11 um trecho da ontologia criada e já mencionada nesta seção. Esta ontologia será a ontologia base para a criação de todas as outras ontologias dos padrões de análise, podendo assim dizer que essa ontologia criada servirá de estrutura para as demais ontologias realizarem um "*include*" da ontologia apresentada.

Tabela 11: Ontologia criada ontoAPattern.

```
<?xml version="1.0" ?>
- <rdf:RDF xmlns="http://localhost:8080/ap.owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#" xml:base="http://localhost:8080/ap.owl">
  <owl:Ontology rdf:about="" />
- <owl:Class rdf:ID="Padrao">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Representa os
padrões de software que são soluções bem sucedida para um problemas
recorrentes</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Projeto" />
  <owl:Class rdf:ID="Artefatos" />
  <owl:Class rdf:ID="Relacionamento" />
  <owl:Class rdf:ID="Participante" />
- <owl:Class rdf:ID="PadraoAnalise">
  <rdfs:subClassOf rdf:resource="#Padrao" />
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Padrao de Analise
ou Conceitual</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="Requisitos" />
- <owl:ObjectProperty rdf:ID="requisito">
- <rdfs:domain>
- <owl:Class>
- <owl:unionOf rdf:parseType="Collection">
  <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing" />
  <owl:Class rdf:about="#Padrao" />


  </owl:unionOf>
  </owl:Class>
  </rdfs:domain>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="relacionamentos" />
- <owl:ObjectProperty rdf:ID="padroes">
  <rdfs:domain rdf:resource="#Projeto" />
  </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="padraoRelacionado">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">O padrao que
está se relacionando.</rdfs:comment>
  <rdfs:domain rdf:resource="#Relacionamento" />
  </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="participantes">
  <rdfs:domain rdf:resource="#Requisitos" />
  </owl:ObjectProperty>


  </rdf:RDF>
```

A ontologia mestre apresentada fica publicada em um servidor de aplicação para que as demais ontologias criadas pela aplicação RPAO possam instanciar *OnLine* a ontologia mestre. O usuário ao abrir uma ontologia gerada pela ferramenta de um padrão de análise estará abrindo, juntamente, a estrutura desta ontologia sem perceber.

A ferramenta RPAO disponibiliza ao engenheiro de software a opção de geração da ontologia sob um padrão criado. O engenheiro, ao escolher a opção gerar na coluna ontologia, determina a criação da ontologia nos padrões que possam ser manipulados posteriormente dentro de uma ferramenta como *Protégé*. A Figura 55 apresenta o processo desta geração de ontologia.

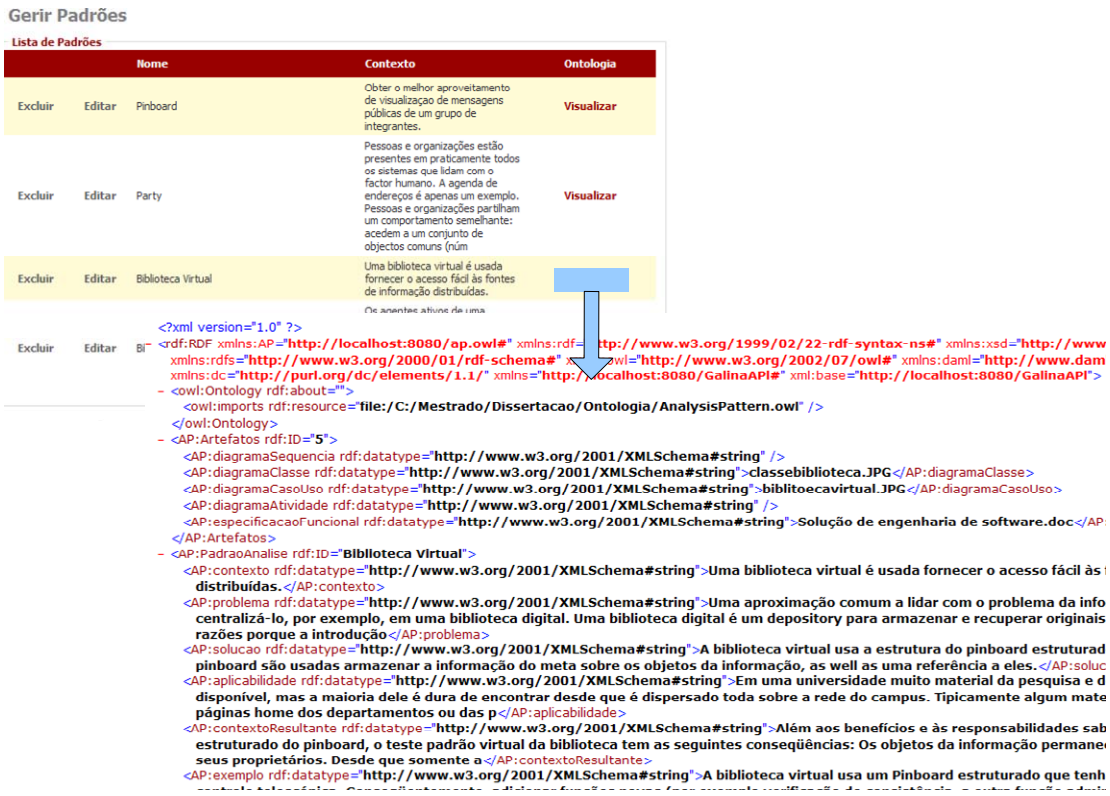

Figura 55: Processo de geração da Ontologia por RPAO

A Tabela 12 apresenta o código do método *GerarOntologia* da classe Ontologia que realiza a geração de ontologias do padrão de análise armazenado na ferramenta RPAO. Dentro do código de geração é incluído o endereço da ontologia mestre que é utilizada para realizar o "*include"* da estrutura original.

Tabela 12: Método de criação da ontologia.

```
protected void GerarOntologia (object sender, EventArgs e)
    {
// Recupera o código do padrão
int padraoID = Convert.ToInt32(((LinkButton)sender).CommandArgument);
// Busca o padrão com requisito, artefato e participantes
Padrao padrao = Padroes.BuscaPadrao(padraoID, true);
// Recupera caminho para salvar ontologia gerada
string pastaOntologias =
Server.MapPath(ConfigurationManager.AppSettings["PastaOntologias"]);
string path = pastaOntologias + "ap_" + padraoID.ToString() + ".owl";

using (StreamWriter sw = new StreamWriter(path))
 {
Requisito requisito = padrao.Requisitos[0];

sw.WriteLine(@"<?xml version=""1.0"" ?>");
sw.WriteLine(@"<rdf:RDF");
sw.WriteLine(@"    xmlns:AP=""http://localhost:8080/ap.owl#""");
sw.WriteLine(@"    xmlns:rdf=""http://www.w3.org/1999/02/22-rdf-
syntax-ns#""");
sw.WriteLine(@"    xmlns:xsd=""http://www.w3.org/2001/XMLSchema#""");
sw.WriteLine(@"    xmlns:rdfs=""http://www.w3.org/2000/01/rdf-
schema#""");
sw.WriteLine(@"    xmlns:owl=""http://www.w3.org/2002/07/owl#""");
sw.WriteLine(@"
xmlns:daml=""http://www.daml.org/2001/03/daml+oil#""");
sw.WriteLine(@"    xmlns:dc=""http://purl.org/dc/elements/1.1/""");
sw.WriteLine(@"    xmlns=""http://localhost:8080/GalinaAPl#""");
sw.WriteLine(@"    xml:base=""http://localhost:8080/GalinaAPl"">");
sw.WriteLine(@"    <owl:Ontology rdf:about="""">");
sw.WriteLine(@"        <owl:imports rdf:resource=""
http://localhost:8080/AnalysisPattern.owl"" />");
sw.WriteLine(@"    </owl:Ontology>");

sw.WriteLine(@"   ");
sw.WriteLine(@"    </AP:PadraoAnalise>");
sw.WriteLine(@"</rdf:RDF>");

// Salva arquivo xml
sw.Close();

// Salva padrão ontologia
Padroes.InsereOntologia(padraoID,
ConfigurationManager.AppSettings["PastaOntologias"] + "ap_" +
padrao.PadraoID.ToString() + ".owl");


}
```

Ontologia Mestre - Estrutura

4.5.3. **Comunicação do *Web Service* com o RPAO**

O desenvolvimento de *Web Services* em .NET já proporciona algumas agilidades na comunicação e desenvolvimento da solução não necessitando criar uma solução em conjunto para a troca de informação de XML e utilização do SOAP ou WSDL, pois o .NET já realiza automaticamente.

A codificação de um *Web Service* em .NET é realizada em um arquivo com extensão "cs" que é interpretado pelo servidor de aplicação como uma página. O *Web Service* criado para interoperar as informações contidas nas ontologias foi nomeado de "*OntologyService.asmx"*. A criação de métodos e instâncias de objetos é realizada da mesma forma que uma classe codificada em C#. Para a criação de métodos é necessário identificá-lo como um atributo "[*WebMethod*]", assim o compilador irá expor o método na página do *web service*. A Tabela 12 apresenta a codificação da *Web Service* e a criação de um *WebMehtod* público que retorna uma string contendo a ontologia, a qual está sendo retornada pela classe *Util* do RPAO.

