Apoio de Gerência de Conhecimento à Engenharia de Requisitos em um Ambiente de Desenvolvimento de Software

Julio Cesar Nardi

Dissertação de Mestrado em Informática

Mestrado em Informática

Universidade Federal do Espírito Santo

Vitória, junho de 2006

Apoio de Gerência de Conhecimento à Engenharia de Requisitos em um Ambiente de Desenvolvimento de Software

Julio Cesar Nardi

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 14/06/2006 por:

_____
Prof. Dr. Ricardo de Almeida Falbo, D. Sc.
Universidade Federal do Espírito Santo (UFES)
Orientador

_____
Prof. Dr. Davidson Cury, D.Sc.
Universidade Federal do Espírito Santo (UFES)

_____
Renata Silva Souza Guizzardi (PhD)
SRA-IRST-ITC, Trento, Itália

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória, junho de 2006

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

Nardi, Julio Cesar, 1980-

N223a Apoio de gerência de conhecimento à engenharia de requisitos em um ambiente de desenvolvimento de software / Julio Cesar Nardi. – 2006.

145 f. : il.

Orientador: Ricardo de Almeida Falbo.

Dissertação (mestrado) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Software - Desenvolvimento. 2. Ontologia. 3. Gestão do conhecimento. 4. Software – Reutilização. 5. Engenharia de requisitos. I. Falbo, Ricardo de Almeida. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

Dedicatória

Dedico este trabalho aos meus avós Antônio Nardi, Genília Fabrice Nardi, Antonieta Gagno Bergami e Anacleto Bergami (*in memorian*), exemplos eternos.

Agradecimentos

Agradeço a toda minha família, em especial aos meus pais (Paulo e Penha), pelo apoio e pelas orações.

Agradeço à Milena, pela compreensão e pelo carinho.

Agradeço aos meus companheiros de LabES (Laboratório de Engenharia de Software) pelas idéias, discussões, ajudas, rasadas e brincadeiras.

Agradeço ao grande orientador e pessoa Ricardo de Almeida Falbo por tudo. Tenho muito a agradecer e serei eternamente grato.

Agradeço aos meus amigos e colegas pelo apoio.

Agradeço a Deus.

Sumário

Lista de Tabelas

Lista de Ilustrações

Resumo

A busca por um desenvolvimento de software com qualidade tem levado as organizações a aprimorarem seus processos. Neste contexto, uma área muito importante é a Engenharia de Requisitos (ER). Se os produtos das atividades da ER forem gerados com qualidade, isso favorecerá a busca por qualidade nas atividades subseqüentes do processo de desenvolvimento de software.

No entanto, para que os desenvolvedores possam desempenhar o trabalho de engenhar requisitos com produtividade e qualidade, é necessário que eles disponham de ferramentas de apoio. Ademais, é necessário que essas ferramentas estejam integradas entre si e com outras ferramentas de apoio à execução das demais atividades do processo de software. Assim, pode-se buscar tal integração nos Ambientes de Desenvolvimento de Software (ADSs).

A reutilização é outro mecanismo importante neste contexto. No entanto, o reúso só é efetivo se contar, também, com o apoio de ferramentas que o favoreçam, pois, caso contrário, o tempo de buscar, encontrar e adaptar um item pode inviabilizar a reutilização. Para tanto se podem utilizar serviços de Gerência de Conhecimento (GC), sobretudo, integrados a ADSs.

Este trabalho está inserido no contexto do Projeto ODE, um ADS centrado em processo e baseado em ontologias, e tem por objetivo apoiar a execução do processo de Engenharia de Requisitos, fornecendo um ferramental integrado e baseado em ontologias, com serviços de Gerência de Conhecimento para apoiar a reutilização de itens de conhecimento no contexto da ER, incluindo ontologias, padrões de análise e modelos de objetos de projetos similares.

Palavras-chave: Engenharia de Requisitos, Ambientes de Desenvolvimento de Software, Ontologias, Padrões de Análise, Gerência de Conhecimento e Reutilização de Software.

Abstract

The search for quality products in software development is motivating software organizations to improve their processes. In this way, an area that is focused is the Requirements Engineering (RE). If the artefacts generated in the RE´s activities have quality, the other activities that use these artefacts are more capable to generate quality products, either.

However, developers must have tools to help them to do their RE tasks with productivity and quality. Also, it is necessary to integrate these tools with other ones that support other activities of the development process. Thus, Software Engineering Environments (SEEs) are a way to reach this integration.

Reuse is another important issue in this context. However, it can only be efective if there are tools that support it too, because, in another way, the time to find and to adapt an item can be impracticable. Thus, Knowledge Management (KM) services, above all integrated into SEEs, can be used to support RE knowledge items reuse.

This work was developed in the context of ODE (Ontology-based software Development Environment), a process centered SEE that is ontology-based. It aims to support the Requirements Engeneering process, providing an integrated tool (ReqODE) that use Knowledge Management sevices to support knowledge items reuse in the RE context. Those items include ontologies, analysis patterns and object models from similar projects.

Keywords: Requirements Engineering, Software Engineering Environments, Ontologies, Analysis Patterns, Knowledge Management and Software Reuse.

Capítulo 1

Introdução

O objetivo deste capítulo é apresentar ao leitor as bases sobre as quais o trabalho foi construído, os objetivos que foram buscados, a maneira como a pesquisa foi conduzida, considerando alterações no rumo da mesma, além de procurar localizar o leitor quanto à estrutura da dissertação e, de certa forma, motivá-lo para a leitura.

1.1 Motivação

A busca por um desenvolvimento de software que objetive qualidade dos seus produtos tem levado as empresas de software a aprimorarem seus processos. Neste sentido, uma área muito visada é a Engenharia de Requisitos (ER). Requisitos são capturados nas fases iniciais do desenvolvimento de software e, portanto, se forem gerados com qualidade, isso favorecerá a busca por qualidade nas atividades subseqüentes do processo.

De acordo com Kotonya et al. (1998), a Engenharia de Requisitos é um processo que contempla as atividades de levantamento de requisitos, análise e negociação, documentação, verificação e validação e gerência de requisitos. Cada uma dessas atividades traz consigo uma complexidade considerável. A gerência de requisitos, por exemplo, busca controlar as alterações em requisitos ao longo de todo o processo de desenvolvimento de software. Para tanto, saber quais artefatos gerados ao longo do desenvolvimento têm alguma ligação com um requisito é imprescindível para determinar o impacto de mudanças.

Dado que requisitos são tão importantes para o sucesso de um projeto e que a ER envolve um processo bastante complexo, faz-se necessário apoiar os desenvolvedores na execução das atividades do processo de ER por meio de ferramentas automatizadas, principalmente no que diz respeito à gerência de requisitos (WIEGERS, 2003). No entanto, é necessário, ainda, que essas ferramentas estejam integradas a outras ferramentas de apoio ao processo de software, uma vez que requisitos são trabalhados em todas as fases de desenvolvimento. Assim, idealmente, ferramentas de apoio à ER devem ser integradas a Ambientes de Desenvolvimento de Software, sistemas que objetivam fornecer apoio ao processo de software, provendo um conjunto de ferramentas e facilidades integradas que sejam capazes de apoiar todas as atividades do processo, ou pelo menos porções significativas

dele. Para justificar essa necessidade de integração, pode-se retomar o exemplo da atividade de gerência de requisitos.

A gerência de requisitos é uma atividade paralela ao processo de software. Sendo assim, uma ferramenta que apóie tal atividade deve interagir com outras ferramentas que apóiem as outras atividades do processo. Um exemplo disso é o caso de uma ferramenta de apoio à modelagem orientada a objetos. Tal ferramenta produz casos de uso que podem ser relacionados a requisitos funcionais de maneira a documentar que tais requisitos foram modelados em tais casos de uso. Assim, caso um requisito seja excluído ou alterado, deve haver uma comunicação entre as ferramentas de modo que os produtos gerados ao longo do processo se mantenham consistentes e não exista, portanto, um caso de uso que modele uma funcionalidade que não esteja contemplada em um requisito funcional do cliente. Neste contexto, os Ambientes de Desenvolvimento de Software (ADS) se colocam como uma possibilidade de integração entre ferramentas para apoiar de maneira mais consistente todo o processo de software.

Ademais, para que o desenvolvimento de software seja mais efetivo em questões de custos, prazos e qualidade, deve-se buscar a reutilização de itens criados ao longo do processo, tais como, planos de projeto, planos de testes, especificações de requisitos, código fonte, conhecimento etc (Freeman apud CIMA et al., 1997). Com o reúso, o esforço de construção de software tende a diminuir e a qualidade aumentar, pois os itens reutilizados já devem ter passado por verificações e validações em seus projetos de origem (BARROCA et al., 2005). No entanto, o reúso só é efetivo se contar com um suporte que o favoreça, caso contrário o tempo de buscar, encontrar e adaptar um item pode inviabilizar a reutilização.

Dessa maneira, a importância do processo de Engenharia de Requisitos na produção de software de qualidade, a possibilidade de reúso de itens de conhecimento ao longo do processo de desenvolvimento e, em específico, no processo de ER, e a importância de se ter um ferramental integrado que apóie tanto a execução das atividades da ER quanto o reúso no âmbito dessa área da Engenharia de Software constituem o cenário de motivação para este trabalho.

1.2 Contexto e Objetivos

Este trabalho está inserido no contexto do Projeto ODE (*Ontology-based Software Development Environment*) (FALBO et al., 2003), um Ambiente de Desenvolvimento de

Software (ADS) centrado em processo e baseado em ontologias. Nesse projeto, parte-se do pressuposto que, se as ferramentas do ADS são construídas baseadas em ontologias, a integração das mesmas é facilitada, pois os conceitos envolvidos são bem definidos e compartilhados. Tal ambiente vem sendo desenvolvido no Laboratório de Engenharia de Software (LabES) da Universidade Federal do Espírito Santo (UFES).

Constituindo a base ontológica do ambiente, ODE conta com as seguintes ontologias: Ontologia de Processo de Software (FALBO, 1998) (BERTOLLO, 2006), Ontologia de Artefatos (NUNES, 2005), Ontologia de Gerência de Configuração de Software (NUNES, 2005), Ontologia de Qualidade de Software (DUARTE, 2001), Ontologia de Organizações de Software (RUY, 2006) e Ontologia de Riscos de Software (FALBO et al., 2004a). Construídas a partir dessa base ontológica, há ferramentas para apoiar a definição de processos de software (BERTOLLO et al., 2006), alocação de recurso, modelagem utilizando UML (SILVA, 2003), gerência de riscos (FALBO et al., 2004a), estimativas (CARVALHO et al., 2006), gerência de configuração (NUNES et al., 2006) etc.

Além disso, essas ontologias fornecem uma sólida conceituação para a infra-estrutura de Gerência de Conhecimento do ambiente (NATALI, 2003), a qual provê serviços de criação e captura de conhecimento, manutenção de conhecimento, uso de conhecimento, recuperação e acesso de conhecimento e disseminação pró-ativa de conhecimento.

Entretanto ODE não conta com ferramentas específicas para apoiar a maior parte do processo de Engenharia de Requisitos e os serviços de Gerência de Conhecimento existentes são, de certa forma, muito gerais, ou seja, não estão focados no apoio ao reúso de conhecimento específico do contexto da ER. Desse modo, o objetivo maior deste trabalho é apoiar a execução do processo de Engenharia de Requisitos, fornecendo um ferramental integrado a ODE e baseado em ontologias, que conte com serviços de Gerência de Conhecimento para apoiar a reutilização de itens de conhecimento no contexto da ER.

Na prática, tal objetivo geral pode ser desmembrado em dois objetivos mais específicos:

- Como não há um ferramental apropriado para tratar o processo de ER em ODE, o primeiro objetivo é construí-lo para dar suporte às funcionalidades básicas de Engenharia de Requisitos, como, por exemplo, cadastro de requisitos, cadastro de tipos de requisitos, definição de ligações de rastreabilidade e geração de relatórios de rastreabilidade.

- O segundo objetivo, por sua vez, é fornecer serviços de Gerência de Conhecimento para apoiar a reutilização de conhecimento no contexto da ER. Tais serviços devem estar integrados à ferramenta de apoio à ER mencionada anteriormente.

1.3 Metodologia

Este trabalho teve início com uma revisão bibliográfica sobre a área de Engenharia de Requisitos, na qual foram avaliados e discutidos artigos científicos, relatórios técnicos, livros, trabalhos acadêmicos e ferramentas de apoio à Engenharia de Software. O foco desse estudo foi investigar a área de Engenharia de Requisitos para entender seus conceitos e levantar requisitos a serem tratados no ferramental de apoio à ER que seria construído no contexto deste trabalho.

Após essa etapa, iniciou-se outra revisão bibliográfica, agora, sobre as áreas de Ontologias, Padrões de Análise e Gerência de Conhecimento. O objetivo desse estudo foi entender os conceitos envolvidos em cada uma dessas áreas, bem como suas aplicações. Paralelamente a essa revisão bibliográfica construiu-se uma Ontologia de Requisitos de Software, a qual serviria de base para a construção da ferramenta de apoio à ER em ODE. Tal ontologia foi publicada nos anais do IX Workshop Íbero-Americano de Engenharia de Requisitos e Ambientes de Software (IDEAS'2006), realizado em La Plata, Argentina, de 24 a 28 de abril de 2006, no artigo intitulado "Uma Ontologia de Requisitos de Software" (NARDI et al., 2006).

Do estudo sobre ontologias, decidiu-se utilizá-las como itens de conhecimento formais a serem reutilizados pelos desenvolvedores durante o processo de ER[1]. Além disso, pelo fato dos padrões de análise serem soluções já consolidadas no desenvolvimento de sistemas, decidiu-se também tratá-los com itens de formais de conhecimento que estariam à disposição dos desenvolvedores, a fim de não precisarem reinventar uma determinada solução. Seguindo a mesma linha de raciocínio, entendeu-se que a utilização de modelos de objetos gerados em

[1] Vale lembrar que, em ODE, ontologias são utilizadas também como um modelo formal de domínio capaz de facilitar a integração de ferramentas e fornecer uma base formal para os serviços de Gerência de Conhecimento. Assim, ontologias foram utilizadas neste trabalho também com esse propósito.

projetos similares também seria interessante no contexto da gerência de conhecimento apoiando a ER.

No que tange ao uso de ontologias para apoiar a ER, a proposta inicial era trabalhar com conhecimento acerca de domínio e conhecimento acerca de tarefa. Para tanto, poder-se-ia utilizar ontologias de domínio e ontologias de tarefa e um possível mapeamento entre o conhecimento representado por ambas. No entanto, essa proposta não foi levada adiante, pois se entendeu que, a despeito de haver trabalhos abordando o mapeamento entre conhecimento de domínio e conhecimento de tarefa (ANGELE et al., 1996) (CHANDRASEKARAN et al., 1998) (BREUKER et al., 1994) (MIZOGUCHI et al., 1995),há muito pouco de concreto acerca do uso de ontologias de tarefa. Assim, manteve-se a proposta apenas com ontologias de domínio.

Após a construção da Ontologia de Requisitos de Software, iniciou-se o trabalho de construção da ferramenta de apoio à Engenharia de Requisitos de ODE, a qual recebeu o nome de *ReqODE*. Durante essa construção, houve uma interação com a VixTeam Consultoria e Sistemas, empresa parceira do LabES, no sentido de tornar a ferramenta aderente ao CMMI (*Capability Maturity Model Integration*) (CHRISSIS et al., 2003), uma vez que essa empresa está em um processo de certificação e almeja utilizar *ReqODE* para tal fim.

Durante o desenvolvimento de *ReqODE*, iniciou-se um estudo mais aprofundado da infra-estrutura de Gerência de Conhecimento (GC) de ODE. A partir desse estudo, decidiu-se utilizar alguns serviços já existentes, tais como aqueles relacionados a lições aprendidas, busca de artefatos de projetos similares, consulta a pacotes de mensagens etc, e tornar direto o acesso a eles a partir de *ReqODE*. Além disso, pensou-se em fornecer serviços mais direcionados à reutilização de itens contextualizados com a ER, a saber requisitos, ontologias, padrões de análise e modelos de objetos de projetos similares. Assim, de um modo geral, organizaram-se os serviços de GC existentes em ODE para apoiarem a ER e outros foram criados para apoiar o reúso de itens de conhecimento ainda não considerados.

Definido o trabalho a ser desenvolvido acerca dos serviços de Gerência de Conhecimento, iniciou-se a integração do editor de ontologias ODEd (SOUZA, 2004), que permite a modelagem de ontologias usando LINGO (FALBO, 1998) como linguagem de modelagem, com a ferramenta OODE (SILVA, 2003), a qual apóia a modelagem utilizando a UML (*Unified Modeling Language*). Isso foi feito porque, como um dos objetivos do trabalho é a integração de ferramentas para apoiar o desenvolvedor, OODE passaria a permitir a

construção tanto de Modelos de Objetos e Padrões de Análise quanto de Ontologias. Além disso, ao se construir ontologias em OODE, o perfil UML proposto por Mian (2003) passaria a ser utilizado para a modelagem de ontologias, obtendo-se uma uniformidade representacional de todos os modelos construídos a partir de OODE.

Após isso, serviços de reúso de ontologias, padrões de análise e modelos de objetos foram incorporados a OODE, que, por sua vez, foi integrado a *ReqODE*. Além disso, a ferramenta de ER de ODE também passou a contar com serviços que facilitam o reúso de requisitos de projetos similares. Dessa maneira, foi estabelecido um ferramental integrado a um ADS que objetiva apoiar a Engenharia de Requisitos, fornecendo itens de conhecimento por meio de serviços de Gerência de Conhecimento.

1.4 Organização do Trabalho

Nesta dissertação, além deste capítulo, há mais cinco capítulos, como segue:

- Capítulo 2 – *Engenharia de Requisitos*: fornece uma fundamentação teórica acerca do domínio da Engenharia de Requisitos, focando no processo de Engenharia de Requisitos, tratando suas atividades individualmente e discutindo, por fim, alguns dos obstáculos enfrentados na Engenharia de Requisitos e possibilidades de solução.

- Capítulo 3 – *Reutilização no Desenvolvimento de Software e Aspectos Relacionados*: fornece uma fundamentação teórica acerca da reutilização de software, com ênfase em aspectos relevantes para este trabalho, a saber Análise de Domínio, Ontologias, Padrões de Análise, Gerência de Conhecimento e Ambientes de Desenvolvimento de Software.

- Capítulo 4 – *Uma Ontologia de Requisitos de Software*: apresenta a Ontologia de Requisitos de Software desenvolvida seguindo a abordagem proposta pelo método SABiO, começando pelas questões de competência, passando pela integração com ontologias existentes e chegando à captura e formalização da ontologia.

- Capítulo 5 – *Apoio à Engenharia de Requisitos e ao Reúso de Itens de Conhecimento*: apresenta a ferramenta *ReqODE* e como esta foi construída a

partir da Ontologia de Requisitos, além de apresentar como os serviços de Gerência de Conhecimento foram modelados e integrados a *ReqODE* e a OODE a fim de favorecer a reutilização de itens de conhecimento no contexto da Engenharia de Requisitos.

- Capítulo 6 – *Considerações Finais*: apresenta as conclusões sobre o trabalho desenvolvido e apresenta contribuições e propostas para trabalhos futuros.

Capítulo 2

Engenharia de Requisitos

O objetivo deste capítulo é fornecer uma fundamentação teórica acerca do domínio da Engenharia de Requisitos. Assim, ele: (i) destaca a área de Engenharia de Requisitos, contextualizando-a em relação à Engenharia de Software e definindo o que vem a ser requisito, (ii) explicita um processo de Engenharia de Requisitos tratando suas atividades individualmente, (iii) fornece uma visão de como os modelos e normas de qualidade abordam o tratamento a requisitos e (iv) discute alguns obstáculos enfrentados pela Engenharia de Requisitos e como eles podem ser enfrentados.

2.1 Introdução

Há décadas a utilização de software era extremamente pontual, ou seja, se resumia a áreas específicas como bélica, engenharia, astronomia etc. A restrição na produção de software se dava pelo grande custo de se adquirir e manter um computador, bem como o elevado custo de fabricação de software para rodar nessas máquinas, uma vez que essa fabricação tinha um caráter artesanal (PRESSMAN, 2002).

Após isso, com a disseminação dos computadores pessoais, o mundo começou a descobrir as possibilidades de aplicação de software, ou seja, a demanda por software já existia, mas ainda não havia sido descoberta. Agora, era preciso produzir software para suprir as necessidades dos usuários de computadores, sendo que entre eles se encontravam tanto empresas quanto usuários domésticos (PRESSMAN, 2002). No entanto, surgiu um problema: era preciso produzir software de maneira mais eficiente e barata que aquela artesanal de décadas atrás, caso contrário, não seria possível atender a todos. Sendo assim, era preciso criar ferramentas, técnicas e métodos. Era preciso "engenhar".

Segundo (PRESSMAN, 2002),

> A engenharia de software é um rebento da engenharia de sistemas e de hardware. Ela abrange um conjunto de três elementos fundamentais – métodos, ferramentas e procedimentos – que possibilita ao gerente o controle do processo de desenvolvimento do software e oferece ao profissional uma base para a construção de software de alta qualidade produtivamente.

Dentro da Engenharia de Software existem subáreas que atuam em contextos mais específicos como, por exemplo, a Engenharia de Requisitos. Segundo ZAVE (1997), a

Engenharia de Requisitos de Software é o ramo da Engenharia de Software que se preocupa com as metas do mundo real que motivam o desenvolvimento de funções e a descoberta de restrições em sistemas de software. Também se preocupa com o relacionamento desses fatores com as especificações precisas do comportamento do software e com sua evolução.

Numa visão um pouco mais ampla, no que tange à interseção de áreas de conhecimento, a Engenharia de Requisitos é uma área multidisciplinar, centrada no ser humano e em seus problemas. Deve, portanto, investigar como as pessoas percebem e entendem o mundo ao seu redor, como elas interagem e como a sociologia dos locais de trabalho afeta suas ações. A Engenharia de Requisitos se utiliza, portanto, das ciências sociais e cognitivas, tais como a filosofia, a psicologia cognitiva, a sociologia e a lingüística, dentre outras, para fornecer fundamentos teóricos e técnicas para elicitar e modelar requisitos (NUSEIBEH; EASTERBROOK, 2000).

Numa visão mais pragmática, KOTONYA et al. (1998) dizem que a Engenharia de Requisitos é um processo sistemático de levantamento, análise e negociação, documentação, validação e gerência de requisitos.

Assim, com base nas definições anteriores, pode-se dizer que a Engenharia de Requisitos é um ramo da Engenharia de Software que lida com os requisitos acerca de um software a ser construído e que, para tanto, utiliza um processo, além de fazer interface com outras áreas do conhecimento com o objetivo de melhor desempenhar a tarefa de engenhar requisitos.

Este capítulo está estruturado da seguinte forma: a seção 2.2 fornece uma base teórica acerca de requisito; a seção 2.3 trata do processo de Engenharia de Requisitos, destacando suas atividades individualmente; a seção 2.4 fornece uma visão do tratamento a requisitos sugerido pelos principais modelos e normas de qualidade de processo; na seção 2.5 são abordados obstáculos enfrentados pela Engenharia de Requisitos e possibilidades de solução; e na seção 2.6 são destacadas as considerações finais.

2.2 Requisitos

No foco da Engenharia de Requisitos estão os requisitos. Para lidar com eles, essa área de pesquisa propõe, por exemplo, uma série de métodos, técnicas e ferramentas. Mas antes de

prosseguir pelas práticas sugeridas pela Engenharia de Requisitos, é necessário consolidar a idéia de requisito.

Existem várias definições de requisito. Obviamente, não é objetivo deste texto listar todas, mas explicitar algumas que possam dar uma idéia do que vem a ser um requisito para que as discussões seguintes, acerca de Engenharia de Requisitos, possam ser construídas sobre essa base conceitual.

Na visão de KOTONYA et al. (1998), requisitos de sistema são especificações de serviços que o sistema deve prover, restrições sob as quais o sistema deve operar, propriedades gerais do sistema e conhecimentos que são necessários para desenvolvê-lo. Já Robertson et al. (1999) definem requisitos como alguma coisa que o produto tem que fazer ou uma qualidade que precisa estar presente.

Segundo o IEEE Standard 610-1990 (apud TOGNERI, 2002), um requisito é: (1) uma condição ou capacidade necessária para o usuário resolver um problema ou alcançar um objetivo, (2) uma condição ou capacidade que deve ser encontrada ou possuída por um sistema ou componente do sistema para satisfazer um contrato, padrão, especificação ou outro documento imposto formalmente ou (3) uma representação documentada de uma condição ou capacidade como em (1) ou (2).

Portanto, com base nas definições acima, podem-se destacar, basicamente, três aspectos que requisitos podem explicitar: (1) serviços que um sistema deve prover; (2) características que um sistema deve possuir; e (3) restrições que devem ser satisfeitas no processo de desenvolvimento do sistema. Assim, se faz necessário estabelecer uma taxonomia para classificar esses requisitos, de modo que facilite sua organização e gerência.

SOMMERVILLE (2004) destaca alguns tipos de requisitos sob dois pontos de vista: nível de descrição e tipo de informação documentada pelo requisito.

Em relação ao nível de descrição tem-se:

- **Requisitos de Usuário:** são declarações sobre o que o sistema deve fornecer e como deve operar. Está em um nível de abstração mais alto, no que tange à descrição.
- **Requisitos de Sistema:** são declarações mais detalhadas e precisas do que aquelas dos requisitos de usuário. Em geral um requisito de usuário pode ser decomposto em vários requisitos de sistema.

Já em relação ao tipo de informação que o requisito documenta tem-se:

- **Requisitos Funcionais:** são declarações de quais funções o sistema deve fornecer e como reagir diante delas, estando, portanto, relacionados aos aspectos funcionais do sistema.
- **Requisitos Não-Funcionais:** são restrições impostas aos serviços ou às funções oferecidos pelo sistema como, por exemplo, restrições de tempo e padrões organizacionais. Os requisitos não-funcionais podem ainda ser classificados mais detalhadamente, dando origem a classes de requisitos não funcionais, tais como requisitos de manutenibilidade, de portabilidade, de segurança etc.

Ainda que haja diversas classificações para requisitos na literatura, é indicado que cada organização defina sua própria classificação (SOMMERVILLE, 2004). Isso vai facilitar, entre outras coisas, o entendimento durante uma atividade de negociação ou de validação de requisitos, pois os envolvidos terão plena consciência das características de cada requisito.

2.3 O Processo de Engenharia de Requisitos

Para que a Engenharia de Requisitos seja estabelecida de forma aceitável, é interessante definir um processo. Os processos de engenharia de requisitos lidam com atividades, métodos, técnicas e práticas, e devem ser seguidos para derivarem, validarem e manterem os artefatos gerados.

Uma completa descrição de um processo de engenharia de requisitos deve incluir as atividades a serem seguidas, a estrutura ou seqüência dessas atividades, quem é responsável por cada atividade, as entradas e saídas das atividades e as ferramentas usadas para dar suporte à engenharia de requisitos (KOTONYA et al., 1998). Além disso, a definição de um modelo de ciclo de vida para o processo é de grande importância e pode ser definido com base nas características do projeto.

A implantação desses processos em uma organização deve ser feita segundo as necessidades da mesma. Ou seja, o processo deve ser definido de acordo com as características da organização. Existem, portanto, fatores que contribuem para a variabilidade do processo de engenharia de requisitos: (1) maturidade técnica, (2) envolvimento disciplinar, (3) cultura organizacional e (4) domínio de aplicação (KOTONYA et al., 1998).

De uma maneira geral, WIEGERS (2003) destaca alguns benefícios que um processo de Engenharia de Requisitos de alta qualidade trás:

- Menos defeitos nos requisitos;
- Re-trabalho de desenvolvimento reduzido;
- Menos características desnecessárias;
- Diminuição de custos;
- Desenvolvimento mais rápido;
- Menos problemas de comunicação;
- Alterações de escopo reduzidas;
- Caos de projeto reduzido;
- Estimativas de teste de sistema mais confiáveis;
- Mais satisfação dos clientes e membros da equipe.

Na literatura, existem vários processos propostos que podem ser adaptados a cada organização. No entanto, esses processos mantêm um conjunto de aspectos em comum que permite estabelecer atividades básicas que devem ser consideradas em qualquer definição de processo de Engenharia de Requisitos (TOGNERI, 2002).

Neste trabalho, a título de ilustração será utilizado o processo de Engenharia de Requisitos proposto por KOTONYA et al. (1998), o qual possui o seguinte conjunto de atividades, descrito a seguir: Levantamento, Análise e Negociação, Documentação, Validação e Gerência de Requisitos, A figura 2.1 mostra os relacionamentos entre essas atividades.

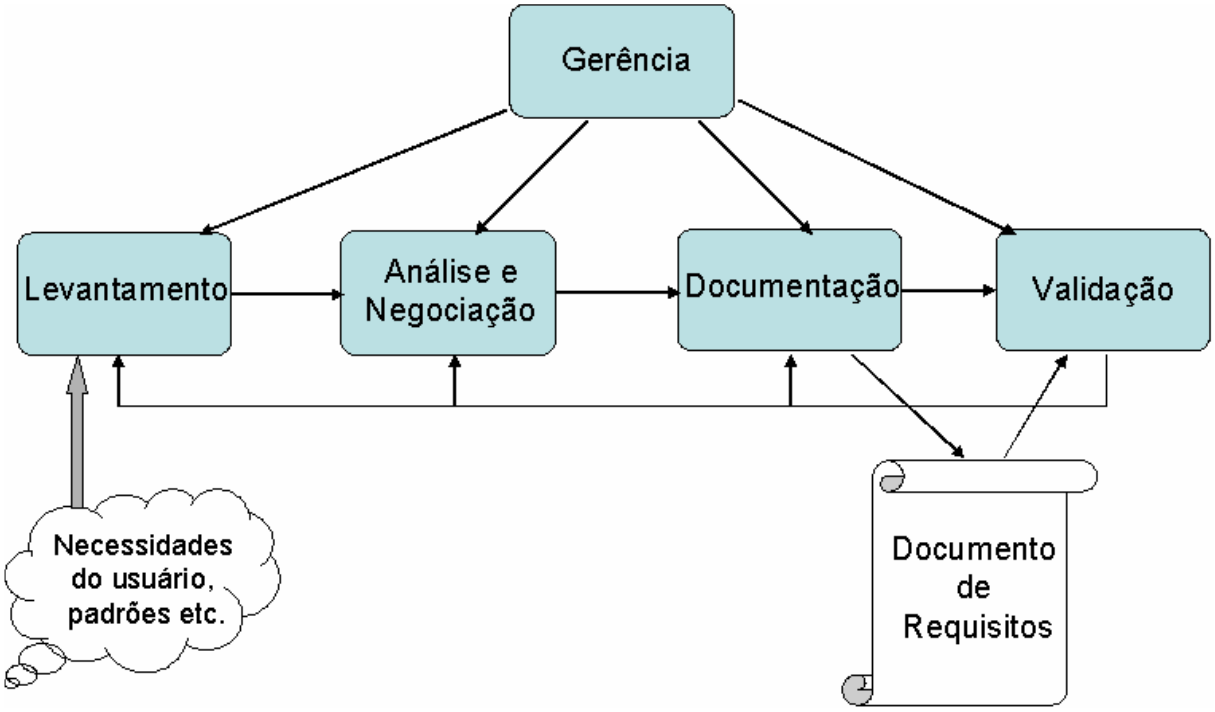


Figura 2.1 – Processo de Engenharia de Requisitos (KOTONYA et al., 1998)

2.3.1 Levantamento de Requisitos

O Levantamento de Requisitos trata, mais especificamente, da descoberta dos requisitos. Nessa atividade, um esforço conjunto de clientes, usuários e especialistas de domínio é necessário, com o objetivo de entender a organização, ou seja, seus processos, necessidades, deficiências dos sistemas de software atuais, possibilidades de melhorias, bem como restrições existentes.

KOTONYA et al. (1998) entendem o levantamento de requisitos como um processo de transferência de conhecimento, no qual engenheiros de requisitos levantam o conhecimento dos interessados (*stakeholders*) na construção do sistema. Além disso, KOTONYA et al. (1998) destacam quatro dimensões na atividade de levantamento de requisitos:

- **Entendimento do domínio da aplicação:** entendimento geral da área na qual o sistema será aplicado;
- **Entendimento do problema:** entendimento dos detalhes do problema específico a ser resolvido com o auxílio do sistema a ser desenvolvido;
- **Entendimento do contexto do negócio:** entendimento da contribuição do sistema para que sejam atingidos os objetivos gerais da organização;

- **Entendimento das necessidades e das restrições dos interessados (*stakeholders):*** entendimento das necessidades de apoio a ser provido pelo sistema à realização do trabalho de cada um dos interessados no sistema, dos processos de trabalho a serem apoiados pelo sistema e do papel de eventuais sistemas existentes na execução e condução dos processos de trabalho. Consideram-se, como interessados no sistema, todas as pessoas que são afetadas pelo sistema de alguma maneira, dentre elas clientes, usuários finais e gerentes de departamentos onde o sistema será instalado.

Na execução da atividade de levantamento de requisitos, várias técnicas podem ser utilizadas, tais como: técnicas tradicionais (englobam técnicas genéricas, tais como o uso de questionários e pesquisas, investigação de documentos e entrevistas), técnicas de levantamento em grupo (exploram as dinâmicas de grupo), prototipação (usada quando existe incerteza sobre os requisitos ou quando se torna necessário ter um *feedback* inicial dos usuários), técnicas dirigidas a modelos (fornecem um modelo para o tipo de informação a ser obtido e usam esse modelo para conduzir o processo de levantamento), técnicas cognitivas (incluem técnicas desenvolvidas originalmente para aquisição de conhecimento em sistemas baseados em conhecimento) e técnicas contextuais (incluem o uso de técnicas de etnografia, tais como observação dos participantes e análise de conversação) (TOGNERI, 2002).

2.3.2 Análise e Negociação de Requisitos

A atividade de Análise e Negociação envolve a modelagem dos requisitos identificados no levantamento de requisitos, auxilia na descoberta de problemas nesses requisitos e favorece a obtenção de requisitos que satisfaçam, na medida do possível, as partes envolvidas (clientes, usuários etc).

De fato, essa atividade é um processo de descoberta, refinamento, modelagem e especificação (PRESMAN, 2002). Durante a tarefa de descoberta de problemas, os engenheiros de requisitos, juntamente com os usuários, clientes e especialistas de domínio, classificam os requisitos com problemas e tentam esclarecê-los. No entanto, é comum haver divergências acerca dos requisitos e de suas prioridades, uma vez que diferentes fontes e tipos de conhecimento devem ser registrados na especificação, como aponta Easterbrook (apud LOPES, 2002). Daí, necessita-se partir para negociações que podem ser influenciadas por questões filosóficas, políticas, sociais, organizacionais e pelos objetivos das pessoas envolvidas (NUSEIBEH et al., 2000). Em muitos casos, algumas questões não serão

respondidas e não será possível obter concordância sobre alguns requisitos, o que pode significar que não há informações suficientes para a negociação e, portanto, será necessário executar um novo estudo dos requisitos (KOTONYA et al., 1998).

Durante a atividade de Análise e Negociação de Requisitos, uma tarefa importante é a modelagem, quando são criados modelos do sistema a ser construído. Esses modelos concentram-se naquilo que o sistema deve fazer, não em como ele o faz (PRESMAN, 2002). Assim, vários métodos de modelagem podem ser utilizados e esses métodos definem a forma que o documento de especificação de requisitos deve ter (GIMENEZ, 2001). Diagramas de casos de uso, diagramas de classes, diagramas de estados e diagramas de colaboração, dentre outros, podem ser utilizados na modelagem de requisitos, dando a eles um caráter mais formal.

Segundo (PRESMAN, 2002), os modelos criados durante a atividade de análise de requisitos cumprem os seguintes papéis:

- Ajudam o analista a entender a informação, a função e o comportamento do sistema, tornando a tarefa de análise de requisitos mais fácil e sistemática;

- Tornam-se o ponto focal para a revisão e, portanto, a chave para a determinação de consistência da especificação;

A construção dos modelos de análise é uma parte importante dessa atividade, pois esses modelos serão a base para a atividade de projeto, fornecendo ao projetista uma representação essencial do software, a qual pode ser mapeada num contexto de implementação.

2.3.3 Documentação de Requisitos

A Documentação de Requisitos visa a representar os resultados da Engenharia de Requisitos em um documento oficial e formal, contendo os requisitos do software que descrevem o que o mesmo vai fazer em um nível apropriado de detalhes, sem descrever como fazê-lo.

Segundo KOTONYA et al. (1998), a linguagem natural é a única que pode ser entendida por todos os envolvidos. Dessa forma, é muito interessante que ela seja utilizada na documentação dos requisitos. No entanto, eles não descartam a utilização de diagramas e gráficos como forma de facilitar o entendimento dos requisitos. Por outro lado, Davis (apud

LOPES, 2002) destaca que, apesar da vantagem da linguagem natural de poder ser entendida por todos os participantes do processo de requisitos, ela apresenta um alto grau de ambigüidade, o que favorece o aparecimento de inconsistências.

O documento de requisitos, também chamado de Especificação de Requisitos de Software (ERS), conforme a norma IEEE 830-1998 (1998), deve ser:

- **Correto:** uma ERS é correta se, e somente se, cada requisito documentado é aquilo que o software deve possuir;
- **Não ambíguo:** uma ERS não é ambígua se, e somente se, cada requisito documentado tem apenas uma interpretação;
- **Completo:** uma ERS é completa se, e somente se, ela inclui os seguintes elementos: todos os requisitos significativos; definição de todas as respostas do software para todas as classes de entrada de dados realizáveis; todas as legendas e referências às figuras, tabelas etc e definição de todos os termos e unidades de medidas;
- **Internamente Consistente:** uma ERS é internamente consistente se, e somente se, nenhum subconjunto de requisitos individuais descrito estiver conflitante;
- **Graduado por importância e estabilidade:** para tanto, cada requisito de uma ERS deve possuir um identificador para indicar sua importância e estabilidade;
- **Verificável:** uma ERS é verificável se, e somente se, cada requisito documentado é verificável;
- **Modificável:** uma ERS é modificável se, e somente se, sua estrutura e estilo são tais que qualquer mudança nos requisitos possa ser feita de maneira fácil, completa e consistente, enquanto se mantém a estrutura e estilo originais do documento;
- **Rastreável:** uma ERS é rastreável se a origem de cada um de seus requisitos está clara e se ela facilita a referência de cada requisito no desenvolvimento futuro ou em documentos de apoio.