Tabela 13: Código de criação do *Web Service*

```
Using System;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;


/// <summary>
/// Criaçao do webservice para comunicaçao da ferramenta.
/// </summary>
[WebService(Namespace = "http://cgalina.googlepages.com/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class OnthologyService : System.Web.Services.WebService {

    public OnthologyService () {

        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
        //Método que retorna a ontologia gerada no servidor em disco,
transporta para o cliente a ontologia desejada identificada pelo ID.

    public string RetornaOntologia(int id)
    {
        return Util.RetornaOntologia(id);
    }
}
```

O *Web Service* depois de compilado é executado através de um *browser* para realizar os testes dos métodos criados. A Figura 56 demonstra o *Web Service* OnthologyService e o método RetornaOntologia implementado.

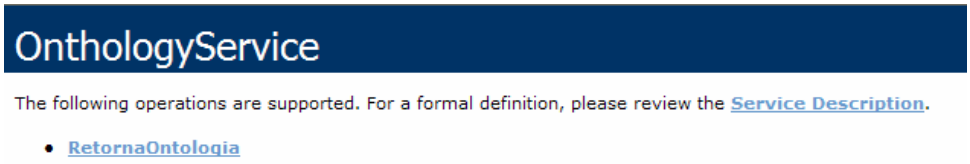

Figura 56: Método RetornaOntologia do *Web Service.*

Como já mencionado no inicio da seção, o.NET gera automaticamente o arquivo WSDL que contém todas as assinaturas dos métodos criados, *namespaces* e URL do *web service* que trabalha como uma biblioteca apresentando para o cliente as funcionalidades disponibilizadas aos clientes. A Tabela 14 apresenta um trecho deste arquivo WSDL gerado, sua definição e funcionamento pode ser visto detalhadamente no capítulo 3.9.3.