Essas características estão diretamente relacionadas à qualidade de um Documento de Requisitos, que envolve aspectos relativos ao controle da qualidade dos artefatos de software produzidos no processo de requisitos (SAYÃO et al., 2003).

Em relação ao conteúdo de uma ERS, esse depende de diversos fatores, como, por exemplo, a natureza do sistema que está sendo especificado, o nível de detalhamento dos requisitos, práticas organizacionais, além de restrições orçamentárias e de tempo aplicáveis ao processo de Engenharia de Software (LOPES, 2002).

Para se estruturar o conteúdo, é necessário que a organização de software defina seus modelos de documentos de requisitos de acordo com as características de classes de projeto. Para isso, pode-se, por exemplo, partir de um modelo como o proposto pelo padrão IEEE 830-1998 (1998) e adequá-lo às necessidades organizacionais.

A construção de uma ERS de qualidade conduz a vários ganhos nas próximas atividades do processo de desenvolvimento, uma vez que esse documento será utilizado, maciçamente, como insumo para essas atividades. TOGNERI (2002) destaca que, dentre outros, o documento de requisitos facilita a comunicação entre os envolvidos no desenvolvimento de software, fornece uma base para validação e verificação, e serve como base para futuras modificações ou incremento de novas funcionalidades.

2.3.4 Verificação e Validação

As atividades de Verificação, Validação e Testes (VV&T) devem ser iniciadas o quanto antes no processo de desenvolvimento, pois os custos são maiores quando os defeitos são identificados mais tardiamente. Dessa forma, a visão de que as atividades de verificação, validação e testes devam ser conduzidas somente após a codificação é, no mínimo, ultrapassada (MALDONADO et al., 2001).

O objetivo da validação é assegurar que os artefatos produzidos ao longo do processo de desenvolvimento são os artefatos corretos, ou seja, estão sendo desenvolvidos conforme as necessidades dos usuários. Já a verificação objetiva assegurar que os artefatos produzidos estão sendo desenvolvidos corretamente, isto é, estão em conformidade com os padrões organizacionais, incluindo os processos (MALDONADO et al., 2001).

Das atividades de verificação e validação, a atividade de teste é considerada um elemento crítico para a garantia da qualidade dos artefatos produzidos ao longo do desenvolvimento e, por conseguinte, do produto de software final (MALDONADO et al.,

2001). Uma das características de qualidade de um requisito bem elaborado é que ele possa ser testado. Uma boa maneira de identificar problemas nos requisitos, tais como falta de completitude ou ambigüidade, é tentar propor casos de teste para um requisito (LOPES, 2002).

MALDONADO et al. (2001) destacam que as atividades de VV&T envolvem atividades de análise dinâmica e análise estática. Na análise dinâmica objetiva-se detectar defeitos ou erros no software e envolve a execução do produto, seja o código propriamente dito ou mesmo uma especificação executável. Já a análise estática não envolve a execução do produto, mas visa a determinar propriedades do mesmo válidas para qualquer execução do produto final, podendo ser aplicada, por exemplo, em nível de especificação de requisitos.

Para avaliar a qualidade acerca de um documento de especificação de requisitos, podem-se utilizar métricas e indicadores, e a análise crítica desse artefato pode ser realizada através de inspeções e revisões por pares (SAYÃO et al., 2003). No que tange às métricas, Sayão et al. (2003) destacam métricas de verificação e validação de requisitos, métricas de evolução e apoio ao gerenciamento do processo e métricas para aferição da qualidade do documento de requisitos. Em relação às inspeções, SAYÃO et al. (2003) citam a utilização de leitura *Ad-hoc*, leitura baseada em listas de verificação (*checklists*) e leitura baseada em perspectivas, como forma de validar e verificar documentos de requisitos.

2.3.5 Gerência de Requisitos

Uma vez que os requisitos são passíveis de alteração por uma série de motivos, tem-se, portanto, a necessidade de uma atividade que apóie as demais atividades de engenharia de requisitos e do processo de software, que objetive gerenciar as mudanças nos requisitos para que todas as alterações sejam feitas de forma consistente e controlada. Essa atividade é a Gerência de Requisitos.

Requisitos podem sofrer alterações de várias naturezas e SOMMERVILLE (2004) destaca quatro tipos de requisitos voláteis:

- **Requisitos mutáveis:** são aqueles que mudam devido a mudanças no ambiente do sistema;
- **Requisitos emergentes:** são os que emergem à medida que a compreensão do sistema aumenta;

- **Requisitos conseqüentes:** são os que resultam da introdução do sistema computadorizado;

- **Requisitos de compatibilidade:** são aqueles que dependem de outros sistemas ou processos organizacionais.

Ao surgir uma necessidade de alteração em um ou mais requisitos, basicamente, o que necessita ser feito é registrar uma solicitação de mudança, que deve ser avaliada por algum membro da equipe do projeto de software. Nessa avaliação, o impacto da alteração deve ser determinado e valores de custo, esforço, tempo e viabilidade devem ser repassados ao solicitante da mudança.

Para determinar o impacto de uma alteração em um requisito, deve ser possível, por exemplo, saber quais são os requisitos dependentes desse requisito e em quais artefatos do processo de desenvolvimento esse requisito é tratado. Portanto, é necessário estabelecer uma rede de ligações de modo que um requisito e os elementos ligados a ele possam ser rastreados. Surge, então, o conceito de rastreabilidade.

2.3.5.1 Rastreabilidade

Segundo JARKE (1998) a rastreabilidade de requisitos está emergindo como uma eficiente ponte que possibilita unir evolução de sistemas com necessidades de mudança dos interessados (*stakeholders*).

A rastreabilidade de requisitos é possível, basicamente, se houver ligações entre requisitos, e entre requisitos e outros elementos do processo de software. Dessa forma, a identificação da composição de requisitos, das dependências entre requisitos, de requisitos conflitantes, da origem dos requisitos e de seus interessados, além da identificação de em qual artefato (documento, módulo, diagrama, componente etc) produzido durante o desenvolvimento de software um requisito é tratado, é de fundamental importância para que a rastreabilidade possa ser implantada (WIEGERS, 2003) (ROBERTSON et al., 1999) (KOTONYA et al., 1998).

Davis (apud KOTONYA et al., 1998) classificou quatro tipos de informações de rastreabilidade interessantes para que uma análise de impacto de mudanças possa ser feita mais facilmente:

- **Regressiva a partir dos requisitos (*backward-from traceability*):** relaciona requisitos com suas origens em outros documentos ou pessoas;

- **Progressiva a partir dos requisitos (*forward-from traceability*):** relaciona requisitos aos artefatos de projeto e implementação;

- **Regressiva em direção aos requisitos (*backward-to traceability*):** relaciona artefatos de projeto e implementação aos requisitos;

- **Progressiva em direção aos requisitos (*forward-to traceability*):** relaciona outros documentos (os quais podem ter precedido o documento de requisitos) aos requisitos relevantes.

Segundo LOPES (2002), "a dificuldade envolvida com a rastreabilidade está no grande volume de informações gerado. As informações do processo de requisitos devem ser catalogadas e associadas aos outros elementos do desenvolvimento de forma que possam ser referenciadas através dos diversos itens de informação registrados". Assim, WEIGERS (2003) destaca que ter um conjunto muito grande de atributos de requisitos não é interessante, pois possivelmente nem todos os atributos serão úteis. Nesse sentido, TORANZO et al. (2002) exploram uma proposta para melhorar o rastreamento de requisitos na qual fornecem uma classificação das informações a serem rastreadas, estando, portanto, preocupados em tratar a questão de "*O que rastrear?"*. Essa classificação aborda quatro níveis de informação: Ambiental (leis, estratégias, padrões etc), Organizacional (regras, processos etc), Gerencial (tarefas, objetos, restrições etc) e Desenvolvimento (requisitos, diagramas, programas etc).

Embora a rastreabilidade de requisitos não possa ser completamente automatizada, porque o conhecimento das ligações se origina na mente dos membros da equipe de desenvolvimento, uma vez identificadas essas ligações, ferramentas são importantíssimas para ajudar a gerenciar a grande quantidade de informações de rastreabilidade (WIEGERS, 2003).

Embora a manutenção da rastreabilidade possa parecer trabalhosa, muitos benefícios são obtidos com a sua implementação (WIEGERS, 2003): (i) certificação, (ii) análise de impacto de mudança, (iii) manutenção, (iv) acompanhamento de projeto, (v) reengenharia, (vi) reúso, (vii) redução de risco e (viii) testes.

Uma questão extremamente importante que atinge a rastreabilidade é a cultura organizacional. Os envolvidos precisam empenhar-se nessa atividade e vislumbrar benefícios pessoais para auxiliar na implementação da rastreabilidade. Para tanto, é necessário capacitar pessoas e definir uma filosofia de trabalho. Assim, pode ser que a implementação da Gerência de Requisitos apoiada na rastreabilidade não seja uma tarefa fácil (LOPES, 2002).

2.4 Engenharia de Requisitos e Normas e Modelos de Qualidade

Ao longo dos últimos anos, muito se tem falado sobre a adoção de modelos e normas de qualidade no processo de software. Entre os vários modelos, podem-se citar o CMMI (*Capability Maturity Model Integration*) (CHRISSIS et al., 2003) e o MPS.BR (SOFTEX, 2005) e no caso das normas, existe, por exemplo, a ISO/IEC[2.1] 12207 (ISO, 1995) (ISO, 2002) (ISO, 2004).

O CMMI é uma evolução do CMM (*Capability Maturity Model*) (FIORINI et al., 1998) e tem como objetivo fornecer diretrizes para a definição e melhoria de processos de uma organização. Entre os vários modelos do CMMI pode-se destacar o CMMI-SE/SW (*Capability Maturity Model Integration for Systems Engineering/Software Engineering*), o qual é um dos componentes do conjunto de produtos do CMMI voltado para as áreas de engenharia de sistemas e engenharia de software.

A norma ISO/IEC 12207 objetiva estabelecer um conjunto de processos de ciclo de vida de software de maneira a auxiliar as organizações na definição e melhoria de seus processos.

O MPS.BR (Melhoria de Processo do Software Brasileiro) é um modelo brasileiro que está em desenvolvimento desde dezembro de 2003 e tem como objetivo auxiliar micro, pequenas e médias empresas a definirem e a melhorarem seus processos de maneira a se certificarem. É um modelo direcionado para a realidade brasileira. Um fato relevante é que na concepção do MPS.BR foram consideradas as normas ISO/IEC 12207 e ISO/IEC 15504, além do CMMI. Dessa maneira, pode-se garantir que o MPS.BR está em conformidade com os padrões internacionais (SOFTEX, 2005).

Uma vez abordados alguns modelos e normas de qualidade, vale ressaltar que eles têm em comum a busca pela produção de software de qualidade. Para tanto, desenvolver e gerenciar requisitos de maneira adequada são considerados aspectos essenciais e cada modelo ou norma, à sua maneira, define processos e/ou diretrizes para isso.

[2.1] A sigla ISO origina-se *de International Organization for Standardization*, enquanto a sigla IEC origina-se *de International Electrotechnical Commission*.

O CMMI, em seus cinco níveis de maturidade (1- Inicial, 2- Gerenciado, 3- Definido, 4- Gerenciado Quantitativamente e 5- Otimizado), trata de requisitos, basicamente, no nível 2 (Gerenciado), no qual aborda a Gerência de Requisitos, e no nível 3 (Definido), no qual trata do Desenvolvimento de Requisitos.

Na Gerência de Requisitos objetiva-se garantir consistência dos requisitos e consistência entre os requisitos e os vários produtos gerados ao longo do processo de software. Para isso o CMMI sugere como práticas específicas da gerência de requisitos: (i) Obter um Entendimento dos Requisitos; (ii) Obter Compromissos com os Requisitos; (iii) Gerenciar as Mudanças de Requisitos; (iv) Manter a Rastreabilidade Bidirecional de Requisitos e (v) Identificar Inconsistências entre o Trabalho do Projeto e os Requisitos.

O Desenvolvimento de Requisitos objetiva produzir e analisar os requisitos de clientes bem como os requisitos de produto. Neste caso o CMMI sugere como práticas específicas dessa área de processo: (i) Levantar Necessidades; (ii) Desenvolver os Requisitos de Clientes; (iii) Desenvolver Requisitos de Produtos; (iv) Estabelecer Requisitos de Produtos e de Componentes de Produto; (v) Alocar Requisitos de Componentes de Produto; (vi) Identificar Requisitos de Interface; (vii) Estabelecer Conceitos e Cenários Operacionais; (viii) Estabelecer uma Definição da Funcionalidade Exigida; (ix) Analisar os Requisitos para Alcançar o Equilíbrio e (x) Validar os Requisitos com Métodos Abrangentes.

No âmbito da Engenharia de Requisitos, a ISO/IEC 12207, mais especificamente sua Emenda 1 (ISO, 2002), destaca em seu Processo de Desenvolvimento três sub-processos: Levantamento de Requisitos, Análise dos Requisitos do Sistema e Análise dos Requisitos do Software. O propósito do Levantamento de Requisitos é obter, processar e acompanhar as necessidades e os requisitos do cliente ao longo da vida do produto e/ou serviço, de forma a estabelecer a linha básica de requisitos que serve de base para a definição dos produtos de trabalho necessários. A Análise dos Requisitos do Sistema tem como propósito transformar os requisitos dos envolvidos em um conjunto de requisitos técnicos desejados para o sistema, que guiarão o seu projeto (*design*). Por fim, a Análise dos Requisitos do Software visa a estabelecer os requisitos dos elementos de software do sistema.

O MPS.BR define em seu nível G o processo de Gerência de Requisitos e no nível D o processo de Desenvolvimento de Requisitos. O processo de Gerência de Requisitos objetiva controlar alterações nos requisitos e identificar inconsistências entre os requisitos e outros produtos de trabalho. Nesse processo tem-se como exemplos de resultados esperados: comunicação contínua com o cliente, obtenção do entendimento dos requisitos,

gerenciamento de mudanças dos requisitos ao longo do projeto, geração de uma matriz bidirecional etc. O Desenvolvimento de Requisitos, por sua vez, objetiva identificar e analisar os requisitos junto ao cliente. No contexto desse processo, são alguns dos resultados esperados: identificação das necessidades, restrições e expectativas do cliente, definição de um conjunto de requisitos funcionais e não-funcionais do problema, validação de requisitos, desenvolvimento e manutenção dos requisitos identificados etc.

De uma maneira geral, pode-se constatar que a Engenharia de Requisitos, pela sua importância no contexto do desenvolvimento de software é abordada pelos principais modelos e normas de qualidade de processo de software hoje vigentes. É fato que engenhar requisitos se torna imprescindível no desenvolvimento de software de qualidade. No entanto, uma das grandes dificuldades na implantação desses modelos e normas é determinar até que ponto políticas de desenvolvimento e gerência de requisitos são suficientes para garantir qualidade no produto final.

2.5 Obstáculos na Engenharia de Requisitos e Perspectivas de Solução

A área de Engenharia de Requisitos é uma área relativamente nova. Portanto, ainda existem muitos obstáculos a serem enfrentados e, por ser uma área crucial para o desenvolvimento de software, vários trabalhos têm sido desenvolvidos com o objetivo de minimizar os problemas nela enfrentados.

LAMSWEERD (2000) cita que a complexidade do processo de engenharia de requisitos reside em questões como:

- Além de aspectos funcionais, vários outros de caráter não-funcional têm que ser tratados, tais como segurança, portabilidade, eficiência etc, sendo que, muitas vezes, esses aspectos são conflitantes;
- As pessoas envolvidas no processo de requisitos, geralmente, possuem pontos de vista distintos e expectativas divergentes;
- Especificações de requisitos possuem deficiências, tais como contradições, ambigüidades etc. Essas deficiências, geralmente, ocasionam problemas nas atividades seguintes do processo de software, nas quais o custo de correção torna-se muito alto.

Considerando os problemas enfrentados no processo de requisitos, NUSEIBEH et al. (2000) acreditam que os maiores desafios da Engenharia de Requisitos nos próximos anos são:

- Desenvolvimento de novas técnicas para modelagem e análise formal das propriedades do ambiente, em oposição ao comportamento do software. Tais técnicas devem se concentrar na necessidade de lidar com modelos inconsistentes, incompletos e evolutivos;
- Diminuir a distância entre abordagens de levantamento de requisitos baseadas em investigação contextual e técnicas de análise e especificação mais formais;
- Modelos mais ricos para análise e captura de requisitos não-funcionais;
- Melhor entendimento do impacto das escolhas arquiteturais de software na priorização e evolução de requisitos;
- Reúso de modelos de software;
- Treinamento multidisciplinar para os especialistas em requisitos.

CROFTS et al. (2004) expõem um ponto de vista mais diferenciado, dizendo que nos últimos anos houve uma importante mudança no contexto organizacional face à Engenharia de Requisitos. No contexto de um desenvolvimento globalizado, eles destacam desafios como: equipes trabalhando virtualmente, software tendo que operar em múltiplos contextos e tratamento de necessidades de diferentes culturas e jurisdições legais.

LAMSWEERD (2000) cita, ainda, que muito tem que ser feito nos próximos anos no sentido de buscar o aprimoramento do processo de engenharia de requisitos e destaca que esforços devem ser aplicados em questões como:

- Diminuição da distância entre a pesquisa de Engenharia de Requisitos e Arquitetura de Software, pois muitas vezes existem conflitos entre requisitos voláteis e arquiteturas estáveis;
- Utilização da abordagem “defina você mesmo” como forma de envolvimento de usuários finais na construção de sistemas;
- Captura de mais conhecimento sobre vários aspectos (tarefa, domínio etc.) nos modelos de domínio e de requisitos. No entanto, deve-se atentar em como

representar esse conhecimento de forma que seja simples, preciso e fácil de usar;

- Melhoria da produção dos documentos de requisitos, os quais são escritos e estruturados de maneira pobre.

Outro obstáculo existente na Engenharia de Requisito está relacionado à Gerência de Requisitos. Com a difusão das normas e modelo de qualidade, tais como o CMMI, a ISO/IEC 12207 e o MPS.BR, e conseqüente adoção dos mesmos por parte das organizações de desenvolvimento de software, tem-se percebido que um grande problema está em atender à rastreabilidade. Neste sentido, várias questões ainda incomodam os responsáveis pela definição de uma abordagem de rastreabilidade:

- Até que ponto é necessário rastrear requisitos e artefatos do processo de desenvolvimento?
- Qual o nível de granularidade dos elementos que deve ser tratado nas ligações de rastreabilidade?
- Quais os elementos que devem ser relacionados?

De fato, essas questões são de difícil resposta, porque dependem de cada organização onde será definida a abordagem de rastreabilidade, ou seja, dependem da maturidade dos processos e da cultura organizacional.

De um modo geral, os obstáculos enfrentados pela Engenharia de Requisitos podem ser abordados levando-se em consideração aspectos sociais, de forma a melhor entender as necessidades dos usuários e aspectos técnicos, os quais, segundo ZAVE (1997) podem ser trabalhados por meio de propostas de solução orientadas a processo, a produto ou a medição, dentre outras. Assim, o monitoramento e a medição da resolução dos problemas e do desenvolvimento das atividades, e o rastreamento dessas medições ao longo do tempo podem apoiar a melhoria do processo de Engenharia de Requisitos.