Tabela 14: Descrição do arquivo WSDL.

```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://cgalina.googlepages.com/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://cgalina.googlepages.com/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
- <wsdl:types>
- <s:schema elementFormDefault="qualified"
targetNamespace="http://cgalina.googlepages.com/">
- <s:element name="RetornaOntologia">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="id" type="s:int" />
  </s:sequence>
  </s:complexType>
  </s:element>
- <s:element name="RetornaOntologiaResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="RetornaOntologiaResult"
type="s:string" />
  </s:sequence>
  </s:complexType>
  </s:element>
  </s:schema>
  </wsdl:types>
- <wsdl:message name="RetornaOntologiaSoapIn">
  <wsdl:part name="parameters" element="tns:RetornaOntologia" />
  </wsdl:message>
- <wsdl:message name="RetornaOntologiaSoapOut">
  <wsdl:part name="parameters" element="tns:RetornaOntologiaResponse" />
  </wsdl:message>
- <wsdl:portType name="OnthologyServiceSoap">
- <wsdl:operation name="RetornaOntologia">
  <wsdl:input message="tns:RetornaOntologiaSoapIn" />
  <wsdl:output message="tns:RetornaOntologiaSoapOut" />
  </wsdl:operation>
  </wsdl:portType>

 . . . . . .
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

Com a execução do método apresentado na Figura 56 é possível visualizar uma página, que serve para realizar testes simples dos *Web Services*, não precisando criar um

cliente que chame externamente um *Web Service*. A página também apresenta como devem ser implementados os métodos GET e POST para requisição e resposta do serviço, respectivamente nos formatos das versões 1.1 e 1.2 do protocolo SOAP, como ilustrado na Figura 56.

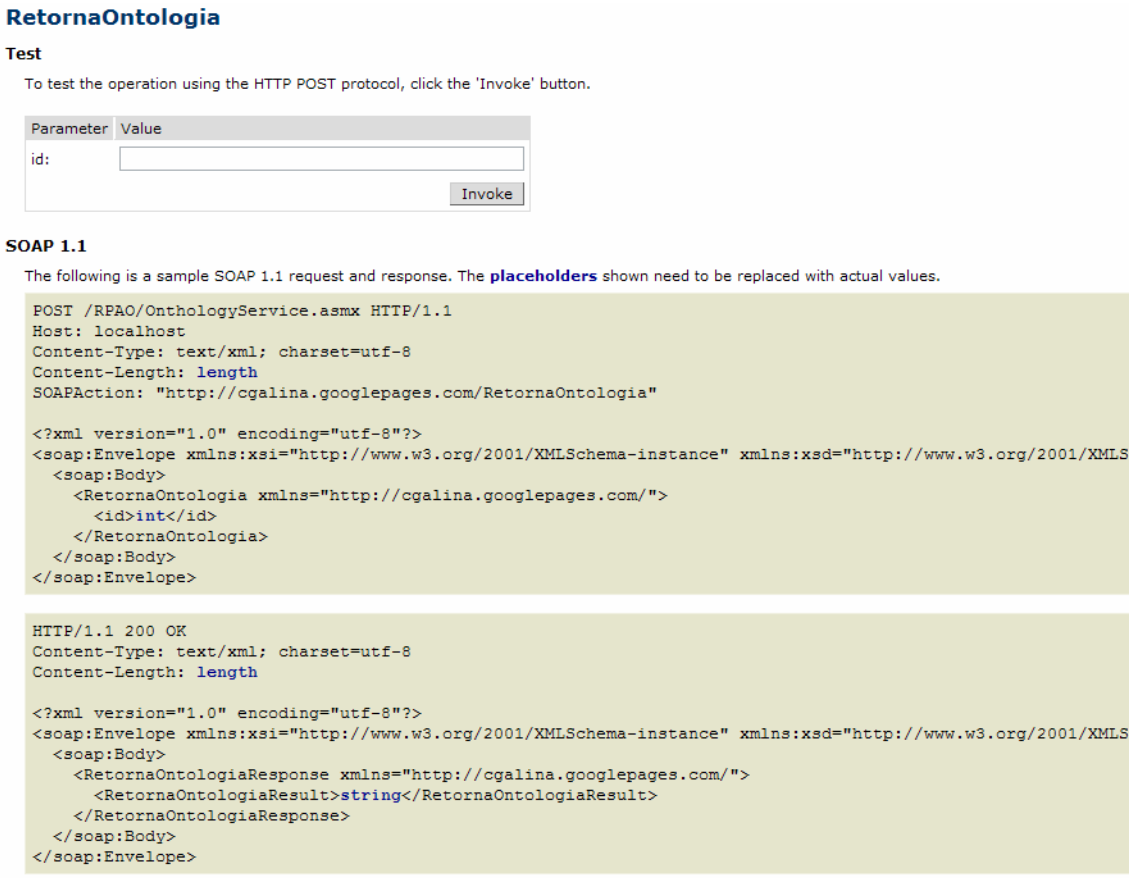

Figura 57: Método RetornaOntologia do *Web Service*.

O *Web Service* desta forma está pronto para utilização de qualquer sistema ou agente externo que deseje reusar as ontologias gerada pela ferramenta, bastando simplesmente configurar sua aplicação para conexão com o *Web Service* do RPAO. Na próxima seção é apresentado o processo de utilização de todas as seções já descritas neste capítulo.

4.6. *Worflow* do RPAO

Foi elaborada uma seqüência de atividades que indicam como o engenheiro de software deverá proceder para trabalhar com o ambiente proposto. A Figura 28 apresenta, de forma resumida, o processo da solução. O Engenheiro de software acessa a aplicação a partir de seu navegador (*browser).* O sistema apresentará uma tela de validação de usuário e senha, necessitando assim que o engenheiro possua permissão para acesso ao repositório. Após a validação do usuário são apresentadas as opções de gerenciamento dentro do repositório. Dentre todas as opções que o sistema disponibiliza para o engenheiro, pesquisar padrões de análise é a opção com maior grau de utilização do sistema, pois permite a pesquisa por qualquer campo do *template* dos padrões. Funcionalidade esta já descrita na seção 5.3.

O usuário ao processar sua requisição de: pesquisa de padrões de análise, criação de padrão, criação de projetos e criação de requisitos, requisita transações da base de dados para o retorno dos dados.

Os dados de armazenamento de padrões de análise atendem às fases de iniciação e elaboração do RUP. O repositório se propõe a armazenar padrões diagnosticados e, com eles, o mapeamento dos requisitos que tornaram esses padrões perceptíveis. Esse gerenciamento de documentos e artefatos, como pode ser visto na seção 3.8.1, caracteriza a utilização das fases de iniciação onde é identificado o “problema” e na fase de elaboração é descrita a “solução” através de artefatos e documentos. Dentre eles a ferramenta RPAO armazena os seguintes artefatos: diagrama de atividades, diagrama de classe, diagrama de casos de uso, diagrama de seqüência e documento textual de Especificação do Requisito.

O engenheiro ao finalizar seu gerenciamento sobre os padrões realiza a geração da ontologia. O processo de geração é apresentado na seção 5.5.2. A partir deste processo já serão disponibilizadas pelo servidor as ontologias para reuso de sistemas ou agentes externos. Esta utilização só será possível através de uma camada de interoperabilidade que a ferramenta fornecerá, descrita na seção anterior.

A camada é composta por um *Web Service* que realiza a comunicação entre o sistema externo e a ferramenta RPAO. Através desta comunicação o cliente externo poderá requisitar a ontologia desejada, e então, o *Web Service* requisita ao RPAO que disponibilize a ontologia para o cliente, sendo o *Web Service* responsável por trafegar os dados da ontologia para o cliente. Este fluxo descrito a cima pode ser visualizado na Figura 58.

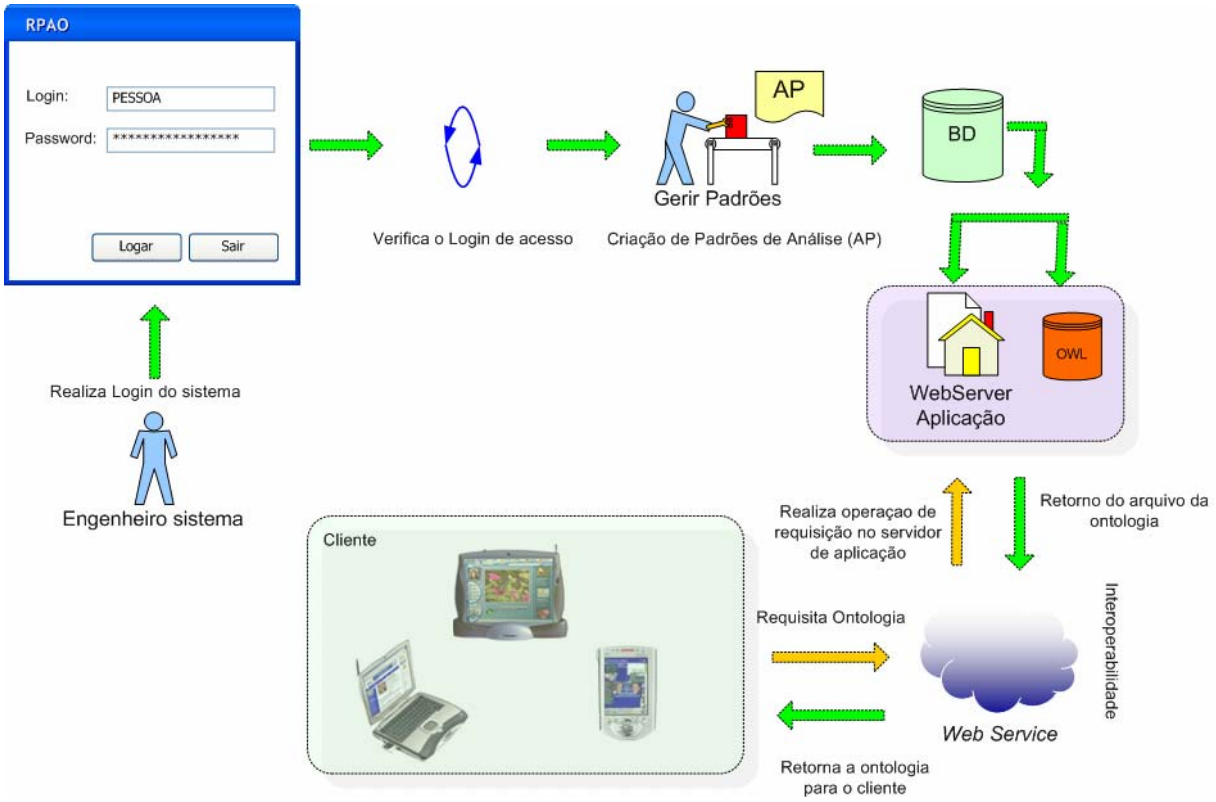

Figura 58: *WorkFlow* do RPAO.

5. ESTUDO DE CASO

O estudo de caso aqui apresentado preocupa-se com a avaliação dos benefícios do repositório de padrões de análise e com a apresentação da eficácia de obter padrões de análise, requisitos e artefatos sob gerenciamento através de uma ferramenta. Neste sentido, o estudo de caso tem o propósito de:

- Apresentar como é realizada a inclusão de padrões de análise e a busca dos mesmos dentro da ferramenta;
- A geração de ontologias após a inclusão de padrões de análise e requisitos relacionados aos padrões de análise;
- Associação dos padrões catalogados por projetos, podendo ser apresentada a reutilização dos padrões nos projetos correlacionados;
- Apresentar a importação de modelos de caso de uso através de arquivo XMI para associação de padrões dentro da ferramenta;
- E por ultimo a visualização dos padrões de análise associados em projetos através de um gráfico.

Para buscar o real potencial da ferramenta é importante utilizá-la dentro de uma entidade que já utilize uma metodologia de desenvolvimento e deseje melhorar seus processos de documentação e exploração de padrões, para tal foco, a ferramenta irá apresentar com o tempo um ganho importante no reuso e descoberta de padrões na base de conhecimento.

A empresa de software Tlantic pertencente ao grupo SONAE, fábrica de software e especializada em sistemas de comércio eletrônico e situada no pólo tecnológico da PUC do Rio Grande do Sul demonstrou interesse pela ferramenta, já que está iniciando o processo de CMMI3 (*Capability Maturity Model Integration*). Em colaboração com esse processo, a ferramenta facilita a instrumentação da fase de análise e armazenamento de padrões de análise relacionada aos PATs (*Process Action Team*) de RM (*Requeriments Management*) e RD (*Requeriments Development*). Inicialmente, não foi possível coletar muitos dados para apresentação de resultados, porém a equipe de qualidade da empresa apresentou alguns recentes requisitos que podiam se tornar padrões para a empresa futuramente.

A ferramenta foi publicada na *intranet* da empresa e pôde ser acessada através do endereço http://intranet-rpao.tlantic.com.br apresentando a tela inicial de controle de acesso para validar o usuário. A Figura 59 apresenta a seqüência de telas para a criação do padrão de análise. Após a validação do usuário, é escolhida a opção Padrões (2), apresentada no menu da tela e que retorna a lista dos padrões criados na ferramenta. Foi criado um novo padrão para avaliar a ferramenta (3), o nome deste padrão de análise é *Pinboard* que, segundo a equipe, é utilizado em muitos projetos que desejam divulgar informações sem ter a necessidade de realizar reuniões.

Foi inserido o padrão dentro do RPAO e acrescentados outros três padrões aleatoriamente que se encontravam nos projetos. Então, para o estudo de caso, foram introduzidos os seguintes padrões de análise: *Party*, Biblioteca Digital, Biblioteca Digital em Agentes e *Pinboard*.

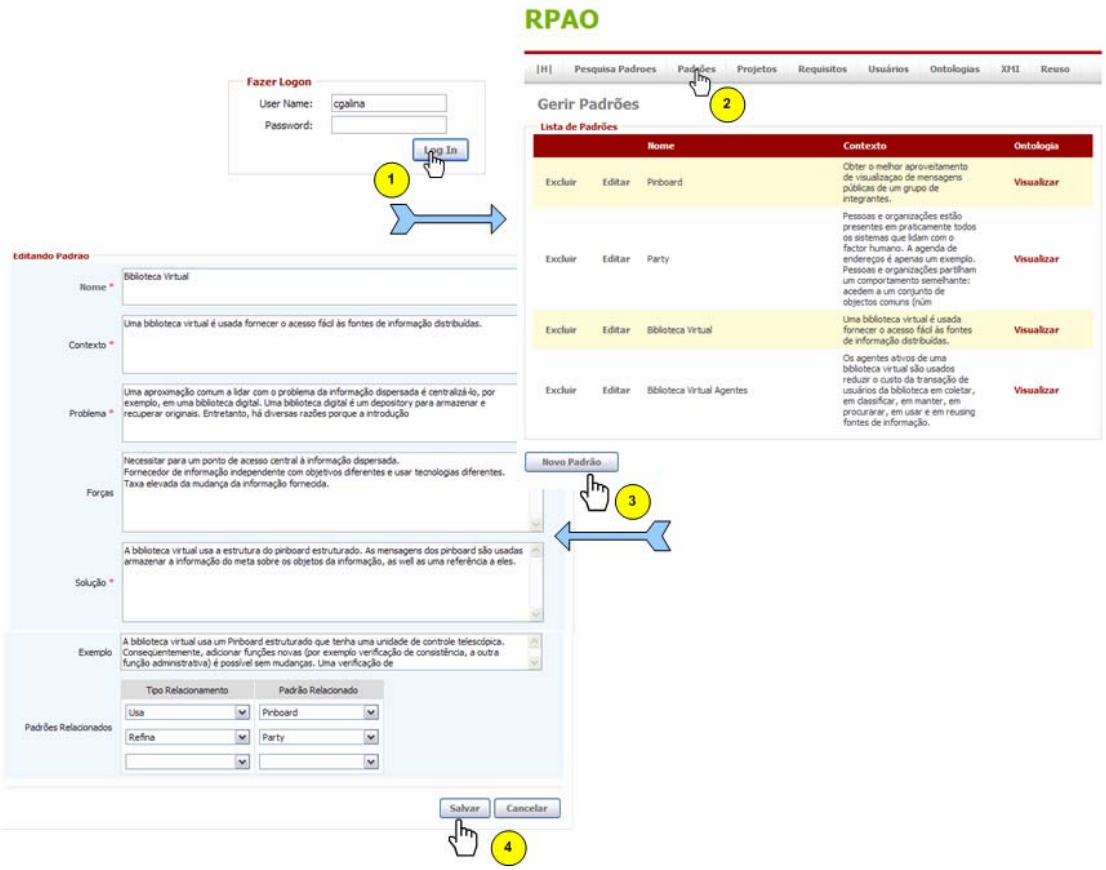

Figura 59: Criação de um padrão de análise

O cadastro de padrões em uma base de conhecimento fornece aos engenheiros de *software* agilidade, pois através deles é possível encontrar, em pouco tempo, a solução para uma análise. Também facilita ao engenheiro de *software* a documentação de novos padrões e

a alteração de um padrão existente para atender uma nova necessidade. Porém, o principal benefício está relacionado à reutilização dos padrões catalogados e que outros projetos estarão usufruindo da definição completa da solução.

Após a criação dos padrões para agrupá-los em projetos a ferramenta disponibiliza a opção de gerenciamento de projetos (1), no qual foram inclusos quatro projetos que a empresa está trabalhando e que serão avaliados no CMMI3 entre eles: *Worten, Coninente, Exit* e *EAI*. Na Figura 58 é representada a criação do projeto *Worten* relacionando-o com os padrões de análise já criados (2). O relacionamento entre padrões e projetos estabelece uma ligação dos padrões implementados dentro dos projetos, desta forma, auxilia a avaliação da utilização dos padrões e o retorno que estão fornecendo para os projetos.

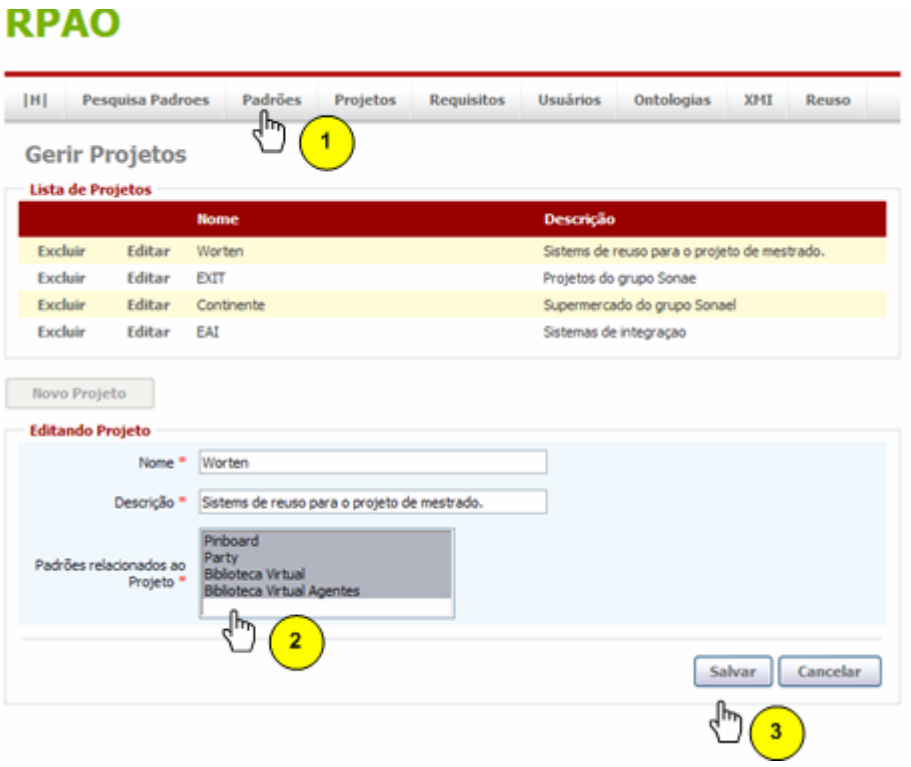

Figura 60: Criação de um padrão de análise

A empresa já disponibilizava alguns casos de uso modelados para realizar testes e disponibilizou o arquivo XMI para que eles pudessem ser utilizados no teste da aplicação e na importação do arquivo. O processo de importação de arquivos XMI traz benefícios quanto ao gerenciamento de casos de uso já existentes. Através desta funcionalidade é possível importar estes casos de uso para a ferramenta. desta forma, é realizada a importação de todos os casos

de uso, assim unificando o mesmo nome do caso de uso dentro do RPAO bem como a sua descrição. Foi realizada a importação de um arquivo XMI denominado “UML.xmi”, o processo de importação é apresentado na seção 5.5.1. Os casos de uso apresentados na listagem foram transferidos para o cadastro de requisitos reusando a nomenclatura importada no arquivo XMI e, também, a descrição quando existir. Através deste processo os dados importados agora passam a ser requisitos documentados e relacionados a padrões de análise.

A ferramenta permite aos engenheiros de *software* gerir requisitos para que eles possam, futuramente, obter dados concretos e avaliar a origem da descoberta do padrão documentado. A Figura 61 apresenta a relação entre o requisito *gerir agenda* e o padrão *Party,* identificando a influência que o requisito teve na descoberta do padrão. Além da documentação do requisito através de campos do *template*, a ferramenta fornece a inclusão de diagramas UML como: diagrama de atividades (5), casos de uso (6), classes (7) e seqüências (8). A integração destes diagramas com a ferramenta dá-se através de *upload* de imagens. Em virtude disso, os diagramas precisam ser exportados para imagens.

A empresa Tlantic trabalha com documentação de descrição de casos de uso, logo, o caso de uso gerir agenda de clientes é um caso de uso descritivo, chamado de EFR (Especificação Funcional de Requisitos). Devido a possibilidade que a ferramenta oferece de armazenar qualquer outro artefato descritivo (9), também foi possível armazenar no repositório o documento EFR utilizado pela empresa.

Todo este processo de unificação de informações relevantes sobre requisitos, projetos, padrões de análise e artefatos, trouxeram para a Tlantic boas práticas. Já que, anteriormente, os dados referentes aos requisitos eram mantidos entre os engenheiros envolvidos somente durante o ciclo de vida do projeto. Após a conclusão do projeto, todas as boas práticas realizadas eram esquecidas em papéis documentados sem o devido reuso das informações.

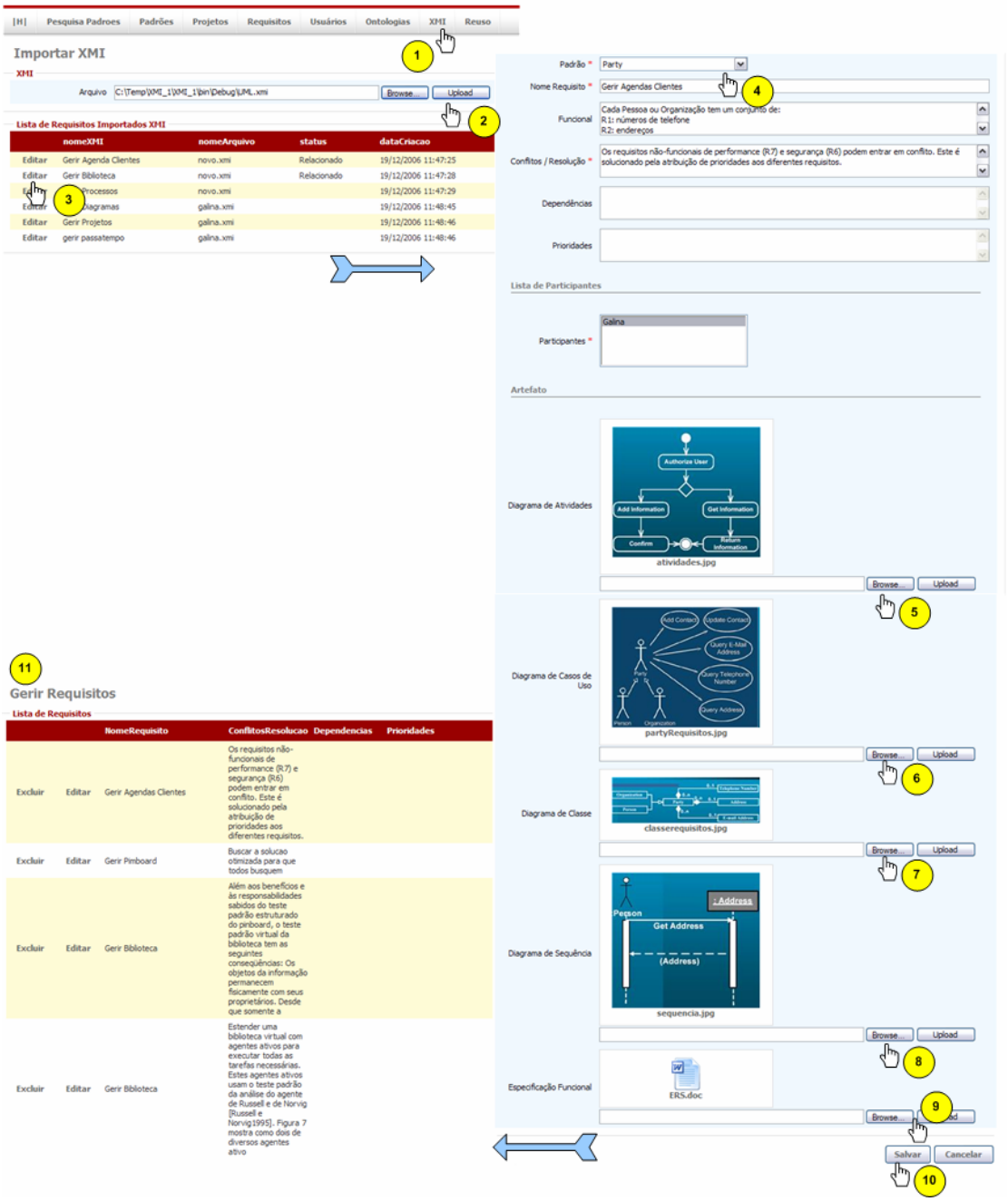

Figura 61: Inclusão de Requisitos através de Importação XMI.

O processo de pesquisa já apresentado na seção 5.3.1 e 5.4, é um mecanismo de busca na base de conhecimento, e retorna para os envolvidos a consulta imediata conforme a palavra chave desejada, como apresentado na Figura 62. É possível aplicar os filtros desejados para procurar os padrões na base de conhecimento, no exemplo (1) foi realizado o filtro para apresentar todos os padrões do projeto *Worten* (2). Na opção de detalhar (3) é possível visualizar, de maneira rápida, os detalhes do padrão e todos os requisitos relacionados a ele.

Analisando o padrão *Party* é possível verificar que ele possui um requisito relacionado Gerir Agenda do Cliente.

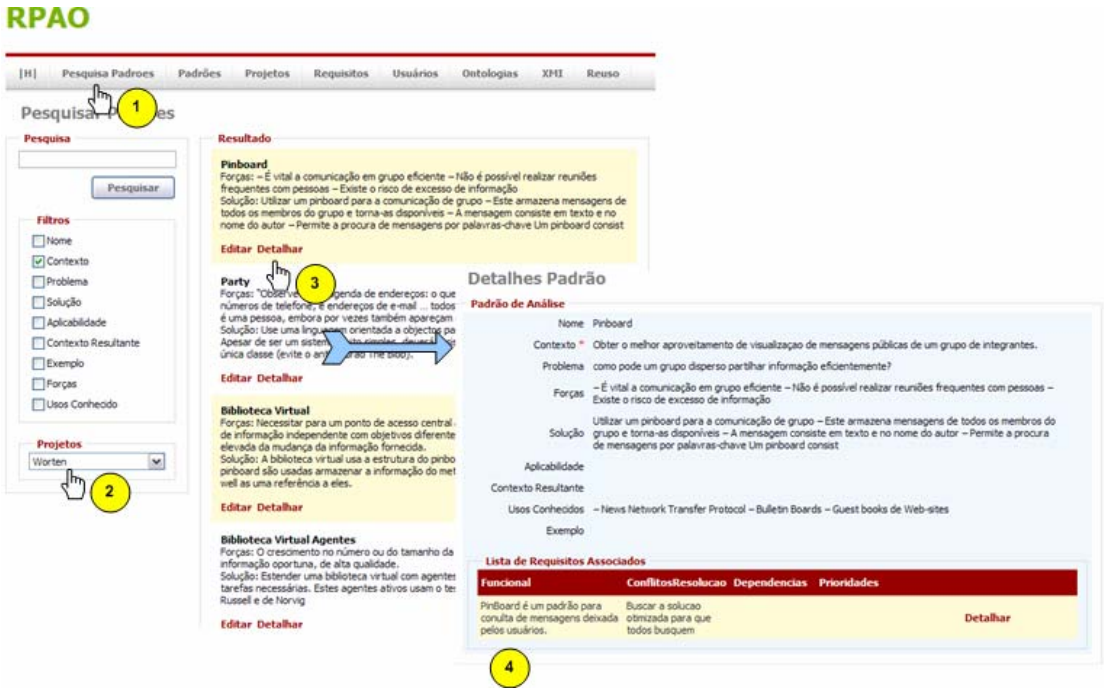

Figura 62: Pesquisa de Padrões de Análise e Detalhe

Ao detalhar um padrão de análise é possível saber todos os requisitos, artefatos e projetos relacionados a este padrão, possibilitando assim uma visão única de todos os dados correlacionados ao padrão.

Os padrões de análise inseridos automaticamente no repositório geraram as ontologias para a utilização dos mesmos em outros sistemas, assim o padrão *Party* utilizado na Tlantic foi publicado para que outras empresas possam avaliar como o padrão foi utilizado, como pode ser visualizado na Figura 63.

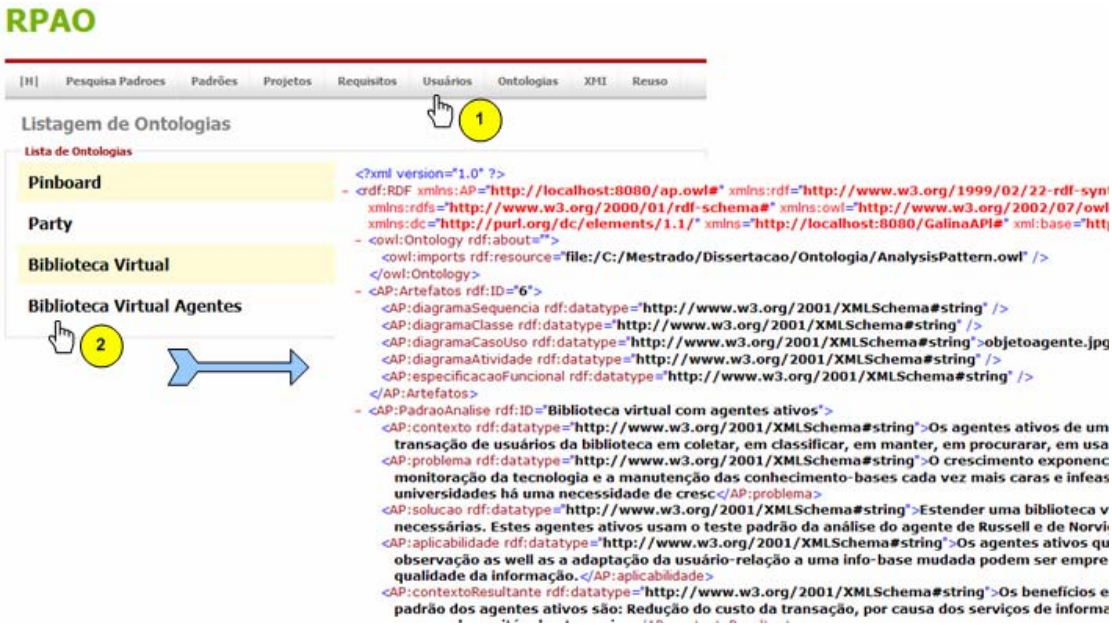

Figura 63: Listagem de Ontologias publicadas

A ferramenta ainda disponibiliza a função de visualização de padrões de análise utilizados dentro dos projetos, sendo possível assim saber o reuso dos padrões nos projetos. Este reuso pode ser observado através de um gráfico que apresenta os resultados já apresentados na Figura 46 da seção 5.4. Esta figura apresenta o gráfico de apresentação dos resultados. A partir do gráfico é possível perceber, por exemplo, que o padrão *Party* é encontrado em 3 projetos.

Após a criação da ontologia *Party* realizada pela ferramenta RPAO, é possível a captura da ontologia para o reuso através do *Web Service.* A Figura 64 demonstra a utilização do *Web Service* acessado através de uma URL do servidor de aplicação que apresenta a tela de requisição do método de busca de ontologia (1) e após a entrada da Ontologia que deseja-se buscar. A ontologia anteriormente gerada foi usada como exemplo para que o cliente possa capturá-la através do *Web Service.*