2.6 Considerações do Capítulo

Neste capítulo foram abordados os conceitos da área de Engenharia de Requisitos com o objetivo de fornecer ao leitor uma base teórica suficiente para a leitura dos próximos capítulos e para fundamentar a proposta desta dissertação.

Após tratar de aspectos relacionados à Engenharia de Requisitos como sub-área da Engenharia de Software, ao esclarecimento do que vem a ser um requisito e ao processo de Engenharia de Requisitos e suas atividades, passou-se a abordar questões relacionadas aos obstáculos e limitações enfrentados pela Engenharia de Requisitos.

Esse último item é de extrema importância, pois, como já citado, a existência desses obstáculos e limitações motiva estudos e o surgimento de vários trabalhos nessa área do conhecimento. Assim, é nessa linha de raciocínio que se insere esta dissertação de mestrado, visando a propor abordagens que possam ajudar a Engenharia de Requisitos a superar seus obstáculos.

Capítulo 3

Reutilização no Desenvolvimento de Software e Aspectos Relacionados

O objetivo deste capítulo é fornecer uma fundamentação teórica acerca de reutilização no contexto do processo de software. Nesse sentido, tópicos afins que são relevantes para esse trabalho foram abordados, tais como Análise de Domínio, Ontologias, Padrões de Software (em específico, Padrões de Análise), Gerência de Conhecimento e Ambientes de Desenvolvimento de Software. Dessa maneira, uma vez fornecido tal embasamento teórico, os capítulos subseqüentes desta dissertação poderão abordar esses assuntos considerando que o leitor já tem um entendimento básico dos mesmos.

3.1 Introdução

A questão da busca por reúso está relacionada à resolução de problemas. Sendo assim, quando se abordam problemas similares, tende-se a utilizar soluções semelhantes. Neste sentido, soluções vão sendo desenhadas para uma determinada classe de problemas até o ponto de serem padronizadas e documentadas para posterior busca e utilização (PRIETO-DIAZ, 1993).

A reutilização no contexto do desenvolvimento de software tem como objetivos melhorar o cumprimento de prazos, diminuir custos e obter produtos de maior qualidade (GIMENES et al., 2005), uma vez que artefatos de software podem ser reutilizados com o intuito de diminuir o tempo de construção e investindo, assim, esforço na adaptação e na reutilização de itens já construídos. Uma vez que esses itens tenham sido desenvolvidos para reúso, o esforço de adaptação e reutilização tende a ser minimizado.

Freeman (apud BARROCA et al., 2005) considera que no reúso de software estão incluídos todos os artefatos gerados ao longo do processo de desenvolvimento, sendo considerados, portanto, modelos, especificações, planos, código-fonte etc. Basili e Rombach (apud GIMENES et al., 2005) definem reutilização de software, destacando explicitamente o reúso de conhecimento ao longo do ciclo de vida de um projeto de software. Assim, se for considerado que conhecimento é gerado ao longo do processo de software, pode-se dizer que as definições de Freeman e de Basili e Rombach apontam para um mesmo horizonte, ou seja, o processo de software envolve tanto a produção quanto o consumo de conhecimento.

No entanto, para que o conhecimento seja capturado adequadamente e possa ser reutilizado com eficiência, é necessário contemplar, no processo de software, atividades relacionadas a tal propósito. Para tanto, pode-se utilizar a Análise de Domínio, que, segundo PRIETO-DIAZ (1990) é o processo pelo qual informação utilizada no desenvolvimento de software é identificada, capturada e organizada com o propósito de torná-la reutilizável na criação de novos sistemas. A Análise de Domínio é parte de um processo de Engenharia de Domínio que, por sua vez, segundo a Emenda 1 da ISO/IEC 12207 (ISO, 2002), visa a desenvolver e manter modelos, arquiteturas e ativos de domínio. Porém, para que a Análise de Domínio produza frutos, é necessário que ela esteja contemplada no processo de software e que conte com aparatos que objetivem promover a sua realização como, por exemplo, ferramentas e ambientes de apoio (CIMA et al., 1997).

Um dos produtos gerados pela Análise de Domínio são os chamados modelos de domínio, que buscam fornecer informações acerca do domínio de uma maneira passível de utilização. Assim, é desejável que esses modelos estejam bem formalizados, de modo a eliminar possíveis ambigüidades e *gaps* semânticos. Para tanto, podem-se utilizar ontologias, que, segundo GRUBER (1995), consistem de uma especificação formal e explícita de uma conceituação compartilhada. Essas conceituações podem tratar de domínios ou tarefas específicas, bem como de conceituações genéricas como, por exemplo, espaço e tempo. Nesse sentido, uma ontologia procura representar conceitos, relações, propriedades e axiomas com o objetivo de explicitar um universo de discurso.

Além das ontologias, podem-se utilizar, também, padrões, que embora possuam um nível de abstração menor que as ontologias, documentam soluções padronizadas para problemas recorrentes. Entre os vários tipos de padrões, podem ser citados: padrões de projeto, padrões de análise, padrões de processo, padrões arquiteturais etc, cada um documentando soluções de áreas ou de abstrações particulares. Ainda que todos esses tipos de padrões possam ser utilizados na Engenharia de Software, o foco deste trabalho está relacionado a padrões de análise. Isso porque, como esta dissertação trata de aspectos relacionados à Engenharia de Requisitos, os padrões de análise estão mais próximos das características dessa sub-área da Engenharia de Software (GIMENEZ et al., 2005). Segundo Fernandez (apud COTA et al., 2004), um padrão de análise é um conjunto de classes e associações que tem algum significado no contexto de uma aplicação, isto é, um modelo conceitual de uma parte da aplicação. Nesse sentido, volta-se à questão da modelagem

conceitual do domínio de uma aplicação, no sentido de auxiliar as atividades do processo de desenvolvimento.

Mas durante a condução de um processo de software, podem ser citados vários outros tipos de itens de conhecimento que podem ser reusados, tais como lições aprendidas sobre novas tecnologias (o que é determinante no contexto de desenvolvimento, pois novas tecnologias têm surgido a todo tempo), conhecimento sobre competências dos membros de uma organização – "Quem sabe o quê?" – o que pode direcionar os esforços pela busca por soluções de problemas, *design rationale* envolvido nas soluções de problemas etc. No entanto, é necessário que todo esse conhecimento seja gerenciado, considerando sua captura, utilização, disseminação, manutenção etc. Nesse sentido, uma maneira de se alcançar esse controle é por meio da utilização de serviços de gerência de conhecimento (NATALI, 2003). Tais serviços fornecem acesso ao conhecimento organizacional, facilitando o reúso por parte dos membros da organização. Portanto, para que esse reúso possa se dar de forma eficiente, é interessante ter em mente a necessidade de integração de uma gerência de conhecimento com os processos de negócio (O´LEARY et al., 2001).

Considerando, agora, que o foco deste trabalho é a Gerência de Conhecimento (GC) no contexto da Engenharia de Requisitos, o processo de negócio pode ser considerado o processo de software, com destaque para o sub-processo de Engenharia de Requisitos. Sendo assim, pode-se dizer que é interessante que a Gerência de Conhecimento esteja integrada aos processos de desenvolvimento e manutenção de software. Desse modo, é possível pensar nessa integração sob dois aspectos: (i) contemplar a Gerência de Conhecimento embutida nos processos organizacionais, os quais devem ser re-engenhados para acomodá-la (O´LEARY et al., 2001) e (ii) possuir um ferramental de forma que o conhecimento possa ser gerenciado enquanto os processos de negócio são conduzidos, buscando tornar a Gerência de Conhecimento o mais transparente possível, o que na visão de O´LEARY et al. (2001) ainda é um desafio.

Em relação ao primeiro aspecto, cabe a cada organização chegar em um nível de maturidade em que ela sinta necessidade de embutir a Gerência de Conhecimento como uma atividade dos seus processos de software e, a partir disso, alterar seus processos-padrão no sentido de contemplá-la. Uma prova do reconhecimento da importância da Gerência de Conhecimento nos processos de software é que a ISO/IEC 12207 em sua Emenda 1 (ISO, 2002), lista uma atividade de Gestão do Conhecimento como parte de seu Processo de Recursos Humanos.

O segundo aspecto surge como um apoio ao primeiro. Uma vez definido um processo de software que contemple a Gerência de Conhecimento, é interessante que se possua um aparato de apoio adequado para que os desenvolvedores possam conduzir o processo de software sem despender grande parcela de tempo gerenciando conhecimento. Pois se assim for, a Gerência de Conhecimento pode ser vista como o vilão de possíveis atrasos ao longo do ciclo de vida do processo. Novamente, usando como referência a ISO/IEC 12207 Emenda 1 (ISO, 2002), uma das tarefas da atividade de Gestão do Conhecimento consiste em definir: (i) uma infra-estrutura para apoiar quem contribui e quem utiliza os ativos de conhecimento da organização, (ii) um esquema de classificação dos ativos e (iii) critérios dos ativos de conhecimento.

Sendo assim, deve-se buscar um ferramental integrado que possa apoiar o desenvolvimento de software e que considere a Gerência de Conhecimento. Nessa visão de solução integrada, os Ambientes de Desenvolvimento de Software (ADS) são uma opção. Tais ambientes surgiram da necessidade de se integrar ferramentas CASE (*Computer-Aided Software Engineering*) que, embora automatizem várias atividades do processo de software (PRESMAN, 2002), possuem pouca ou nenhuma integração, dificultando, assim, que um produto do processo de desenvolvimento gerado por uma ferramenta possa ser utilizado por outra. Assim, um ADS, por sua filosofia, pode congregar ferramentas diversas e estruturá-las de modo que serviços de Gerência de Conhecimento possam estar disponíveis a qualquer momento para os membros da organização de software.

Assim, pelo exposto acima, durante o processo de software vários são os itens produzidos passíveis de reúso. Se esses itens forem produzidos e gerenciados de maneira a promover reúso, pode-se dizer que as organizações de software têm a possibilidade de tornar seus processos mais eficientes no sentido de diminuir custos, atender prazos e minimizar re-trabalho. No entanto, é fato que se o reúso não for sistematizado, ou seja, se não estiver incorporado aos processos da organização e se os produtos da Engenharia de Software não forem gerados visando ao reúso, os benefícios da reutilização não serão alcançados e os seus custos sempre serão elevados (PRIETO-DIAZ, 1990).

Este capítulo está estruturado da seguinte forma: a seção 3.2 trata dos conceitos gerais da Engenharia de Domínio, com foco na Análise de Domínio e como ela pode ajudar na captura do conhecimento; a seção 3.3 explicita o que vêm a ser ontologias e como podem ser aplicadas no contexto da Engenharia de Domínio; a seção 3.4 trata, inicialmente, dos padrões de software e depois foca nos padrões de análise, sobre os quais este trabalho tem mais

interesse; na seção 3.5 é tratada a Gerência de Conhecimento e seus serviços; na seção 3.6 discorre-se sobre Ambientes de Desenvolvimento de Software (ADS) e, em específico, sobre ODE (*Ontology-based Software Development Engineering*) (FALBO et al., 2003), que é o ADS no qual este trabalho está inserido; e, finalmente, na seção 3.7 são destacadas algumas considerações do capítulo.

3.2 Engenharia de Domínio

A Engenharia de Domínio representa um enfoque sistemático para a produção de componentes reutilizáveis que engloba as atividades de Análise, Projeto e Implementação de Domínio, as quais objetivam, respectivamente, representar requisitos comuns de uma família de aplicações por meio de modelos de domínio, disponibilizar modelos arquiteturais para cada aplicação a partir de um único modelo de domínio e disponibilizar implementações de componentes que representam funcionalidades básicas de aplicações relacionadas a um domínio (WERNER et al. 2005).

Segundo a ISO 12207 (ISO, 2002) a Engenharia de Domínio é considerada um processo que, além dessas atividades de produção de componentes reutilizáveis, aborda também atividades para a provisão e manutenção desses componentes. Ainda nessa visão de processo, a ISO 12207 considera que a Engenharia de Domínio pode ser executada junto com o processo de desenvolvimento de software, de maneira que componentes possam ser construídos, fornecidos ou mantidos enquanto produtos de software são gerados.

Das três atividades básicas da Engenharia de Domínio (Análise, Projeto e Implementação de Domínio) este trabalho foca na primeira delas, uma vez que objetiva o reúso no contexto da Engenharia de Requisitos e prega a reutilização de modelos de domínio a serem utilizados na construção de aplicações afins. Dessa maneira, partindo de uma visão global da Engenharia de Domínio, direciona-se o texto desta seção para a Análise de Domínio, a fim de embasar o processo de construção de modelos de domínio que serão, posteriormente, tratados como itens de conhecimento reutilizáveis.

3.2.1 Análise de Domínio

Como citado anteriormente, a Análise de Domínio é um processo no qual elementos relevantes de um domínio são identificados e disponibilizados para serem utilizados no

desenvolvimento de sistemas. Nesse sentido, a Análise de Domínio objetiva explicitar e formalizar aspectos de domínio para auxiliar os desenvolvedores na resolução de questões relacionadas a um domínio específico (Arango apud CIMA et al., 1997). A idéia central é identificar o que constitui o domínio, ou seja, o que pode e o que não pode ser considerado como parte dele (HENNINGER et al., 1995). Assim, tem-se como produto da Análise de Domínio modelos que contêm informações sobre um domínio e que podem ser reutilizados no desenvolvimento de sistemas (CIMA et al., 1997).

Para ilustrar melhor esse caráter de processo, pode-se utilizar o diagrama de contexto SADT mostrado da Figura 3.1. Por esse diagrama, pode-se observar que, ao se realizar a Análise de Domínio, utilizam-se recursos humanos (Analistas de Domínio, Especialistas e Engenheiro de Domínio), procedimentos (Métodos de Análise e Procedimentos de Gerência), insumos (Fontes de Conhecimento) e geram-se produtos (Taxonomias, Padrões, Modelos etc.). Outro fato que deve ser analisado é a retro-alimentação do processo, na qual produtos gerados pela Análise de Domínio são reavaliados, de acordo com sua utilização, e mantidos para que estejam sempre atualizados, uma vez que, em geral, os domínios de conhecimento são dinâmicos (PRIETO-DÍAZ, 1990).

Em toda sua extensão, a Análise de Domínio congrega três conceitos básicos que merecem destaque (ARANGO, 1994):

- **Domínio do problema:** representa um conjunto de itens de informação, em um certo contexto, inter-relacionados e que têm suas fronteiras delimitadas de acordo com o interesse e necessidade de um certo grupo de indivíduos.

- **Modelo do Domínio:** define entidades, operações, eventos e relações que estão presentes em um determinado domínio, estabelecendo uma visão unificada de componentes comuns às aplicações analisadas.

- **Análise e Modelagem do Domínio:** conjunto de atividades cujo propósito é auxiliar no entendimento e na captura de conhecimento de um domínio por meio de experimentos, técnicas, métodos etc.

Uma vez que busca auxiliar o trato com conhecimento, a Análise de Domínio é um processo importante no desenvolvimento de software e, portanto, é interessante que esteja integrado ao processo de desenvolvimento. PRIETO-DÍAZ (1990) destaca a equivalência entre a Análise de Domínio e a Análise de Requisitos, considerando, no entanto, um maior nível de abstração existente no contexto da Análise de Domínio. Logo, pode-se considerar que

a Análise de Domínio direciona a Engenharia de Requisitos, pois seus modelos de domínio, mais abstratos, fornecem base para o trato com requisitos (CIMA et al., 1997).

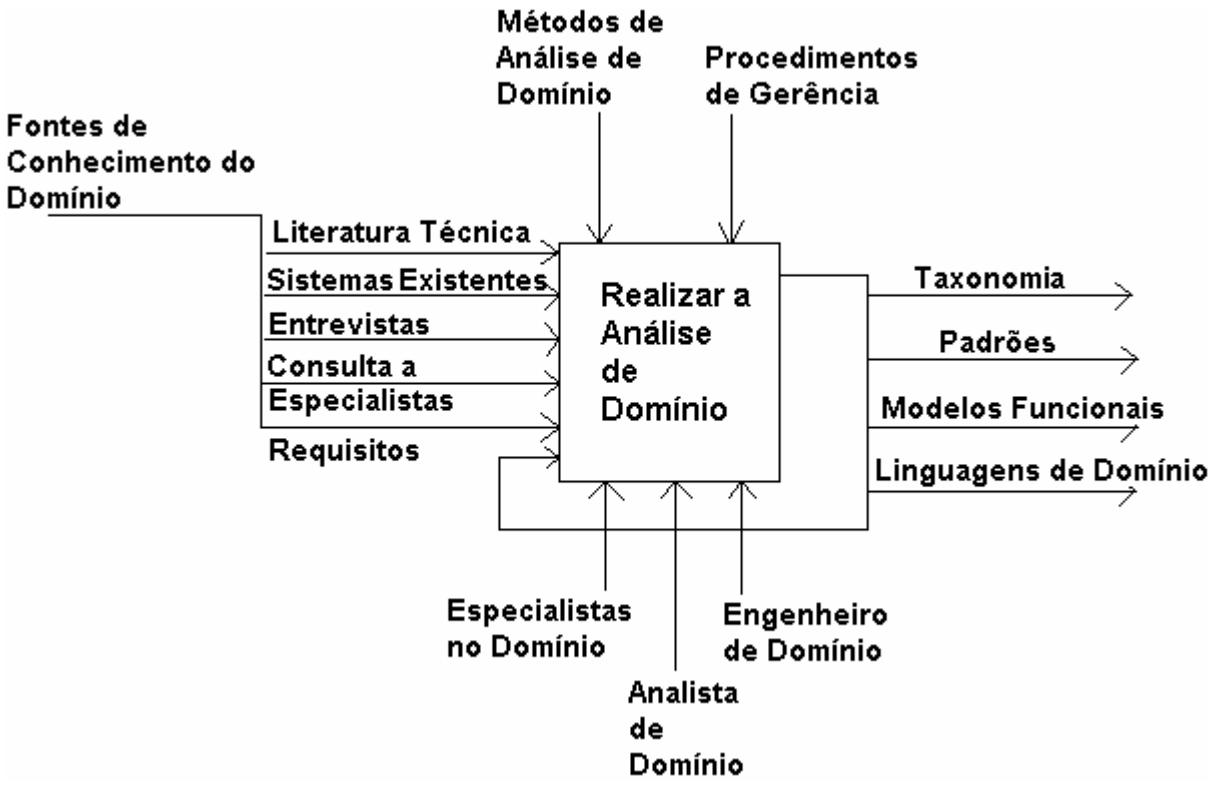


Figura 3.1 - O Processo de Análise de Domínio (Prieto-Díaz apud CIMA et al., 1997)

CIMA et al. (1997) destacam, ainda, que a Análise de Domínio pode ser considerada como uma atividade anterior ao Desenvolvimento de Software, uma vez que esse se utiliza dos modelos daquela. Ademais, faz-se necessário destacar que durante os processos de Desenvolvimento de Software, os artefatos da Análise de Domínio estarão constantemente sendo utilizados, avaliados e evoluídos.