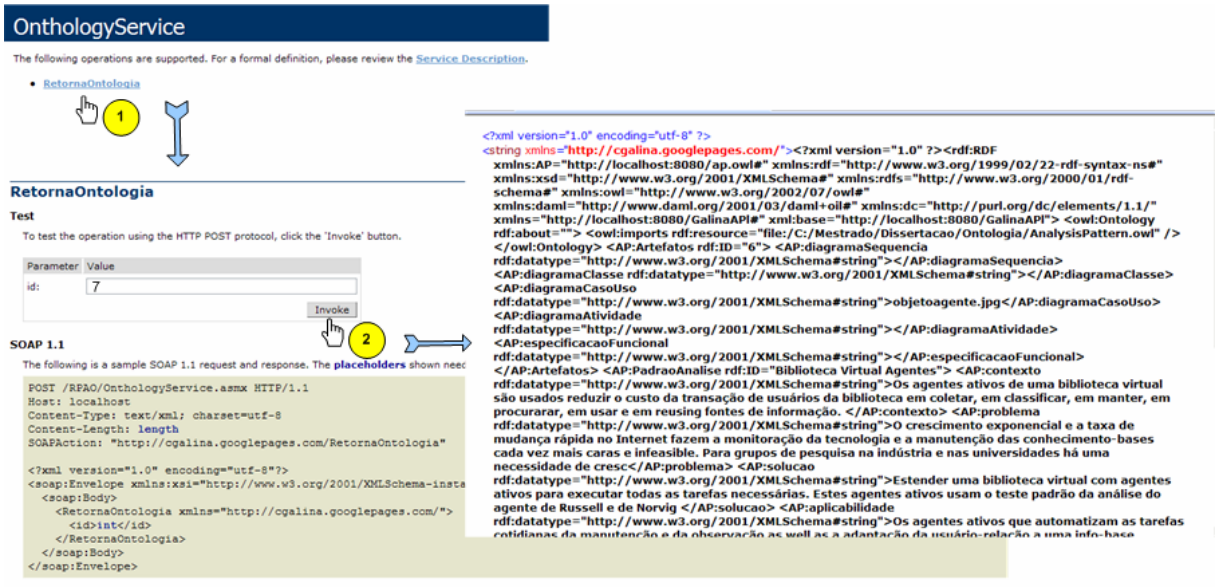

Figura 64: Transporte da ontologia através do *Web Service.*

Uma aplicação cliente foi construída para realizar a busca da ontologia simulando, assim, um sistema externo que a requisita. Na ferramenta foi inserido o ID da ontologia *Party,* que processa a requisição para o *Web Service*, onde verifica se a ontologia requisitada existe e retorna para o cliente a resposta composta pelo arquivo da ontologia, como pode ser visualizado na Figura 65.

Figura 65: Exemplo de Cliente acessando o *Web Service.*

6. CONCLUSÃO

A realização deste trabalho proporcionou uma visão mais abrangente sobre ontologias e sobre o significado de se obter padrões para qualquer fase de um projeto. Os padrões de análise apesar de terem um pequeno espaço na comunidade de pesquisa, possuem uma importância muito significativa.

As empresas hoje falham em reescrever processos e documentos por não obter um repositório de padrões e requisitos mantido por uma equipe focada a buscar o reuso de engenharia de *software.* A não consolidação desta equipe ocasiona a má utilização de padrões de análise que poderiam concentrar os seus esforços na documentação dos padrões para futuramente ser reusados por outros engenheiros.

Hoje, muitas empresas utilizam metodologia RUP e seguem as fases apresentadas pelo processo. Muitas dessas empresas geram artefatos em papel que, normalmente, são esquecidos durante o projeto e durante o desenvolvimento de novos projetos. A solução de armazená-los em um repositório que seja de fácil acesso e manipulação agrega dinâmica na percepção de utilização dos padrões.

As ontologias, para este trabalho, tiveram duplo sentido. Primeiro, por uma ontologia representar um domínio, através da criação da ontologia que os requisitos foram mapeados para a criação da ferramenta. E segundo, para que os projetos pudessem identificar um padrão percebeu-se a necessidade de algo que conduzisse esse processo, para isso foi criada uma ontologia mestre, fornecendo assim uma estrutura da ontologia dos padrões de análise para sistemas externos.

O repositório ainda está sendo testado na empresa Tlantic como já mencionado no estudo de caso, porém já puderam ser avaliados alguns benefícios quanto à utilização e à unificação de dados envolvendo requisitos e padrões de análise. A Tlantic está criando a prática de reusar suas informações e conteúdos criados nos projetos. A publicação dos padrões de análise fornece aos engenheiros uma coleção de dados que, anteriormente, eram acessados através de documentos do tipo *word* através de uma busca manual de informações, agregando custo e tempo nesta atividade. Desta forma, o *template* para a inclusão dos padrões

fornece campos que documentem um padrão de forma legível e entendível por engenheiros de *software*.

6.1. Trabalhos Futuros

Algumas *features* não puderam ser implementadas neste projeto entre elas são:

- A geração de indicadores de reuso de padrões de análise por projeto também é identificado como uma boa característica para o auxílio do gerenciamento da ferramenta;

- Criação de *templates* dinâmicos para que a ferramenta os forneça de forma customizada aos engenheiros e, ao mesmo tempo, gere a estrutura do *template* criado;

- Transformar a ferramenta RPAO em genérica para que possa atender a qualquer tipo de padrão existente, desde padrões de projetos a padrões de requisitos. Hoje, a RPAO está preparada para esta evolução já que a classe "*PadraoAnalise"* herda características de uma classe "*Padrao"*, totalmente desacoplada.

REFERÊNCIAS BIBLIOGRÁFICAS

[ALEX 79] ALEXANDER, C. **The Timeless Way of Building**. Oxford University Press, New York, NY, 1979.

[ALEX 77] ALEXANDER, C. et al. **A Pattern Language: Towns, Buildings, Construction**. Oxford University Press, New York, NY, 1977.

[ANKO 04] ANKORI, Ronit. **Automatic Requirements Elicitation in Agile Process**. In: IEEE International Conference on Software – Science Technology & Engineering (SwSTE'05), 2005.

[ANDR 01] 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.

[ANDR 03] 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

[ANDE 03] 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.

[BECH 06] BECHHOFER, S; HARMELEN, F. V; HENDLER, J; HORROCKS, I; MCGUINNESS,D. L; PATEL-SCHNEIDER, P. F; STEIN, L. A. OWL Web Ontology Language Reference.

http://www.w3.org/TR/2004/REC-owl-ref-20040210/, acessado em Abril 2006.

[BENJ 99] BENJAMINS, R. et al. Wondertools. **A comparative study of ontological engineering tools**. In: INTERNATIONAL WORKSHOP ON KNOWLEDGE ACQUISITION, MODELING, AND MANGEMENT, 12, Banff, Canada, 1999. , Banff, Canada. Proceedings... Banff, Canada : [s. n.], 1999.

[BORL 05] Borland Together. Disponível em: http://www.borland.com.br/togehter

[BYSC 96] BYSCHMANN, F.; MEUNIER, R; ROHNER, H.; SOMMERLAD, P. and STAL, M. "**Pattern-Oriented Software Architecture**", J. Wiley and sons, New York, NY, 1996

[BUDI 96] BUDINSKY, F. at al. "**Automatic Code Generation From Design Patterns**". IBM Research Journal, Vol. 35, No.2, 1996.

[CHAN 99] CHANDRASEKARAN, B.; JOSEPHSON, Jhon, R. **What are Ontologies, and why do you need them?** In IEEE Intelligent Systems. 1999.

[COAD 92] COAD, Peter. **Object-oriented patterns.** Communications of the ACM, September 19992.

[CODE 06] CodePro Studio. Disponível em http://www.instantiations.com/codepro/ws/docs/default.htm

[CONT 02] CONTE, A. at al. **A tool and a formalism to design and appy patterns.** SugarLoafPLoP, 2002.

[COPL 96] COPLIEN, J. O. **Software Patterns**. SIGS books and Multimedia, June 1996.

[COPL 00] 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.

[DEVE 02] DEVEDZIC, Vladan. **Software Patterns.** In: Chang, S.K. (ed), "Handbook of Software Engineering and Knowledge Engineering Vol.2 – Emerging Technologies", World Scientific Publishing Co., Singapore, 2002, pp 645-671

[DECK 99] DECKER, S. et al. **The semantic web: the roles of XML and RDF**. IEEE Expert, v. 15, n. 3, 2000. 23, PATTERSON, D.; NYDER, L.; ULLMAN, J. Best practices memo: evaluating computer scientists and engineers for promotion and tenure. Computing Research News, p. A-B, Sept. 1999. Special insert. 24, LAWRENCE, Steve. Online or invisible Nature, v. 411, n. 6837, p. 521, 2001

[ERIK 00] ERIKSSON,H.E.; PENKER, M. **Business modeling with UML: Business Patterns at work.** New York, NY: John Wiley & Sons, 2000.

[FARI 03] FARIA, Carla; GIRARDI, Rosario. **Especificação de uma Ontologia Genérica para Análise de Requisitos da Engenharia de aplicações Multiagente**. Terceiro Congresso Brasileiro de Computação (CBComp 2003), UNIVALI, Itajaí, SC, Brasil, Agosto de 2003.

[FERR 03] Ferreira. S. L. C.; Girardi, R. "Arquiteturas de Software Baseadas em Agentes:

Do Nível Global ao Detalhado.” In: Revista Eletrônica de Iniciação Científica, v. 2, n. 2, Junho, 2002.