Moore e Bailin (apud CIMA et al., 1997) entendem que a Análise de Domínio é complementar ao processo de desenvolvimento, o que conduz a duas perspectivas: (i) a perspectiva do desenvolvedor de sistemas, que utiliza os itens de conhecimento disponíveis e (2) a perspectiva do gerente de conhecimento, que é aquele que fornece e mantém o conhecimento a ser reutilizado. Esses dois atores cumprem papéis interdependentes, pois o desenvolvedor de sistemas necessita de conhecimento para cumprir suas tarefas mais eficientemente e, por outro lado, o gerente de conhecimento necessita do *feedback* do desenvolvedor para que ele possa manter os itens de conhecimento o mais atualizados e úteis quanto possível.

3.3 Ontologias

Ontologias, na corrente discussão da Ciência da Computação, são recursos baseados em computador que representam semânticas compartilhadas sobre domínios (SPIN et al., 2002). Essa definição é importante, no contexto deste trabalho, por três aspectos: (1) contextualiza ontologias na área da Ciência da Computação; (2) considera ontologias como recursos baseados em computador; e (3) destaca o fato de ontologias tratarem semânticas acordadas sobre domínios, o que reforça a utilização de ontologias para a captura de conhecimento compartilhado acerca de um domínio.

Ainda que se tenha falado acima de ontologia de uma maneira mais específica, voltada para o conhecimento de domínio, pode-se destacar pelo menos quatro tipos de ontologias, segundo o que elas se propõem a modelar (GUARINO, 1998):

- **Ontologias Genéricas:** descrevem conceitos gerais, como espaço, tempo, problema, objeto, evento, ação etc.
- **Ontologias de Domínio:** descrevem o vocabulário relacionado a um domínio, como, por exemplo, medicina, direito etc.
- **Ontologias de Tarefa:** descrevem o vocabulário relacionado a uma tarefa genérica, como, por exemplo, diagnose, venda etc.
- **Ontologias de Aplicação:** descrevem conceitos dependentes de um domínio e de uma tarefa particulares, os quais são, freqüentemente, especializações de ontologias relacionadas.

A Figura 3.2 ilustra os relacionamentos de especialização entre esses quatro tipos de ontologias, sendo que no topo estão as ontologias genéricas que definem conceitos gerais que embasam todas as outras conceituações e no nível mais baixo estão as ontologias de aplicação, que especializam conceituações para um determinado domínio de aplicação.

Uma vez que por meio de ontologias pode-se modelar conhecimento, elas possuem muitas aplicações práticas. Segundo DEVEDZIC (1999), ontologias podem ser utilizadas para integração de bases de conhecimento, uma vez que provêem um esqueleto de conhecimento e uma infra-estrutura independente de uma implementação particular. STAAB et al. (2001), por sua vez, destacam que uma ontologia pode ser usada para derivar visões de conhecimento.

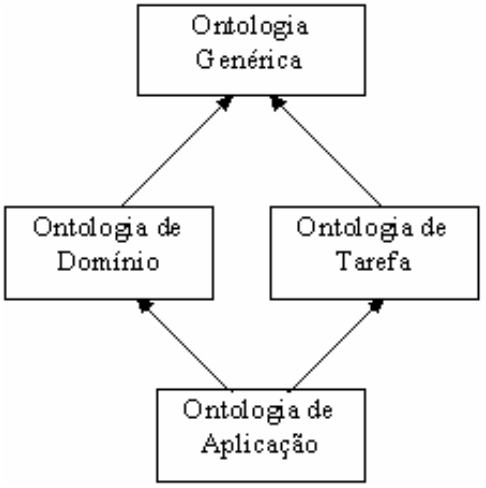


Figura 3.2 - Relacionamentos entre os tipos de ontologias segundo (GUARINO, 1998).

GUARINO (1998) prega a utilização de ontologias no desenvolvimento de sistemas de informação (SI) sob dois aspectos:

1) **Ontologias utilizadas em tempo de execução ("SI guiados por ontologia"):** fornece a capacidade de comunicação entre agentes de software, uma vez que podem utilizar conceituações compartilhadas e formalizadas em ontologias;

2) **Ontologias utilizadas em tempo de desenvolvimento ("desenvolvimento de SI guiado por ontologia"):** neste caso tem-se um conjunto de ontologias reusáveis à disposição, organizadas em uma biblioteca contendo ontologias de domínio e de tarefa. Isso permite reduzir o custo da análise conceitual, possibilita o desenvolvedor a praticar um reúso de alto nível e a compartilhar conhecimento de domínio de aplicação usando um vocabulário comum em torno de uma plataforma de software heterogênea.

Uschold et al. (apud ZLOT, 2002) destacam, em relação ao desenvolvimento de software, que ontologias trazem algumas vantagens, principalmente no que se refere a:

- **Comunicação entre pessoas:** ontologias reduzem os conflitos conceituais e terminológicos dentro da organização, uma vez que provêm uma conceituação unificada para a mesma;
- **Interoperabilidade entre sistemas:** ontologias podem ser utilizadas na tradução de diferentes representações de bases de dados, fornecendo uma conceituação única a partir da qual outras conceituações se normalizam.

- **Apoio à Engenharia de Software:** ontologias auxiliam na especificação, uma vez que promovem um entendimento compartilhado acerca de um domínio ou tarefa, e na reutilização durante a engenharia de software, uma vez que o conhecimento capturado pode ser aplicado em outras situações.

Em relação ao reúso de ontologias no contexto da Engenharia de Software, GUIZZARDI (2000) desenvolveu um trabalho que foca o Desenvolvimento *para* e *com* Reúso. Dessa maneira, ao se estabelecer um processo de desenvolvimento *para* reúso, questões relacionadas com a definição de um processo para contemplar a Análise de Domínio e com a representação dos produtos dessa atividade foram tratadas, tendo-se utilizado ontologias de domínio como modelos de domínio. Além disso, Guizzardi destacou a necessidade de, no Desenvolvimento *com* reúso, ser possível, de posse de ontologias de domínio, aplicar tais modelos no processo de software. Sendo assim, ele criou uma abordagem para mapear o conhecimento documentado em ontologias para estruturas computacionais de reúso (*frameworks*), compostos de classes, métodos, atributos. Em linhas gerais, conceitos são mapeados para classes, relações para associações, propriedades para atributos e axiomas para métodos.

Tal trabalho, portanto, está relacionado à proposta desta dissertação, uma vez que prega o reúso de conhecimento para apoiar a Engenharia de Software, conhecimento esse que está, por sua vez, documentado em ontologias de domínio. Ademais, a idéia de conversão de ontologias também é muito útil para este trabalho, pois permite pensar na derivação de ontologias em modelos de níveis mais baixos de abstração, como, por exemplo, modelos de objetos de análise, que também documentam conhecimento e podem ser utilizados em atividades da Engenharia de Requisitos. Essa abordagem também está de acordo com a idéia de GUARINO (1998) de utilizar ontologias durante o desenvolvimento de sistemas.

De uma maneira geral, ainda que existam muitas vantagens no uso de ontologias, algumas questões carecem de estudo e esclarecimento, como destaca O'Leary (apud GUIZZARDI, 2000): (i) como o consenso de muitas partes é dificultado, a definição de uma conceituação compartilhada em uma ontologia é um processo político no sentido de que algumas pessoas mais significativas em um determinado contexto terão maior poder de voto na conceituação resultante; (ii) ontologias não são necessariamente estacionárias e a evolução de ontologias é uma tarefa complexa dada sua formalização e forte relação entre suas partes. Além disso, a integração entre ontologias desenvolvidas de forma independente não é uma tarefa trivial, uma vez que podem estar baseadas em conceituações distintas (FALBO, 1998).

3.4 Padrões de Software

Padrões podem ser vistos como experiências reutilizáveis, uma vez que documentam problemas contextualizados, recorrentes e comuns, que desenvolvedores de software têm enfrentado e encontrado uma solução bem sucedida (BARROCA et al., 2005). No contexto da Engenharia de Software encontram-se os padrões de software, os quais têm sido cada vez mais utilizados com o objetivo de aumentar a reutilização e a qualidade dos projetos, pois, por meio desses padrões, consegue-se refinar o conhecimento e a experiência de muitos desenvolvedores (COTA, 2002).

Pode-se dizer que um padrão é uma combinação de unidades significativas que ocorrem em determinado contexto. O vocabulário oferecido pelos padrões ajuda a tornar claro o pensamento, além de aumentar o nível de abstração, possibilitando a discussão entre especialistas e novatos, sendo uma unidade transferível de conhecimento especializado (COTA et al., 2004).

Os padrões de software podem ser classificados em categorias mais específicas, de acordo, por exemplo, com as atividades do processo de desenvolvimento em que são mais utilizados. Assim, tem-se, por exemplo, os padrões de análise (FOWLER, 1997) e os padrões de projeto (GAMMA et al., 1995). Um padrão de análise, segundo FOWLER (1997), é um grupo de conceitos que representa uma construção comum na modelagem de negócio. Para GAMMA et al. (1995), um padrão de projeto é a "descrição de classes e objetos comunicativos, que são adaptados para resolver um problema geral de projeto em um contexto particular".

Um aspecto importante a ser considerado nos padrões é a simplicidade. Para que eles possam ser facilmente reutilizados em outras situações, é interessante que eles contenham o que for comum ao domínio que se propõem a modelar, caso contrário, o padrão conterá aspectos específicos de uma aplicação o que dificultará seu reúso em outras situações (COTA, 2002). Ademais, considerando soluções mais simples documentadas nos padrões, torna-se mais fácil a composição entre eles para que se alcance soluções mais elaboradas.

Considerando a utilização de padrões no desenvolvimento de sistemas podem-se destacar duas abordagens (PALUDO et al., 2005):

- **Dirigida a exemplos:** considera a aplicação de padrões a partir de um catálogo ou repositório, no qual os padrões estão organizados de maneira a facilitar sua

localização e conseqüente utilização. Nessa categoria estão os padrões de projeto e os padrões de análise.

- **Dirigida a processos:** apresenta passos a serem seguidos para a concepção e elaboração de sistemas, utilizando os padrões conforme predeterminado no processo. Neste caso, o processo é o principal elemento, aquele que conduz a utilizações de outros padrões.

Considerando a Engenharia de Requisitos, um dos padrões que mais se aplica a esse processo são, por definição, os Padrões de Análise. Eles são construídos a partir de modelos conceituais de aplicações, que descrevem modelos de processos de negócios resultantes da fase de análise de requisitos (COTA et al., 2004). GEYER-SCHULZ et al. (2001) destacam que esse tipo de padrão foca em questões organizacionais, sociais e econômicas, uma vez que são aspectos centrais a serem considerados durante a análise de requisitos. Além disso, a utilização de padrões logo nas atividades iniciais do processo de desenvolvimento contribui para a construção de arquiteturas mais flexíveis e reutilizáveis (FOWLER, 1997).

Padrões de Análise, assim como ontologias, descrevem aspectos no nível de conhecimento. Eles provêem conhecimento sobre soluções bem sucedidas a problemas recorrentes no desenvolvimento de software (DEVEDZIC, 1999), o que favorece o reúso (COTA et al., 2004). No entanto, ontologias capturam a estrutura conceitual intrínseca de um domínio, enquanto padrões de análise capturam a estrutura conceitual intrínseca de aplicação (DEVEDZIC, 1999).

Usar ontologias e padrões de análise para a modelagem conceitual de sistemas abre espaço para uma visão mais abrangente do domínio tratado, levando a um melhor entendimento desse domínio. Isso pode diminuir o tempo gasto na especificação de requisitos, pois os engenheiros de software já têm uma fonte preliminar para aprender sobre o domínio, que estabelece uma terminologia comum aos especialistas do negócio (COTA et al., 2004).

3.5 Gerência de Conhecimento

Um dos interesses de usar a Gerência de Conhecimento (GC) na Engenharia de Software é decorrente da constante expectativa de aumento da produtividade por meio da reutilização.

O desenvolvimento baseado em reúso considera que esforços investidos e experiências adquiridas em projetos anteriores podem ser reutilizados em novos projetos. Dessa maneira, é interessante que o conhecimento gerado seja armazenado de uma forma que se torne fácil a sua recuperação e utilização. Essa utilização, por sua vez, propicia a expansão do conhecimento adquirido, uma vez que o conhecimento humano e, conseqüentemente, o conhecimento organizacional são criados e expandidos por meio da interação contínua e social entre conhecimento tácito e conhecimento explícito (NONAKA et al., 1997). O conhecimento tácito é pessoal e relacionado a um contexto, sendo difícil de ser formulado e comunicado. Já o conhecimento explícito é o conhecimento que pode ser comunicado por meio da linguagem natural, pelo fato de estar codificado (NONAKA et al., 1997).

De acordo com DAVENPORT et al. (1998), para que as organizações possam tirar proveito da GC, ela deve estar voltada para a solução das seguintes questões:

- Como as organizações podem tirar maior proveito do conhecimento existente dentro delas?
- Como membros da organização podem distribuir o conhecimento para quem este pode ser útil?
- Como registrar as soluções adequadas para tratar problemas?
- Como reter o conhecimento de seus especialistas, mesmo quando estes deixam a organização?
- Como gerar conhecimento novo a partir do conhecimento existente dentro da organização ou a partir de fontes externas?

No entanto, gerenciar conhecimento não é trivial. Deve-se, assim, estabelecer um plano, em médio prazo, de como trabalhar as questões mais relevantes para o contexto de uma determinada organização e amadurecer os aspectos de GC na medida em que toda a organização absorva a nova filosofia de trabalho. Uma maneira de atacar a implantação da Gerência de Conhecimento é a utilização de um processo. Segundo BENJAMINS et al. (1998), a própria Gerência de Conhecimento é um processo que congrega relacionamentos humanos, práticas de negócios e tecnologia da informação (TI). Ou seja, nessa definição vê-se uma Gerência de Conhecimento institucionalizada, pois são tratados seus três elementos básicos: pessoas, negócio e tecnologia.

Um processo de GC é composto por várias atividades relacionadas entre si (NATALI, 2003) e a definição dessas atividades varia de autor para autor, de acordo com sua interpretação das fases do processo. STAAB et al. (2001) propõem as seguintes atividades: Criação ou Importação, Captura, Recuperação e Acesso e Uso; já Fischer et al. (apud NATALI, 2003) citam as atividades de Criação, Integração e Disseminação. No entanto, independentemente das atividades presentes no processo de GC, é imprescindível a automatização desse processo por meio da utilização de tecnologias de informação. Um primeiro passo nesse sentido é pensar um sistema de GC como sendo composto de uma memória organizacional e serviços para apoiar as atividades do processo de GC.

Segundo Heijst et al. e Dieng et al. (apud NATALI, 2003), para sistematizar a Gerência de Conhecimento, é interessante que haja serviços atuando em torno de uma Memória Organizacional, a qual é uma representação explícita e persistente do conhecimento e das informações cruciais para uma organização. A finalidade é facilitar o acesso, compartilhamento e reúso de conhecimento pelos diversos membros da organização. Cada serviço, especializado com sua função e por ferramentas computacionais, tem a possibilidade de dar maior eficiência ao Processo de Gerência de Conhecimento. Esses serviços visam a disponibilizar o conhecimento adequado à pessoa certa, a fim de possibilitar a ação devida. Dessa maneira, como exemplos de sistemas que objetivam especializar e implementar, pelo menos parcialmente, sistemas de Gerência de Conhecimento, podem ser citados (RUS et al., 2002):

- **Sistemas de Informação de Memória Organizacional**: visam a apoiar a criação de uma memória organizacional, armazenando documentos e outros artefatos utilizados ou gerados durante o andamento do trabalho na organização. Nessa categoria, podem ser incluídas também as ferramentas de controle de versão que, apesar de não terem a criação de uma memória organizacional como um de seus objetivos, indiretamente apóiam a sua criação.
- **Sistemas de Gerência de Competências:** têm o intuito de possibilitar a localização de membros da organização com as habilidades necessárias para a formação de equipes qualificadas para novos projetos.
- **Sistemas de Gerência de Documentos**: realizam o controle computadorizado de documentos, permitindo o seu armazenamento, em um banco de dados, e pesquisas para a sua rápida localização, sendo, portanto, a base para os Sistemas

de Gerência de Conhecimento, os quais tipicamente precisam lidar com muitos documentos.

De uma maneira geral, uma das grandes preocupações de muitas empresas é a fragilidade de se ter o conhecimento acerca dos variados processos que regem a companhia restritos a um número limitado de funcionários. Tal fragilidade é visível quando pensamos que um profissional experiente está à mercê de várias casualidades, como a mudança de empresa, levando consigo toda a experiência acumulada. Assim, se o conhecimento for capturado e gerenciado adequadamente, a possibilidade de ser útil em um novo contexto de solução de problemas cresce (TOGNERI, 2002). Dessa forma, a experiência adquirida na solução de um problema passa a estar disponível para todos os membros da organização. Isso, no contexto de organizações de software, pode diminuir riscos durante o desenvolvimento de sistemas.

3.6 Ambientes de Desenvolvimento de Software

Um Ambiente de Desenvolvimento de Software (ADS) é um conjunto integrado de ferramentas que apóia a realização de atividades de engenharia de software durante todo o processo de software ou pelo menos em porções significativas dele (HARRISON et al., 2000).

A integração de ferramentas é uma questão fundamental para ADSs. Quando se fala em integração em ADS, deve-se definir até que ponto ela é estabelecida. Tomando por base (PFLEEGER, 2001), (TRAVASSOS, 1994) e (FALBO, 1998), podem-se citar cinco níveis de integração:

- **Integração de Dados:** tem o objetivo de compartilhar informações consistentes entre as ferramentas ao longo do desenvolvimento.
- **Integração de Apresentação:** tem o objetivo de criar uma uniformidade entre as interfaces do ambiente, aumentando a sua usabilidade, de modo a permitir que o usuário das ferramentas possa alternar entre elas sem ter que se adaptar a novas interfaces, favorecendo a utilização das mesmas.
- **Integração de Controle:** tem o objetivo de estabelecer uma integração flexível das funcionalidades do ambiente e de suas ferramentas, por meio do compartilhamento de funções ou serviços.

- **Integração de Processo:** tem o objetivo de assegurar que ferramentas interajam de maneira efetiva, tendo como base um processo definido. Para tal, uma ligação explícita entre as ferramentas e o processo de software seguido pelo ambiente deve ser estabelecida.
- **Integração de Conhecimento:** buscando uma concepção semântica do funcionamento integrado das ferramentas, faz-se necessário modelar e disponibilizar o conhecimento para todas as ferramentas do ambiente, com o objetivo de que essas possam trabalhar de modo mais consistente. Serviços de gerência de conhecimento, permitindo capturar conhecimento durante os projetos de software e oferecendo apoio baseado em conhecimento aos engenheiros de software durante a realização de atividades do processo, devem ser providos.

Como benefícios dessa integração em ADSs, podem ser citados a transferência constante de informação entre ferramentas, a automatização de atividades que outrora demandavam esforços consideráveis, tais como a gerência de configuração e a produção de documentos, o maior controle do projeto, a maior coordenação entre os membros da equipe de desenvolvimento e o maior controle na utilização das ferramentas do ADS (PRESSMAN, 2002).