[FOWL 97] FOWLER, M. **Implementing Analysis Patterns**. Presentation at OOP'97, 1997.

[GAMM 95] GAMMA, E.; HELM, R.; JOHNSON, R. and VLISSIDES, J. “**Design Patterns: Elements of Reusable Object-Oriented Software**”, Addison-Wesley, MA, 1995.

[GERB 99] 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.

[GRUB 93] GRUBER, Thomas R. A Translation Approach to Portable Ontology Specifications. In: Knowledge Acquisition, 1993.

[HAMZ 02] HAMZA. H. and FAYAD, M.E. (2002) “Model-based Software Reuse Using Stable Analysis Patterns” ECOOP 2002, Workshop on Model-based Software Reuse, June 2002, Malaga, Spain.

[HASL 02] Hahsler, M. ***Software Patterns: Pinwände***. Masters Thesis, Vienna University of Business Administration and Economics, November 2002.

[HANS 02] Hansen, R. P.; Santos, C. T.; Crespo, S; Lanius, G. L.; Massen, F. “Web Services: An Architectural Overview.” In: Proceedings of the First Seminar on Advanced Research in Electronic Business, Rio de Janeiro, 2002.

[HANS 03] Hansen, R. P. “GlueScript: Uma Linguagem Específica de Domínio para Composição de Web Services.” Dissertação de Mestrado, São Leopoldo: Centro de Ciências Exatas e Tecnológicas da UNISINOS, 2003.

[HIDR 02] MANGAN, M.A.S. et al. **Modelos de Domínio e Ontologias: uma comparação através de um estudo de caso prático em hidrologia.** COPPE/UFRJ – Programa de Engenharia de Sistemas e Computação. Rio de janeiro, 2002.

[JACO 02] JACOBSON, I; BOOCH, G; RUMBAUGH, J. **The Unified Software Development Process.** Reading, MA: Addison-Wesley, 2002.

[LAMS 00] LAMSWEERDE, Axel Van. **Requirements Engineering in the Year 00: A Research Perspective.** In: IEEE 22th International Conference on Software Engineering, 2000.

[METS 04] METSKER, Steven John, “Padrões de Projeto em Java”, Reading: Bookman, 2004

[MYLO 92] MYLOPOULOS, John; CHUNG, Lawrence; NIXON, Brian. **Representing and Using Non-Functional Requirements: A Process-Oriented Approach.** In: IEEE Transactions on Software Engineering, 1992.

[NATA 04] NOY, Natalya F.; MCCUINNESS, Deborah L. **Ontology Development 101: A guide to creating your first ontology**. Stanford University. Stanford, CA. 2004

[NATA 06] NATALYA F., MCGUINNESS, Deborah L. Ontology Development 101: **A Guide to Creating Your First Ontology**. Stanford University, Stanford, 2001. Disponível em: http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noymcguinness.html. Último acesso: 05 de abril de 2006

[NEWC 02] Newcomer, E. “Understanding Web Services: XML, SOAP, UDDI, and WSDL.” Boston: Addison-Wesley, 2002.

[NUSE 00] NUSEIBECH, Bashar; EASTERBROOK, Steve. **Requirements Engineering: A Roadmap**. In ACM Future of Software Engineering, 2000.

[NOVE 05] NOVELLO, Taisa Carla. **Ontologias, Sistemas baseados em conhecimento e modelo de Banco de Dados**. Disponível em www.inf.ufrgs.br/~clesio/ cmp151/cmp15120021/artigo_taisa.pdf. Último acesso em fevereiro de 2005.

[OELL 01] Oellermann Junior, W. L. “Architecting Web Services.” New York: Apress, 2001.

[PACH 01] PACHECO, Roberto C. dos Santos; KERN, Vinícius Medina. **Uma ontologia comum para a integração de bases de informações e conhecimento sobre ciência e tecnologia**. Ci Inf., Brasília, v. 30, n.3, p. 56-63, set./dez. 2001.

[PREE 95] PREE, W. **Design Patterns for Object-Oriented Software Development**. Addison - Wesley Publishing Company, ACM Press, 1995.

[PRES 02] PRESMAN, Roger S. **Engenharia de Software**. Rio de Janeiro: McGraw-Hill, 2002.

[PROT 06] PROTÉGÉ 3.1.1. Disponível em: http://protege.stanford.edu. Acesso em: 20/05/2006

[RATI 05] RATIONAL UNIFIED PROCESS Tutorial. Versão 2005 05 00.

[RATI 06] The Ten Essentials of RUP – The Essence of an Effective Development Process Leslee Probasco. A Rational Software White Paper www.rational.com

[RHEI 04] RHEINHEIMER, Letícia. **WSAgent: Um agente baseado em *Web service*s para promover interoperabilidade entre sistemas heterogêneos no domínio da saúde.** Dissertação de Mestrado do Programa de Pós Graduação em Computação Aplicada – UNISINOS, São Leopoldo. 2004.

[SOMM 04] SOMMERVILLE, Ian. **Engenharia de *Software.*** Reading Addison Wesley. 2004.

[STAA 00] STAAB, S.; MAEDCHE, A. **Ontology engineering beyond the modeling of concepts and relations**. In: ECAI'2000 WORKSHOP ON APPLICATION OF ONTOLOGIES AND PROBLEM-SOLVING METHODS, 2000, Amsterdam. Proceedings ... [S. l.] : IOS ress, 2000.

[SECU 07] SecurityPatterns.org. Disponível em: http://www.securitypatterns.org. Acessado em:
11 JAN, 2007.

[VLIS 97] VLISSIDES, J. ***Patterns: The Top Ten Misconceptions***. Object Magazine, Mar, 1997. Disponível em: http://hillside.net/patterns/papersbibliographys.htm. Acesso em: 06/04/2006.

[WERN 00] 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.

[WIEG 03] WIEGERS, Karl. **Software Requirements**. Canada: H. B. Fenn and Company Ltda, 2003.

[WORL 04] World Wide Web Consortium Web Services Architecture Working Group. "Web Services Architecture Requirements: W3C Working Draft." August, 2002. Disponível em http://www.w3.org/TR/2002/WD-wsa-reqs-20020819, último acesso em 14/05/2004.

[ZIMM 95] ZIMMER, W. ***Relationships Between Design Patterns***. Pattern Languages of Program Design, Addison-Wesley, 1995.

ANEXO 1 – DESCRIÇÃO DE CASO DE USO MANTER PADRÕES DE ANÁLISE

Caso de Uso: Manter Padrões de Análise		
Descrição: Este caso de uso é responsável pela manutenção de padrões de análise sob a utilização de campos de um *template* que o possibilite documentar da forma mais eficiente para reuso.		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema		
Pós-condições: Não de Aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Após o engenheiro de software ter se logado no sistema é apresentado um menu de opções que o engenheiro pode buscar. Engenheiro clica na opção Manter Padrões.	Sistema apresenta a tela de inclusão do padrão de análise com o formulário e campos específicos do *template*.
2	Na tela de manter padrões de análise o engenheiro possui duas opções para seqüência: Incluir; Cancelar.	Se usuário clicar em Incluir, o sistema apresenta o caso de uso **Incluir Padrões de análise.** Se o usuário clicar em Cancelar o sistema retorna para tela inicial do sistema.
3	Encerra seqüências básicas do caso de uso.	Encerra operações.

Caso de Uso: Incluir Padrões de Análise
Descrição: Este caso de uso é um refinamento de Manter Padrões de análise que destina-se unicamente ao processo de gravação dos dados postados pelo ATOR. Protótipo: Figura 31
Atores: Engenheiro de sistema.

Pré-condições: Logar no sistema, Manter padrões de análise.		
Pós-condições: Manter padrões de análise.		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator através do caso de uso Manter padrões de análise realiza a operação de inclusão.	Sistema apresenta a tela de inclusão de padrões de análise.
2	Ator preenche os dados do *template*.	
3	Ator escolhe outros padrões o qual o padrão atual se relaciona.	Sistema processa o caso de uso **Criar Relacionamentos.**