De maneira geral, os ADSs, pelo seu caráter de integração, se apresentam como uma opção que permite uma ligação forte entre ferramentas de apoio à Engenharia de Software e ferramentas que abordam serviços de Gerência de Conhecimento. Pois como já citado na seção de introdução, para que a Gerência de Conhecimento seja efetivamente utilizada, ela deve estar integrada à condução do processo de negócio (O´LEARY et al., 2001). Uma vez que os ADSs visam a apoiar a execução do processo de software, que nesse caso pode ser considerado o processo de negócio de uma organização de software, se os serviços de Gerência de Conhecimento estiverem integrados às ferramentas do ambiente, eles estarão mais acessíveis e poderão, portanto, ser utilizados à medida que o desenvolvedor constrói um produto de software.

3.6.1 O Ambiente ODE

ODE (*Ontology-based Software Development Environment*) (FALBO et al., 2003) é um ADS centrado em processo, que tem sua fundamentação baseada em ontologias. Com base nessa fundamentação, as ferramentas de ODE são construídas, pois se acredita que a integração pode ser facilitada pelo fato dos conceitos envolvidos estarem bem definidos pelas ontologias.

ODE possui diversas ferramentas integradas, dentre elas, ferramentas de apoio à: definição de processos de software (BERTOLLO et al., 2006), gerência de riscos (GeRis) (FALBO et. al, 2004a), documentação (XMLDoc) (NUNES et. al, 2004), gerência de configuração (NUNES, 2005) e estimativas (EstimaODE) (CARVALHO et al., 2006).

Essas ferramentas são baseadas nas seguintes ontologias: Ontologia de Processo de Software (BERTOLLO, 2006) – que é a ontologia base do ambiente, Ontologia de Artefato de Software (NUNES, 2005), Ontologia de Gerência de Configuração de Software (NUNES 2005), Ontologia de Qualidade (DUARTE, 2001), Ontologia de Organizações de Software (RUY, 2006) e Ontologia de Requisitos de Software (NARDI et al., 2006).

A partir do momento que se constrói uma ontologia a ser utilizada em ODE, deve-se incorporá-la ao ambiente segundo sua arquitetura conceitual, a qual abrange desde aspectos de nível ontológico até aspectos de nível de aplicação e como eles se comunicam.

Além de sua fundamentação ontológica, ODE possui uma infra-estrutura de Gerência de Conhecimento, com base na qual serviços são desenvolvidos. Esses serviços são também construídos segundo a arquitetura conceitual do ambiente, de forma a utilizar a base ontológica estabelecida.

3.6.1.1 A Arquitetura Conceitual de ODE

O ambiente ODE possui uma arquitetura conceitual que favorece a correlação entre elementos desde o nível ontológico até o nível de aplicação, ou seja, onde as ferramentas são, de fato, implementadas. A Figura 3.3 exibe essa arquitetura conceitual abordando seus três níveis (FALBO et al., 2004b) (RUY, 2006):

Figura 3.3 - Arquitetura Conceitual de ODE (FALBO et al., 2004b)

- O Nível Ontológico é responsável pela descrição das ontologias segundo a meta-ontologia adotada pelo ambiente. Portanto, esse nível compreende as ontologias de ODE que são instâncias da meta-ontologia.
- O Meta-Nível abriga as classes que descrevem o conhecimento em relação a um domínio de aplicação. Suas classes são derivadas das ontologias e as instâncias dessas classes atuam no papel de conhecimento sobre os objetos do nível base.
- O Nível Base abriga as classes implementadas no contexto das aplicações de ODE. Essas classes são também derivadas das ontologias, mas tipicamente incorporam detalhes não descritos por elas, necessários para implementar as aplicações do ambiente.

Para se incorporar uma ontologia à estrutura do ambiente, primeiramente os elementos dessa ontologia (conceitos, relações, propriedades e restrições) devem ser definidos segundo o meta-modelo da Figura 3.4. Ou seja, esses elementos serão definidos como instâncias de classes do meta-modelo, constituindo assim, o Nível Ontológico. Vale destacar que o modelo da meta-ontologia de ODE é aderente ao meta-modelo da UML (SOUZA, 2004).

Após isso, utiliza-se parcialmente a abordagem de derivação proposta em (FALBO et al. 2002) que considera o mapeamento de conceitos e relações em classes e associações, propriedades de conceitos e relações em atributos de classes e axiomas em métodos. Vale ressaltar que, durante esse processo de derivação, dois modelos de objetos são criados: um será derivado para o Meta-Nível (nível de Conhecimento) e o outro será derivado para o Nível Base.

Nessa derivação, pode-se criar uma classe somente no nível de conhecimento, somente no nível base, em ambos os níveis ou mesmo em nenhum deles, de acordo com as seguintes situações (FALBO et al., 2004b) (RUY, 2006):

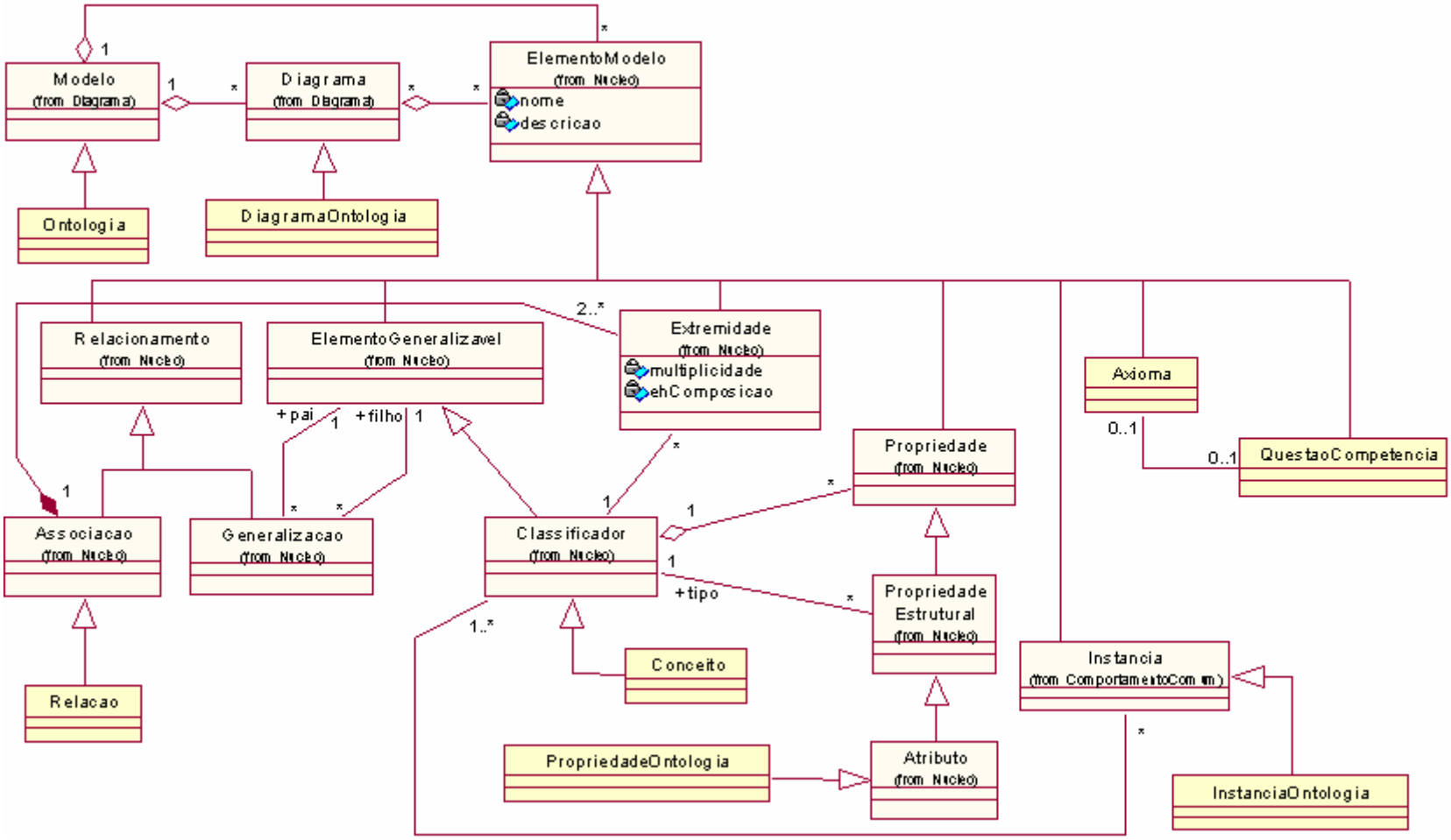


Figura 3.4 - Meta-Ontologia de ODE (SOUZA, 2004)

- **Sem Criação de Classe:** para alguns conceitos, podem não ser criadas classes correspondentes pelo fato deles representarem algo muito particular da ontologia e que não vai agregar valor no contexto do ambiente.
- **Criação de Classes tanto no nível de Conhecimento quanto no nível de Controle:** alguns conceitos são representados tanto no Conhecimento quanto no Controle pelo fato de haver a necessidade de modelar dois aspectos: (1) o tipo do objeto (Conhecimento) e (2) o próprio objeto (Controle). Um exemplo típico é o caso do conceito *FerramentaSofware*. Quando ele for derivado para a Arquitetura Conceitual haverá uma classe no Conhecimento – *KFerramentaSoftware* – que representa os tipos de ferramentas de software (Ferramenta de Apoio à Gerência de Riscos, Ferramenta de Apoio a Estimativas etc.) e outra no Controle (*FerrametaSoftware*), a qual representa uma ferramenta de software específica (GeRis, EstimaODE etc). Além disso, para que os dois níveis estejam integrados, a classe do Controle tem uma referência para a classe do Conhecimento. Dessa forma, é possível conhecer, a partir de um objeto do Controle, qual o seu tipo (Conhecimento).
- **Criação de Classe somente no nível de Conhecimento ou no nível de Controle:** alguns conceitos não figuram em ambos os níveis. Ou eles caracterizam um conhecimento por si mesmo ou um objeto do Controle que não necessita de conhecimento para determiná-lo. No primeiro caso tem-se, como exemplo, o conceito *Procedimento*. Esse conceito dá origem apenas a uma classe no Conhecimento (*KProcedimento*) que é utilizada para representar os tipos de procedimentos utilizados pela organização de software, o que, tipicamente, constitui um conhecimento. Assim, durante a definição de um processo, por exemplo, quando se definem quais são as atividades do processo, pode-se relacionar essas atividades a procedimentos (ou tipos de procedimento) utilizados na realização das mesmas. Um exemplo de ocorrência do segundo caso é o conceito *Projeto*. Neste caso, apenas uma classe *Projeto* é derivada no nível base, sendo que instâncias dessa classe representam projetos concretos.

Pela Figura 3.3 pode-se perceber que os níveis de ODE mantêm dependências entre eles. O Nível Base depende do Meta-Nível que depende do Nível Ontológico. Ao se derivarem classes para os níveis de Controle e de Conhecimento, essas dependências devem ser observadas. Assim, objetos do Nível de Controle, geralmente, possuem associações com

objetos do Nível de Conhecimento. No entanto, o contrário nunca é verdade. Quando apenas uma classe é gerada no nível base, a amarração com o nível ontológico é feita diretamente. De fato, há duas formas de associar os objetos pertencentes aos três níveis (RUY, 2006):

- as classes do nível base possuem uma associação com sua correspondente no nível de conhecimento, associando os objetos de aplicação aos seus "tipos" no nível de conhecimento;
- a classe Conceito (Figura 3.4) provê dois atributos para que suas instâncias possam endereçar as classes correspondentes nos níveis de conhecimento e controle. Esses atributos são do tipo Class (meta-classe de Java).

3.6.1.2 A Infra-Estrutura de Gerência de Conhecimento de ODE

O ambiente ODE conta com uma infra-estrutura de Gerência de Conhecimento (NATALI et al., 2003) (FALBO et al., 2004c), composta de uma memória organizacional e cinco tipos básicos de serviços. A estratégia adotada é a codificação do conhecimento na memória organizacional do ambiente para que, quando necessário, os desenvolvedores possam acessá-la em busca de itens de conhecimento relevantes. A Figura 3.5 fornece uma visão geral da organização dessa infra-estrutura e a seguir cada um dos tipos de serviço é brevemente descrito.

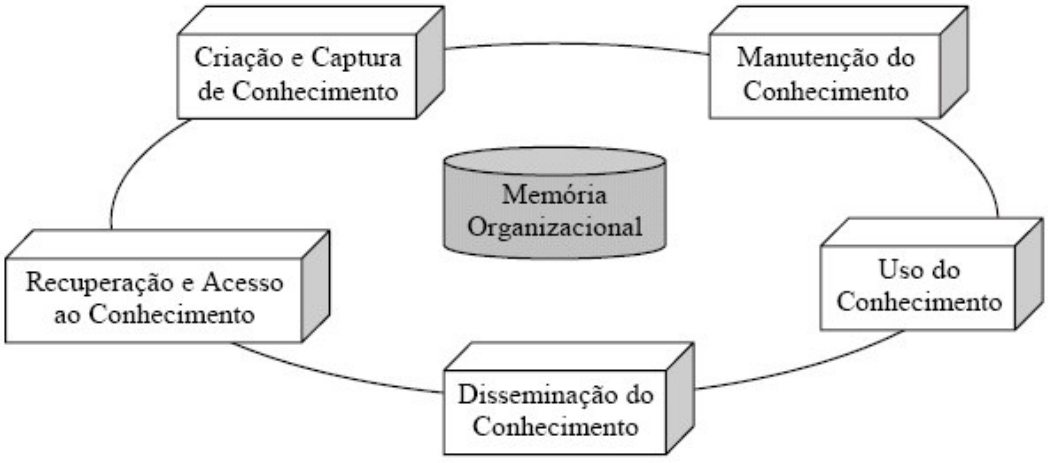


Figura 3.5 – Sistema de Gerência de Conhecimento (NATALI, 2003).

- **Criação e Captura:** permite a captura e criação de itens de conhecimento que podem ser artefatos, lições aprendidas, pacotes de mensagens trocadas por membros da organização etc.
- **Recuperação e Acesso:** apóia a busca do usuário por itens de conhecimento. Assim, é o usuário que parte em busca de itens de conhecimento.
- **Disseminação pró-ativa:** é um serviço iniciado pelo sistema, o qual fornece, baseado nas necessidades do usuário, itens de conhecimento relevantes para uma determinada tarefa que o mesmo está executando.
- **Uso:** dá suporte ao momento em que o usuário utiliza um item de conhecimento. Na utilização, o usuário pode dar informações acerca da eficácia de um determinado item de conhecimento em uma dada situação e até avaliar o item de conhecimento. Essa avaliação pode ser utilizada na manutenção do repositório de conhecimento.
- **Manutenção:** dá apoio à manutenção dos itens de conhecimento, possibilitando ao gerente de conhecimento atualizar a memória organizacional, bem como efetuar algumas correções nos itens de conhecimento.

Além dessa classificação, os serviços de GC são categorizados de duas formas (NATALI et al., 2003): (1) serviços gerais e (2) serviços específicos. Os serviços gerais compreendem a captura, recuperação, uso e manutenção de conhecimento. Esses serviços estão disponíveis no ambiente, da mesma forma, para todas as ferramentas. Ou seja, são independentes de características particulares de cada ferramenta. Os serviços específicos estão ligados diretamente à disseminação de conhecimento. Por isso eles são especializados de acordo com as características de cada ferramenta e do perfil dos usuários.

Cada serviço pode ser implementado de diferentes maneiras. Por exemplo, um serviço de empacotamento de mensagens é responsável por criar pacotes de mensagens trocadas utilizando as ferramentas de *groupware*, permitindo capturar o conhecimento proveniente das interações entre membros de uma organização de software (ARANTES et al., 2004).

O modelo de classes relacionado aos conceitos da infra-estrutura de GC de ODE é apresentado na Figura 3.6.

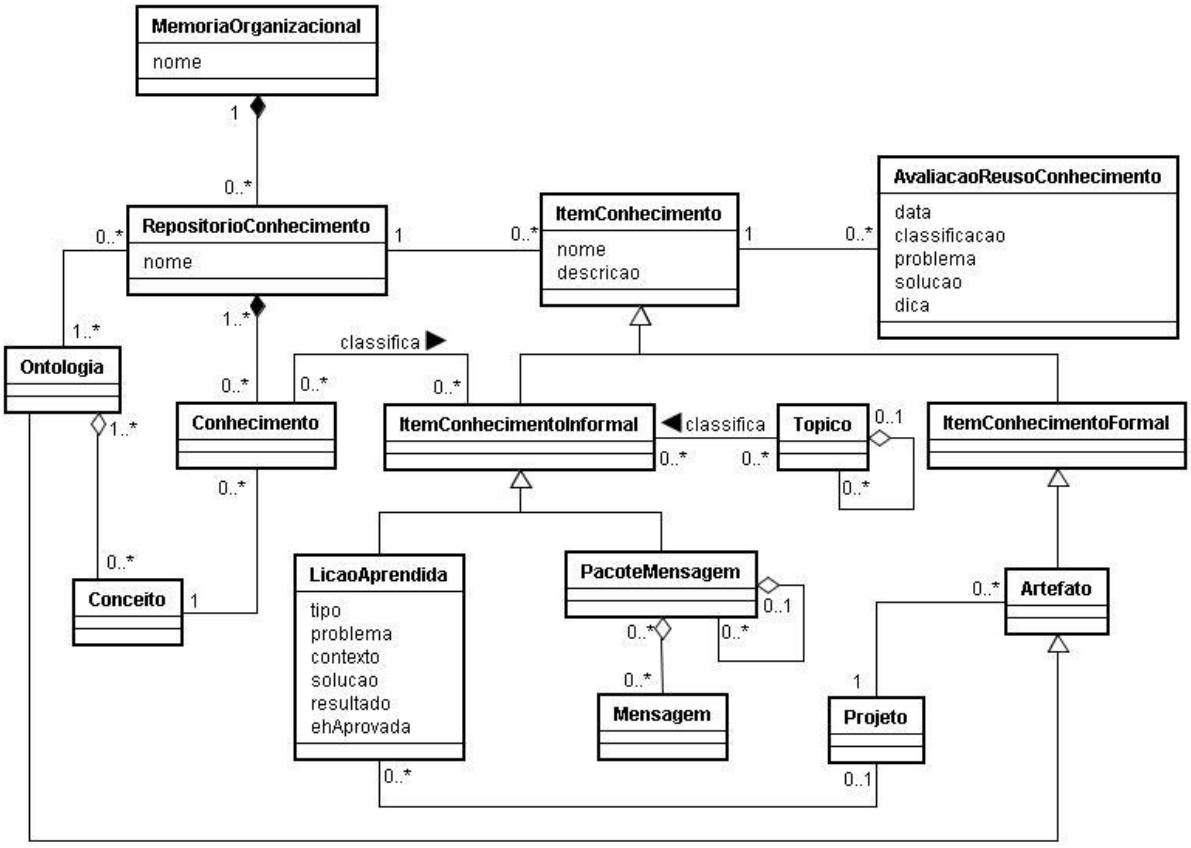


Figura 3.6 - A Infra-estrutura de Gerência de Conhecimento de ODE.

Pelo diagrama da Figura 3.6 pode-se perceber que a *MemoriaOrganizacional* é composta por vários repositórios de conhecimento (objetos da classe *RepositorioConhecimento*) que, por sua vez, contêm itens de conhecimento (instâncias de *ItemConhecimento*) e conhecimento organizacional (objetos da classe *Conhecimento*) cadastrado no ambiente como, por exemplo, os tipos de requisitos utilizados pela organização na condução de seus projetos.

Um *ItemConhecimento* pode ser formal (*ItemConhecimentoFormal*), como no caso de *Artefatos*, ou informal (*ItemConhecimentoInformal*), como no caso de instâncias das classes *LicaoAprendida* e *PacoteMensagem* (ARANTES et al., 2004). Esses itens de conhecimento podem ser produzidos no contexto de um *Projeto* e podem ser reutilizados quando necessário.

Visando a facilitar a busca de itens de conhecimento informais, eles são organizados e classificados por meio de instâncias das classes *Topico e Conhecimento*. Objetivando a manutenção desses itens, recursos humanos da organização podem avaliá-los quanto à utilidade para a tarefa que estão realizando, fornecendo, assim, ao Gerente de Conhecimento,

parâmetros acerca da relevância do reúso desses itens nos projetos de software. Tais avaliações são representadas por meio de instâncias da classe *AvaliacaoReusoConhecimento.*

Outro ponto importante a ser destacado é que as ontologias existentes em ODE são modeladas como artefatos e, por conseguinte, itens de conhecimento. Logo, poderiam ser reutilizadas como tais. No entanto, as ontologias de ODE são utilizadas, atualmente, apenas como modelos de estruturação da base de conhecimento do ambiente.

Por fim, sobre a estrutura de classes da Figura 3.6 atuam os serviços de Gerência de Conhecimento no sentido de apoiar os desenvolvedores na execução de suas tarefas diárias. Tal infra-estrutura foi construída em ODE motivada por estudos sobre Gerência de Conhecimento, que têm mostrado que se ela for aplicada ao desenvolvimento de software de forma a apoiar suas atividades, muito se tem a ganhar em termos de resolução de problemas (FALBO et al., 2004c).

3.7 Considerações do Capítulo

Neste capítulo, foram tratadas questões relacionadas ao reúso no contexto do processo de software e fornecidos fundamentos teóricos acerca de Análise de Domínio, Ontologias, Padrões de Análise, Gerência de Conhecimento e Ambientes de Desenvolvimento de Software.

Assim, a partir da fundamentação teórica deste capítulo e do anterior, o qual trata da Engenharia de Requisitos, objetiva-se, no Capítulo 5, expor uma abordagem prática de reúso de itens de conhecimento no contexto da Engenharia de Requisitos. Como este trabalho está no âmbito de um ADS que possui uma infra-estrutura de Gerência de Conhecimento, todos esses aspectos teóricos serão relacionados na constituição de um apoio ao reúso no contexto da Engenharia de Requisitos: proposta desta dissertação.

Capítulo 4

Uma Ontologia de Requisitos de Software

O objetivo deste capítulo é apresentar a Ontologia de Requisitos de Software construída no contexto deste trabalho, seguindo a abordagem proposta pelo método SABiO. Sendo assim, esse método é brevemente descrito, juntamente com suas atividades. Após isso, a ontologia de requisitos é apresentada, começando pelas questões de competência, passando pela integração com ontologias existentes e chegando à captura e formalização da mesma.

4.1 Introdução

Considerando que os requisitos são o foco do processo da Engenharia de Requisitos (ER), é interessante que se tenha uma compreensão clara do que vem a ser um requisito e de como ele se relaciona com outros elementos do processo de software. Como fruto dessa compreensão, é possível construir ferramentas de apoio ao processo de ER que sejam mais amplamente utilizáveis, já que estas estarão baseadas em definições compartilhadas do domínio de requisitos.

Como forma de estabelecer uma conceituação básica acerca de requisitos, desenvolveu-se uma Ontologia de Requisitos de Software a ser usada como base para a construção de ferramentas de apoio ao processo de ER no ambiente de desenvolvimento de software (ADS) ODE (*Ontology-based software Development Environment*) (FALBO et. al, 2003). Como o nome indica, ODE tem sua fundamentação baseada em ontologias, partindo do pressuposto que, se as ferramentas do ADS são construídas baseadas em ontologias, a integração das mesmas é facilitada, pois os conceitos envolvidos são bem definidos e compartilhados.

Como ressaltado no capítulo anterior, ontologias são usadas em ODE, dentre outros, para estruturar o ambiente e sua infra-estrutura de gerência de conhecimento e para estabelecer uma forma padrão de comunicação entre agentes (humanos e de software) no ambiente (FALBO et. al, 2003). Ontologias servem a esse propósito, na medida em que elas definem um vocabulário de representação, que captura os conceitos e as relações de um domínio, e um conjunto de axiomas que restringe a sua interpretação (GUARINO, 1998).

Este capítulo está organizado da seguinte forma: a seção 4.2 discute o método de construção de ontologias utilizado; a seção 4.3 apresenta as questões de competência da ontologia de requisitos de software; na seção 4.4 é discutida a integração da Ontologia de Requisitos com outras ontologias utilizadas por ODE; a seção 4.5 trata da captura e formalização da ontologia, visando a responder as questões de competência; e, na seção 4.6 são tecidas algumas considerações finais.

4.2 A Metodologia Utilizada na Construção da Ontologia de Requisitos de Software

Na construção da Ontologia de Requisitos foi adotado o método SABiO (***S**ystematic **A**pproach for **Bu**ilding **O**ntologies*) (FALBO, 2004d) (FALBO, 1998), o qual define atividades para o processo de construção de ontologias, bem como uma linguagem gráfica para expressar ontologias.

Em sua versão original (FALBO, 1998), SABiO adotava LINGO como linguagem de modelagem de ontologias. Mais recentemente, com o crescimento da UML como linguagem de modelagem para ontologias, um perfil UML foi proposto, guardando os princípios definidos em LINGO (MIAN et al., 2003).

4.2.1 O Método SABiO (*Systematic Approach for Building Ontologies*)

O método SABiO, além de estabelecer atividades para a construção de ontologias, define, também, orientações de como proceder na sua realização. Além disso, é proposto um ciclo de vida para o processo de desenvolvimento de ontologias, indicando as interações entre as várias atividades, mostrado na Figura 4.1.

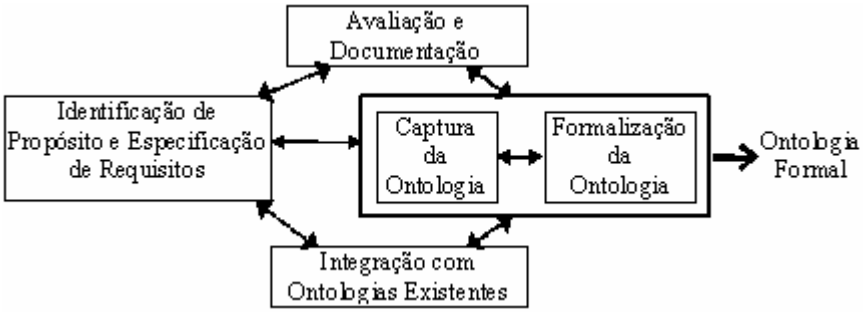


Figura 4.1- O Processo proposto pelo Método SABiO (FALBO, 1998).

A seguir, é dada uma breve descrição de cada uma das atividades do processo de desenvolvimento proposto em SABiO (FALBO, 2004) (FALBO et al., 1998):

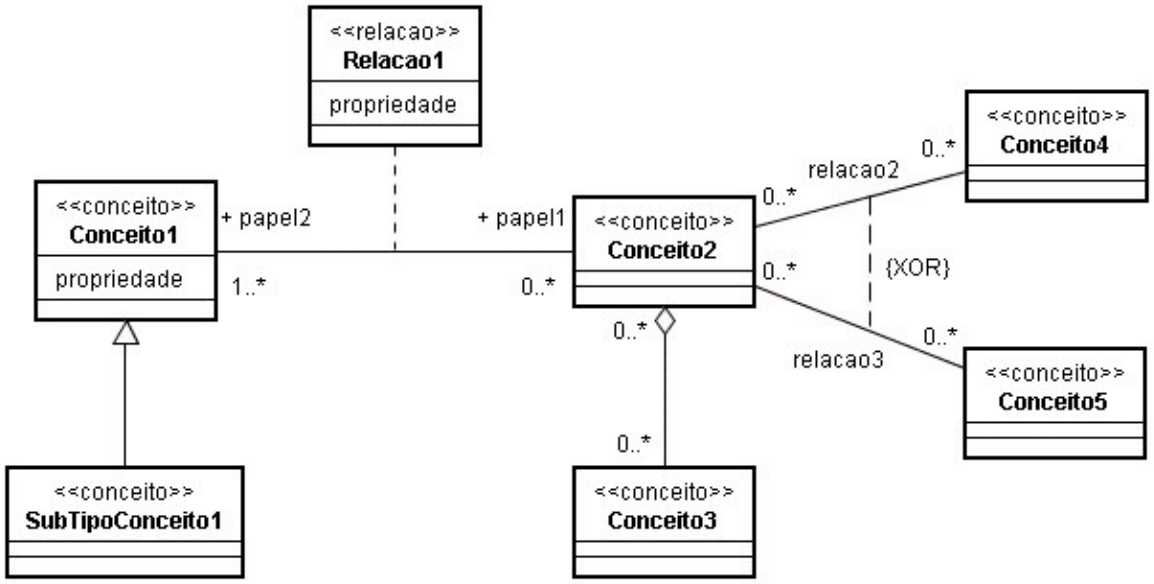
- **Identificação de Propósito e Especificação de Requisitos:** Visa a identificar claramente o propósito da ontologia e os usos esperados para ela, i.e., sua competência. Para sua realização, questões de competência são utilizadas.
- **Captura da Ontologia:** O objetivo é capturar a conceituação do domínio, com base na competência da ontologia. Conceitos, relações, propriedades e axiomas relevantes devem ser identificados e organizados. Um modelo usando uma linguagem gráfica e um dicionário de termos devem ser utilizados para facilitar a comunicação com os especialistas do domínio.
- **Formalização da Ontologia:** Visa a representar explicitamente a conceituação capturada pela ontologia em uma linguagem formal. Neste trabalho, utilizou-se lógica de primeira ordem.
- **Integração com Ontologias Existentes:** Durante a captura ou formalização da ontologia, pode ser necessário integrar a ontologia em questão com outras já existentes, para reutilizar conceituações previamente estabelecidas.
- **Avaliação da Ontologia:** A ontologia deve ser avaliada a fim de verificar se os requisitos estabelecidos na especificação estão sendo satisfeitos. A competência da ontologia e a alguns critérios de qualidade, como aqueles definidos por Gruber (1995), devem ser verificados.
- **Documentação:** Todo o desenvolvimento da ontologia deve ser documentado.

4.2.2 Um Perfil UML para Modelagem de Ontologias

A atividade de Captura da Ontologia proposta pelo método SABiO destaca a utilização de uma linguagem gráfica para facilitar a comunicação com os especialistas do domínio. Para tanto se utilizou o perfil UML para modelagem de ontologias proposto em (MIAN, 2003).

Dá-se o nome de perfil UML a um subconjunto pré-definido de elementos padrão da UML, associados a estereótipos, valores etiquetados e restrições, que conjuntamente especializam e configuram a UML para um determinado domínio de aplicação ou propósito particular. O perfil UML para modelagem de ontologias proposto em (MIAN, 2003), parcialmente mostrado na Figura 4.2, define elementos para representar conceitos, relações e

propriedades, bem como define a axiomatização associada a notações específicas para certas relações, seguindo a linha de pensamento da linguagem LINGO. Esses axiomas são ditos axiomas independentes de domínio e, usando o perfil proposto, estão implicitamente definidos pela linguagem de modelagem, não sendo necessário que o engenheiro de ontologias os escreva explicitamente.



Axiomas:

Todo-parte:

(AE1) $(\forall x) \neg parteDe(x,x)$

(AE2) $(\forall x,y)\ parteDe(y,x) \leftrightarrow todoDe(x,y)$

(AE3) $(\forall x,y)\ parteDe(y,x) \rightarrow \neg parteDe(x,y)$

(AE4) $(\forall x,y)\ parteDe(y,x) \rightarrow (\exists z)(parteDe(z,x))$

(AE5) $(\forall x,y,z)\ parteDe(z,y) \wedge parteDe(y,x) \rightarrow parteDe(z,x)$

Sub-tipo-de:

(AE6) $(\forall x,y,z)\ subTipoDe(x,y) \wedge subTipoDe(y,z) \rightarrow subTipoDe(x,z)$

(AE7) $(\forall x,y)\ subTipoDe(x,y) \rightarrow superTipoDe(y,x)$

Ou-exclusivo (XOR):

(AE8) $(\forall a \in C2)((\exists b)(b \in C3) \wedge R2(a,b)) \rightarrow \neg((\exists c \in C4) \wedge R3(a,c)))$

(AE9) $(\forall a \in C2)((\exists c)(c \in C4) \wedge R3(a,c)) \rightarrow \neg((\exists b \in C3) \wedge R2(a,b)))$

Figura 4.2 – Perfil UML para construção de ontologias e seus axiomas.

Para dar uma semântica diferente aos elementos de modelo da UML, foi utilizado o mecanismo de extensão estereótipo. Sendo assim, classes com estereótipo <<*conceito*>> representam conceitos da ontologia, da mesma forma que relações que contêm propriedades ou que possuem aridade maior que dois, são representadas como classes associativas com estereótipo <<*relação*>>.

As relações binárias sem propriedades são definidas como associações nomeadas. As propriedades de conceitos e relações são representadas como atributos de classes estereotipadas. Relações de supertipo e todo-parte são representadas como relações de generalização/especialização e de agregação/composição, respectivamente. Condicionantes entre relações são representados por outro mecanismo de extensão da UML, as restrições entre associações (MIAN, 2003).

Ainda que se esteja utilizando axiomas independentes de domínio para auxiliarem na formalização dos elementos representados pelo perfil da UML utilizado, tais axiomas não são suficientes, uma vez que tratam apenas da estruturação de conceitos e relações. Na formalização de uma ontologia, devem-se também tratar os significados e restrições dos conceitos e relações, bem como as leis de integridade que os regem. Para tanto, podem-se utilizar axiomas dependentes de domínio que podem ser dois tipos axiomas de consolidação e axiomas ontológicos. O primeiro tipo impõe restrições que precisam ser atendidas para que uma relação seja estabelecida consistentemente. O último pretende representar o conhecimento declarativo que pode derivar novo conhecimento a partir de conhecimento factual representado na ontologia, mas que não é capturado pela estruturação de conceitos e relações da ontologia.

4.3 Identificação de Propósito e Especificação de Requisitos

A Ontologia de Requisitos de Software deve ser capaz de responder às seguintes questões de competência:

QC1. O que é um requisito?

QC2. Qual a natureza de um requisito?

QC3. Quais são os requisitos de um projeto de software?

QC4. Para que módulos do sistema um requisito é alocado?

QC5. Quais os responsáveis por um requisito?

QC6. Quais os interessados (*stakeholders*) em um requisito?

QC7. Qual a origem de um requisito?

QC8. Qual o estado de um requisito?

QC9. Como um determinado requisito é decomposto em outros requisitos?

QC10. Que requisitos são dependentes de um determinado requisito?

QC11. Que requisitos são conflitantes entre si?

QC12. O que é uma especificação de requisitos de software (ERS)?

QC13. Que artefatos descrevem, modelam ou implementam um requisito?

QC14. Como gerenciar as mudanças nos requisitos e nos artefatos a eles relacionados?

QC15. Como avaliar a qualidade em requisitos?

4.4 Integração com Ontologias Existentes

Uma vez que requisitos estão no universo da Engenharia de Software, é útil examinar ontologias nesse domínio. Neste trabalho foram inspecionadas as seguintes ontologias, construídas de forma integrada:

- **Ontologia de Processo de Software:** desenvolvida em (FALBO, 1998) e recentemente revisada em (BERTOLLO, 2006), essa ontologia é composta de *Ontologias de Atividade, de Recurso e de Procedimento.* A Ontologia de Processo de Software busca modelar a conceituação acerca do domínio de processos de software, abordando conceitos como: atividade, artefato, processo, recurso, procedimento etc.
- **Ontologia de Artefatos:** desenvolvida em (NUNES, 2005), tem o intuito de prover um entendimento compartilhado acerca de alguns tipos de artefatos, a saber: documento, diagrama e artefatos de código. Por conseguinte, essa ontologia é subdividida nas seguintes ontologias: *Ontologia de Documento, Ontologia de Artefato de Código* e *Ontologia de Diagrama*.

- **Ontologia de Gerência de Configuração de Software:** desenvolvida em (NUNES, 2005), busca tratar de conceitos relacionados à gerência de configuração de software, atuando, basicamente, sobre artefatos produzidos e consumidos ao longo do processo de software e as ferramentas usadas para produzi-los.

- **Ontologia de Qualidade de Software:** desenvolvida em (DUARTE, 2001), essa ontologia tem por fim elaborar uma conceituação sobre o domínio de Qualidade de Software, tratando de questões como avaliação da qualidade, características de qualidade e métricas.

- **Ontologia de Organizações de Software:** proposta em (RUY, 2006), tem como propósito fornecer um vocabulário comum que possa ser utilizado para representar conhecimento útil sobre organizações de software, incluindo competências de seus membros.

A Figura 4.3 mostra as ontologias utilizadas na construção da Ontologia de Requisitos, destacando as dependências entre elas.

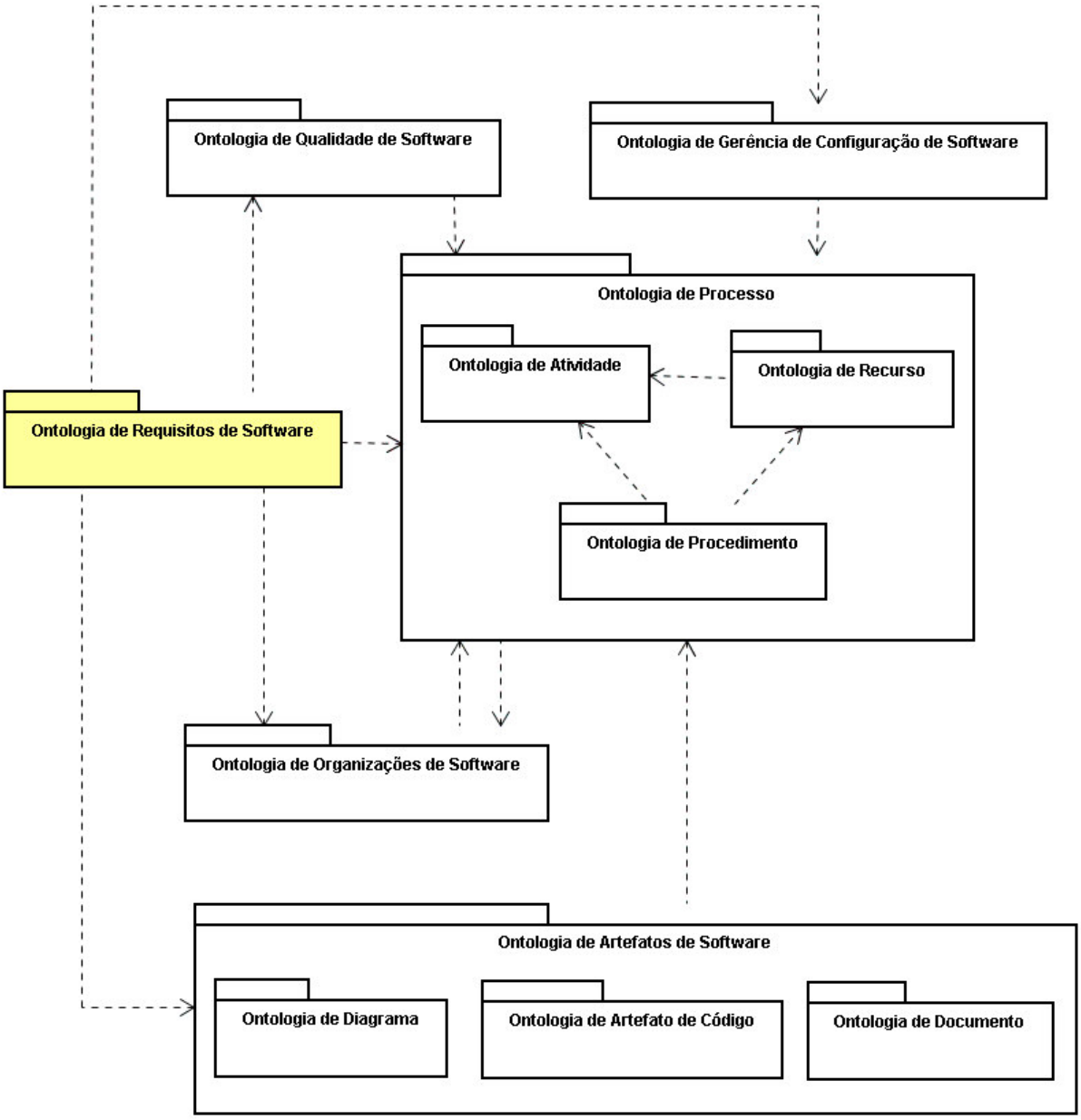


Figura 4.3 - Dependência com as ontologias utilizadas

Como mostra a Figura 4.3, visando à reutilização, procurou-se apoiar a construção da Ontologia de Requisitos nas conceituações das Ontologias de Processo, de Artefato (e suas respectivas sub-ontologias), de Qualidade de Software, de Gerência de Configuração de Software e de Organizações de Software.