4	Ator pode escolher as seguintes opções: Salvar Padrão de Análise Cancelar Gerir Projetos Gerir Requisitos	Se o ator clicou em salvar, o sistema realiza a persistência em base de dados e resulta no caso de uso **Gerar Ontologia.** Se o ator clicou em cancelar, é finalizado o caso de uso atual. Se o ator clicou em Gerir projetos é apresentado o caso de uso **Gerir Projetos.** Se o ator clicou em Gerir Requisitos, é apresentado o caso de uso **Gerir Requisitos.**

Caso de Uso: Excluir Padrões de Análise		
Descrição: Este caso de uso é um refinamento de Manter padrões de análise que destina-se unicamente ao processo de exclusão de padrões já relacionados. Protótipo: Figura 31		
Atores: Engenheiro de sistema.		
Pré-condições: Logar no sistema, Manter padrões de análise, Listar padrões de análise.		
Pós-condições: Manter padrões de análise.		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Se ator estiver relacionado ao caso de uso Pesquisar padrões de análise então seguir o passo **2**. Senão, se ator estiver relacionado ao caso de uso Detalhar Padrão de análise, seguir o passo **3**.	

2	Com os resultados apresentados do caso de uso Pesquisar padrões de análise o ator escolhe um dos itens apresentados na listagem e clica em excluir padrão de análise	Sistema processa a requisição do ator e verifica os relacionamentos entre Requisitos e Projetos, se existir *constraints* irá apresentar uma mensagem ao ator referente aos relacionamentos que utilizam o padrão.
3	Ator dentro da tela mestre de detalhes do padrão de análise clica em excluir padrão de análise	Sistema processa a requisição do ator e verifica os relacionamentos entre Requisitos e Projetos, se existir *constraints* irá apresentar uma mensagem ao ator referente aos relacionamentos que utilizam o padrão.
4	Ator finaliza caso de uso.	

Caso de Uso: Alterar Padrões de Análise		
Descrição: Este caso de uso é um refinamento de Manter Padrões de análise que destina-se unicamente ao processo de alteração de padrões já cadastrados. Protótipo: Figura 31		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Manter padrões de análise, Pesquisar Padrões de Análise.		
Pós-condições: Manter padrões de análise		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator através dos resultados do caso de uso Pesquisar padrões de análise escolhe a opção Alterar Padrões de análise.	Sistema processa a requisição e apresenta a tela de Alterar padrões de análise com os campos que são necessários para alteração de textos.
2	Ator realiza as alterações necessárias.	Sistema processa as alterações, busca a ontologia já criada do padrão e altera a ontologia específica do padrão. Sistema gera um log de alterações.
3	Ator finaliza caso de uso	

Caso de Uso: Detalhar Padrões de Análise
Descrição: Este caso de uso é um refinamento de Manter Padrões de análise que destina-se unicamente ao processo apresentação dos padrões catalogados já cadastrados. Protótipo: Figura 31
Atores: Engenheiro de sistema

Pré-condições: Logar no sistema, Manter padrões de análise, Pesquisar Padrões de Análise.		
Pós-condições: Manter padrões de análise		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator através dos resultados do caso de uso Pesquisar padrões de análise escolhe a opção Detalhar Padrões de análise.	Sistema processa a requisição e apresenta a tela de detalhes de padrões de análise, com todos os campos da tela somente leitura sem a possibilidade de alteração.
2		Sistema apresenta também os projetos que o padrão está relacionado.
3		Sistema apresenta os Requisitos envolvidos ao padrão e artefatos que estão vinculados ao requisito.
4	Ator finaliza caso de uso	

Caso de Uso: Criar Relacionamentos		
Descrição: Este caso de uso refere-se aos padrões de relacionamento para si próprio, para cada padrão que deseja cadastrar é possível referenciar outro padrão já cadastrado, assim existirão padrões aos quais tem-se a possibilidade de se relacionar. Os tipos de relacionamentos permitido no sistema são: REFINA, USA, REQUER e ALTERNATIVO. Por exemplo, o Padrão contrato pode refinar padrão Perfil. O sistema possibilitará a inclusão do tipo de padrão através de um domínio pré-definido. Protótipo: Figura 31		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Manter padrões de análise.		
Pós-condições: Manter padrões de análise		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ao escolher o campo “Padrões relacionados” do *template*, o ator clica na lista para associar padrões.	Sistema apresenta uma listagem de padrões que o ator pode escolher.
2	Ator escolhe o tipo de relacionamento que o padrão atual (em registro) tem sob o padrão “B” (escolhido).	Sistema apresenta as opções de relacionamento (REFINA, USA, REQUER e ALTERNATIVO).
3	Ator escolhe a opção de tipo de relacionamento.	Sistema persiste os relacionamentos desejados.
4	Ator finaliza caso de uso.	

Caso de Uso: Pesquisar Padrões de Análise		
Descrição: Este caso de uso é um refinamento de Manter Padrões de análise que destina-se a pesquisa avançada dos padrões de análise da base de conhecimento. Os dados cadastrados podem ser pesquisados por qualquer campo do *template*, assim como os projetos relacionados ao padrão e requisitos. Protótipo: Figura 35		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Montar Menu sistema.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator escolhe no menu do sistema a opção de pesquisa avançada.	Sistema apresenta a tela com campos de pesquisa avançada como apresenta o protótipo já mencionado.
2	Ator seleciona os campos para refinar sua pesquisa e encontrar os padrões catalogados	
3	Ator realiza a pesquisa com os campos desejados	Sistema processa a requisição apresentando os dados que foram encontrados com a pesquisa refinada.
4	Ator finaliza caso de uso	

ANEXO 2 – DESCRIÇAO DE CASOS DE USO GERIR PROJETOS

Caso de Uso: Gerir Projetos		
Descrição: Podem ser identificados projetos e então armazená-los em base para que possa distribuir os padrões encontrados dentro dos projetos. Protótipo: Figura 34		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator escolhe no menu do sistema a opção gerir projetos	Sistema apresenta a tela de gestão de projetos
2	Ator Deseja realizar a inclusão de um novo projeto	Sistema retorna para o ator o caso de uso Incluir Projetos.
3	Ator finaliza caso de uso.	

Caso de Uso: Incluir Projetos		
Descrição: Este caso de uso é um refinamento de gerir projetos que destina-se a inclusão de novos projetos que podem ser catalogados e associados aos padrões. Protótipo: Figura 37		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator sob refinamento do caso de uso gerir projetos, preenche o formulário dos campos para criação de um novo projeto.	Sistema apresenta os campos para preenchimento.
2	Ator deseja relacionar padrões já	Sistema executa o caso de uso **Associar Padrões de**

	existentes para o presente projeto.	**análise**.
3	Ator salva os dados preenchidos.	Sistema persiste os dados.
4	Ator finaliza caso de uso	

Caso de Uso: Excluir projeto		
Descrição: Este caso de uso é um refinamento de gerir projetos que destina-se a exclusão projetos. Protótipo: Figura 34		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Montar Menu sistema, Gerir Projetos, Pesquisar Projetos.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator realiza operação através do caso de uso Pesquisar projetos.	
2	Com os resultados apresentados do caso de uso Pesquisar padrões de análise o ator escolhe um dos itens apresentados na listagem e clica em excluir projetos.	Sistema processa a requisição do ator e verifica os relacionamentos entre Requisitos e Projetos, se existir *constraints* irá apresentar uma mensagem ao ator referente aos relacionamentos que utilizam o padrão.
3	Ator finaliza caso de uso.	

Caso de Uso: Alterar Projeto		
Descrição: Este caso de uso é um refinamento de gerir projetos que destina-se a alteração de projetos. Protótipo: Figura 34		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Montar Menu sistema, Gerir Projetos, Pesquisar Projetos.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator realiza operação através do caso de uso Pesquisar projetos.	

2	Com os resultados apresentados do caso de uso Pesquisar padrões de análise, o ator escolhe um dos itens apresentados na listagem e clica em alterar projeto.	Sistema apresenta a tela de alteração de projetos para que o ator alterar os dados postados.
3	Ator clica em salvar.	Sistema processa requisição persistindo os dados.
4	Ator finaliza o caso de uso.	

Caso de Uso: Pesquisar Projetos		
Descrição: Este caso de uso tem por definição a pesquisa simples de projetos catalogados. Protótipo: Figura 34		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Montar Menu sistema.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator escolhe no menu do sistema a opção de pesquisa projetos	Sistema apresenta a tela para pesquisa simples de projetos
2	Ator digita o nome como chave do projeto que deseja pesquisar e clica na operação pesquisar.	Sistema processa a requisição apresentando os dados que foram encontrados com a pesquisa refinada, com as operações de exclusão e alteração.
4	Se ator escolher exclusão.	Caso de uso Excluir Projetos.
5	Se ator escolher alteração.	Caso de uso Alterar Projetos.
6	Ator finaliza caso de uso.	

ANEXO 3 – DESCRIÇÃO DE CASO DE USO GERIR REQUISITOS

Caso de Uso: Gerir Requisitos		
Descrição: Este caso de uso refere-se a gestão de requisitos que fazem parte de um padrão de análise. Protótipo: Figura 36		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Montar Menu sistema, Manter Padrões de Análise.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator através do menu principal ou na gestão de um Padrão de análise pode escolher a operação Gerir Requisitos.	Sistema apresenta a tela do Caso de uso Refinado **Incluir Requisito**
2	Ator finaliza caso de uso.	
5	Se ator escolher alteração.	Caso de uso Alterar Requisitos.
6	Ator finaliza caso de uso.	

Caso de Uso: Incluir Requisitos		
Descrição: Este caso de uso refere-se a inclusão de requisitos que fazem parte de um padrão de análise. Protótipo: Figura 36		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Montar Menu sistema, Manter padrões de análise, Manter Requisitos.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator através do menu principal ou na gestão de um Padrão de análise pode escolher a operação Gerir Requisitos.	Sistema apresenta a tela do Caso de uso Refinado **Incluir Requisito.**

2	Ator escolhe participantes que trabalharam no requisito.	Sistema executa caso de uso **Buscar Participantes**
5	Ator escolhe artefatos para inclusão.	Sistema executa caso de uso Gerir Artefatos
6	Ator realiza a operação salvar e finaliza caso de uso.	Sistema persiste os dados e retorna para caso de uso anterior.

Caso de Uso: Alterar Requisitos		
Descrição: Este caso de uso refere-se a alteração de requisitos que fazem parte de um padrão de análise. Protótipo: Figura 36		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Montar Menu sistema, Pesquisar Requisitos.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator realiza operação através do caso de uso Pesquisar requisitos.	
2	Na listagem de requisitos, o ator escolhe a opção detalhar para alteração do registro.	Sistema apresenta a tela de alteração de requisitos.
5	Ator clica em salvar.	Sistema processa requisição persistindo os dados.
6	Ator finaliza o caso de uso.	

Caso de Uso: Excluir Requisitos		
Descrição: Este caso de uso refere-se a exclusão de requisitos que fazem parte de um padrão de análise. Protótipo: Figura 36		
Atores: Engenheiro de sistema		
Pré-condições: Logar no sistema, Montar Menu sistema, Pesquisar Requisitos.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA

1	Ator realiza operação através do caso de uso Pesquisar requisitos.	
2	Com os resultados apresentados do caso de uso Pesquisar requisitos, o ator escolhe um dos itens apresentados na listagem e clica em excluir requisitos.	Sistema processa a requisição do ator e verifica os relacionamentos entre Requisitos, Padrões de análise e Participantes, se existir *constraints* irá apresentar uma mensagem ao ator referente aos relacionamentos que utilizam o padrão.
5	Ator confirma exclusão.	Sistema processa a exclusão em base de dados e ontologia.
6	Ator finaliza caso de uso	

ANEXO 4 – DESCRIÇÃO DE CASOS DE USO GERAR ONTOLOGIAS

Caso de Uso: Gerar Ontologia		
Descrição: Este caso de uso tem a finalidade de gerar as informações dos padrões de análise em ontologias, para a interoperabilidade de dados com outras aplicações, e comunidades de engenheiros de software. Protótipo: Figura 38		
Atores: RPAO		
Pré-condições: Logar no sistema, Montar Menu sistema, Manter padrões de análise.		
Pós-condições: não se aplica		
SEQÜÊNCIA PRINCIPAL		
AÇÃO DO ATOR		AÇÃO DO SISTEMA
1	Ator de sistema após a persistência de dados do caso de uso manter padrões de análise, gera a ontologia.	Sistema processa requisição dos dados persistidos e busca a estrutura através do caso de uso Gerir Estrutura da ontologia.
2		Sistema processa os campos dos padrões e formatos persistidos e cria *parser* de conversão para o formato de ontologia.
3		Sistema após finalizar o processo de geração de padrões em ontologia realiza a publicação da ontologia no servidor de aplicação.
4		Sistema gera a ontologia com o mesmo nome do padrão e persiste o nome da ontologia e local de destino em base de dados.
5		Sistema finaliza publicação e geração dos dados.

Caso de Uso: Gerir Estrutura de Ontologia
Descrição: Caso de uso destinado a criar a estrutura de ontologia referente a todo o sistema RPAO.
Ator do sistema destina-se ao fato de modelar a estrutura do sistema já descrito em todos os casos de uso expressa através de modelo de domínio de ontologia. A geração da ontologia com padrões de análise depende da estrutura de classes, slots e propriedades desenvolvidas pelo engenheiro de sistema.