Da Ontologia de Processo de Software, foram utilizados os conceitos *Atividade* e *Artefato.* O primeiro conceito foi utilizado para compor o contexto no qual um requisito é originado, uma vez que um requisito pode ter origem em uma atividade do processo de software. Já por meio de *Artefato* foi possível modelar o tratamento de requisitos em artefatos de software, buscando tratar a rastreabilidade.

É importante ressaltar que aspectos relacionados à estrutura do processo de Engenharia de Requisitos, como, por exemplo, atividades, sub-atividades, procedimentos etc, não foram alvo da Ontologia de Requisitos, pois os mesmos já são abordados pela Ontologia de Processo. De fato, a descrição de um processo de Engenharia de Requisitos pode ser feita pela instanciação da ontologia de processo de software.

A Ontologia de Artefato e suas sub-ontologias definem conceitos acerca dos tipos de artefatos e de suas respectivas estruturas. Para este trabalho, foi especialmente importante a Ontologia de Documento, dado que um Documento de Especificação de Requisitos de Software (DERS) é um *Documento*. Assim, toda a conceituação usada para descrever artefatos e, mais especificamente, documentos, foi reutilizada para tratar DERSs, incluindo a definição de modelos de documento organizacionais para documentos.

Outro aspecto a ser ressaltado é que o conceito de *Artefato* é tratado na Ontologia de Processo e não na Ontologia de Artefato. Isso porque a Ontologia de Artefato se propõe a conceituar os tipos de artefatos e não o que é um artefato. A conceituação de artefato é tratada pela Ontologia de Processo, pois um artefato é um produto ou insumo para atividades do processo de software.

A Ontologia de Qualidade de Software define uma série de conceitos relativos à avaliação da qualidade de software, incluindo a avaliação de produtos (artefatos) de software. Sendo assim, como um DERS é um tipo de artefato, toda essa conceituação acerca da qualidade de artefatos de software é igualmente válida para o documento de requisitos. Tal conceituação abrange *Características de Qualidade* diretamente e indiretamente mensuráveis, *Métricas, Medidas* etc.

A Ontologia de Gerência de Configuração de Software trata dos conceitos relacionados ao controle de versão e de alteração de itens de configuração, que representam elementos que estão sob gerência de configuração. Já estava contemplado nessa ontologia, que *Artefato* é um item de configuração. Sendo assim, a partir do momento que se definiu que um DERS é um artefato, a gerência de configuração do documento de requisitos passou a ser contemplada. Além disso, como requisitos também têm de ter sua configuração gerenciada, uma relação entre *Requisito* e *Item de Configuração* foi estabelecida.

Da Ontologia de Organizações de Software reutilizaram-se os conceitos *Projeto*, *Pessoa e Equipe*. Uma vez que requisitos são definidos no âmbito de um *Projeto*, utilizou-se esse conceito para relacionar *Requisito* a *Projeto*. Pessoa representa os indivíduos de uma

organização. Esse conceito é útil ao se definirem os interessados (*stakeholders*) e os responsáveis por um requisito. O último conceito, *Equipe*, define as equipes de uma organização que podem estar alocadas a um projeto.

Por meio da reutilização de ontologias, algumas questões de competência da Ontologia de Requisitos foram totalmente respondidas e outras parcialmente respondidas. Em alguns casos, entretanto, ainda se fez necessário propor extensões às ontologias reutilizadas, de modo a responder algumas questões de competência. Na seção seguinte, discutimos como as questões de competência foram respondidas por meio dos conceitos reutilizados e de novos conceitos identificados no contexto da Ontologia de Requisitos.

4.5 Captura e Formalização da Ontologia

Analisando as questões de competência anteriormente relacionadas, identificamos alguns aspectos relevantes:

1. Definição e Taxonomia de Requisitos (questões 1 a 4 e 12)
2. Aprovação e interesse em requisitos (questões 5 e 6)
3. Gerência de Requisitos e Gerência de Configuração (questões 7 a 11, 13 e 14)
4. Qualidade em requisitos (questão 15)

Definição e Taxonomia de Requisitos

De maneira geral, requisitos são sentenças que descrevem serviços que um sistema deve prover, restrições que ele deve obedecer e características que ele deve possuir (*QC1*). Além disso, requisitos são definidos no âmbito de um projeto (*QC3*).

Conforme discutido na seção anterior, um DERS é um *Artefato*. Mais especificamente, um DERS é um *Documento* (*QC12*), que, segundo (NUNES, 2005), é um artefato de software não passível de execução, constituído tipicamente de declarações textuais, normalmente associado a padrões organizacionais (roteiros) que definem a forma como ele deve ser produzido. A Figura 4.4 mostra o modelo da ontologia referente ao tratamento dessas questões de competência. Os conceitos destacados foram propostos neste trabalho. Os demais foram reutilizados de outras ontologias, conforme discutido na seção anterior.

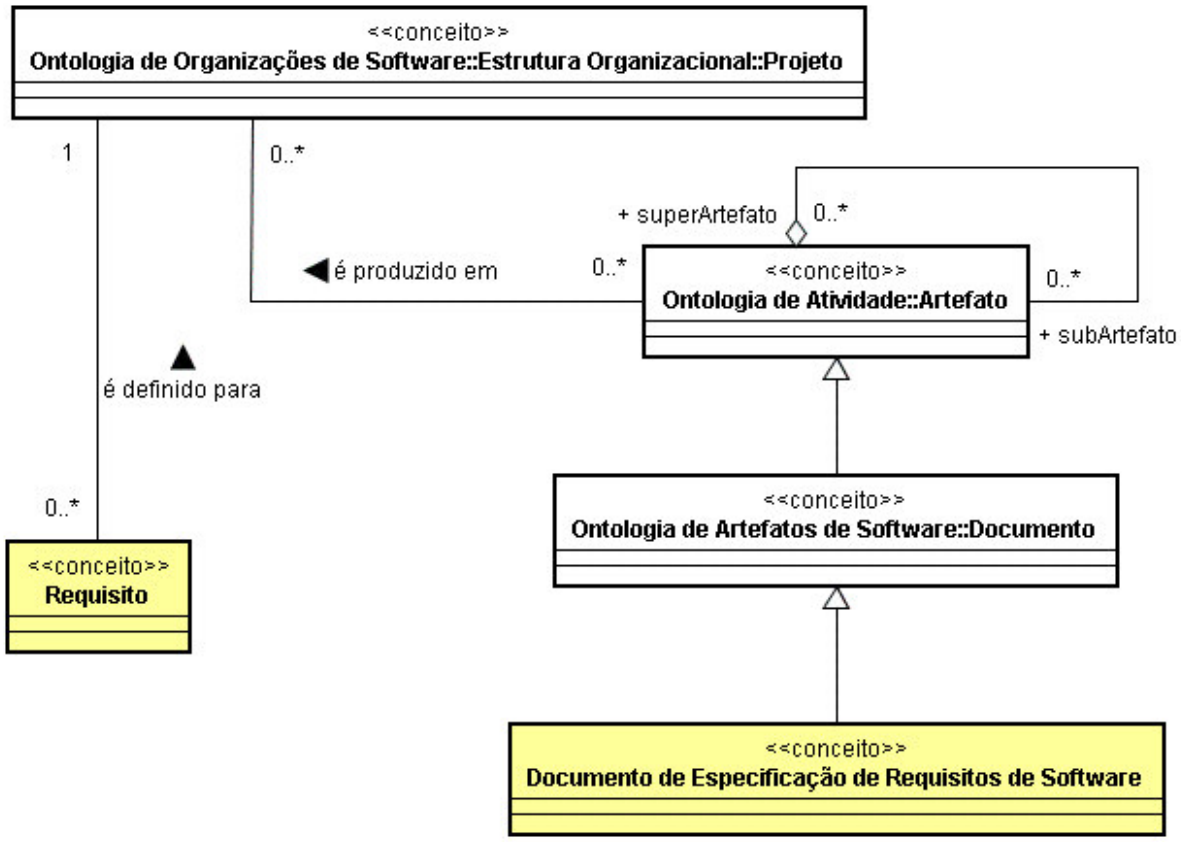


Figura 4.4 - Definição de Requisito e DERS

Para formalizar a existência de diferentes tipos de artefatos, foram usados os seguintes predicados, representando cada um dos tipos identificados na taxonomia: *artefato(a)*, indicando que *a* é um artefato; *documento(d)*, indicando que *d* é um documento; e *documentoERS(e)*, indicando que *e* é um documento de especificação de requisito de software.

$(\forall d)(documento(d) \rightarrow artefato(d))$	(AE1)
$(\forall e)(documentoERS(e) \rightarrow documento(e))$	(AE2)

Como esses e outros axiomas independentes de domínio são implicitamente capturados pelo perfil UML utilizado, conforme mostra a Figura 4.2, neste texto esses axiomas não serão mais apresentados.

Ainda que requisitos sejam definidos no âmbito de um projeto, é importante definir, também, a que módulos do software tal requisito está sendo alocado (*QC4*). Para responder a essa questão de competência, foi proposta a conceituação de *Módulo.* Nesse caso módulos compõem um *Escopo* definido para um dado *Projeto.* Ademais, módulos podem ser

decompostos em outros módulos, da mesma forma que pode existir dependência entre módulos. A Figura 4.5 mostra o modelo correspondente.

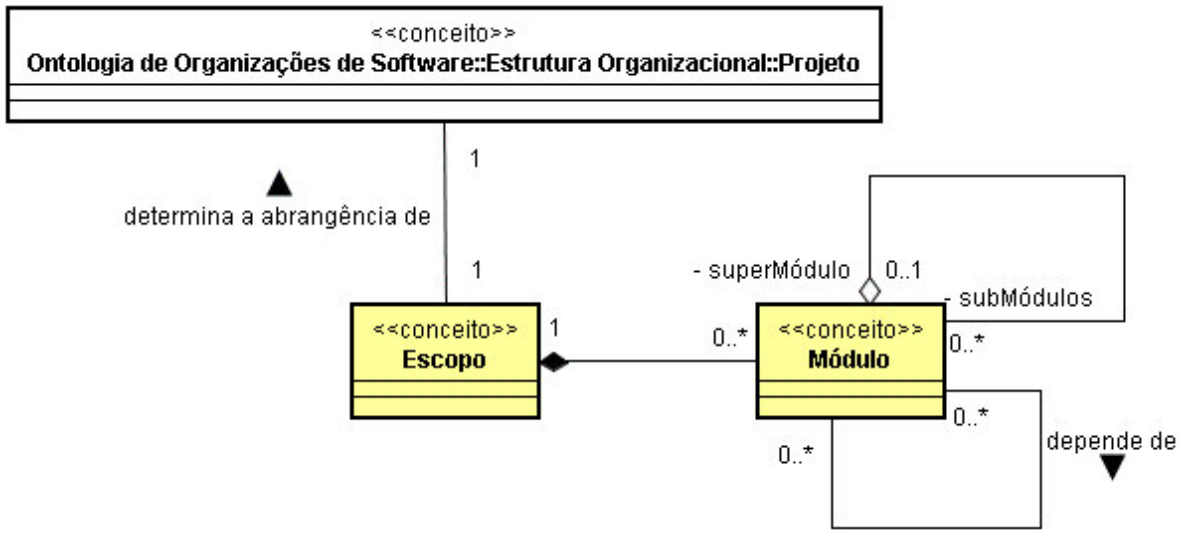


Figura 4.5 - Conceituação de Módulo

Visando à formalização da conceituação de módulo, é necessária a definição de alguns axiomas para a relação *dependência.* Primeiramente, é importante observar que a relação de dependência é transitiva, ou seja, se um módulo depende de um segundo módulo e este depende de um terceiro, então o primeiro depende do terceiro. Essa regra é expressa na sentença a seguir, na qual o predicado *dependeDe(m1,m2)* indica que o módulo *m1* depende do módulo *m2*.

$$(\forall m1, m2, m3)\ (dependeDe(m1, m2) \wedge dependeDe(m2, m3) \rightarrow dependeDe(m1, m3)) \qquad \text{(A1)}$$

Assim, pelo axioma A1, considerando que *m1, m2* e *m3* são módulos, tem-se que se *m1*depende de *m2* e esse, por sua vez, depende de *m3*, então *m1* depende de *m3*.

Os axiomas abaixo formalizam a dependência entre módulos compostos. Nesses axiomas, o predicado *submódulo(m1, m2)*, indica que *m1* é um sub-módulo de *m2* .

$$(\forall m1, m2, m3)\ (dependência(m1, m2) \wedge submódulo(m1, m3) \rightarrow dependeDe(m3, m2)) \qquad \text{(A2)}$$

$$(\forall m1, m2)\ (submódulo(m1, m2) \rightarrow dependeDe(m2, m1)) \qquad \text{(A3)}$$

Considerando que *m1*, *m2* e *m3* são módulos, pelo axioma A2 tem-se que se *m1* depende de *m2* e *m1* é sub-módulo de *m3*, então *m3* depende de *m2*. Já o axioma A3 destaca o fato de que se *m1* é sub-módulo de *m2*, então *m2* depende de *m1*.

Uma vez definido o conceito de *Módulo*, pode-se agora tratar da alocação de requisitos a módulos, de modo a responder à questão de competência *QC4.* A Figura 4.6 apresenta o modelo proposto para responder a essa questão de competência.

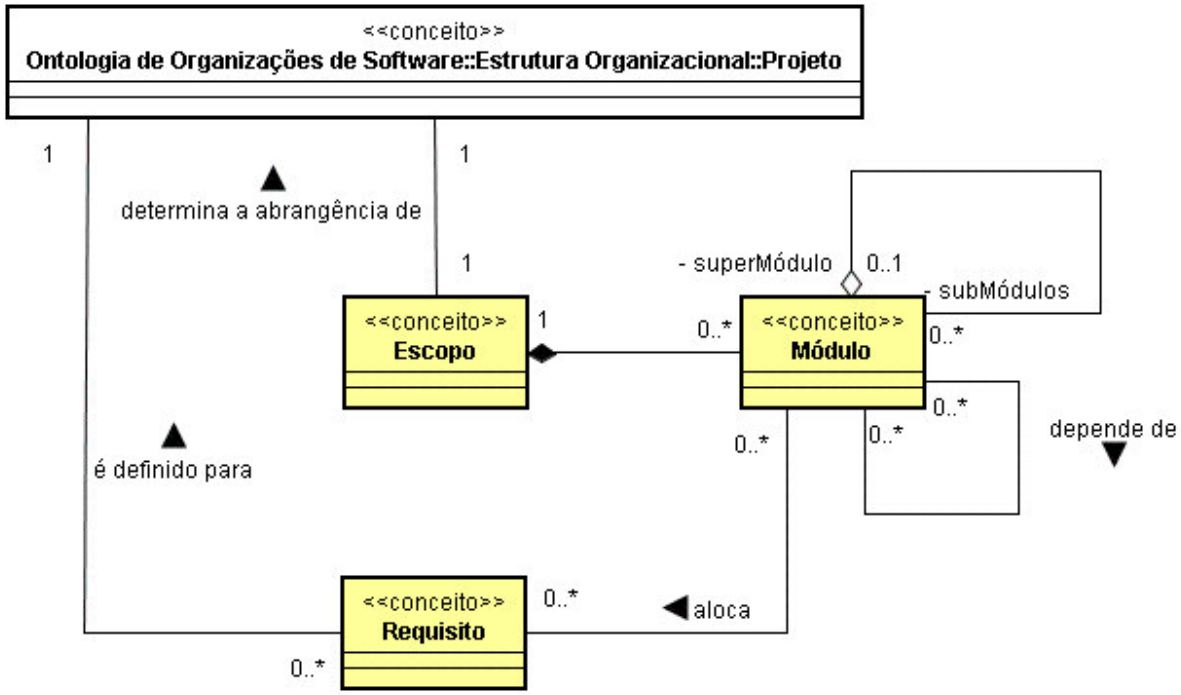


Figura 4.6 - Alocação de Requisito a Módulo

Para formalizar questões importantes na alocação de requisitos a módulos, foi utilizado o predicado *aloca(m, r)*, indicando que o módulo *m* aloca o requisito *r*. Assim, a seguinte sentença, apontando que os requisitos alocados a um módulo estão alocados também a seus super-módulos, é válida:

$$(\forall r, m1, m2)\ (submodulo(m1, m2) \wedge aloca(m1, r) \rightarrow aloca(m2, r)) \quad \text{(A4)}$$

Além disso, é importante notar que um requisito só pode estar alocado a um módulo que esteja no escopo do projeto que definiu esse requisito. Essa restrição é formalizada no axioma de consolidação (AC) abaixo apresentado, no qual o predicado *ehDefinidoPara(r, p)* indica que um requisito *r* foi definido no contexto de um projeto *p*, o predicado *parteDeEscopo(m, e)* indica que um módulo *m* compõe um escopo *e*; e o predicado *determinaAbrangenciaDe(e,p)* indica que um escopo *e* foi definido para um projeto *p*.

$(\forall\ p,\ r,\ m)\ (aloca(m,\ r) \rightarrow ehDefinidoPara(r,\ p) \wedge parteDeEscopo(m,e) \wedge ehDefinidoPara(e,\ p))$ (AC1)

Como existem vários tipos de requisito, se faz necessário criar uma taxonomia para organizá-los. No entanto, não há consenso a respeito de quais tipos de requisito devem ser utilizados, exceto no que tange a *Requisitos Funcionais* e *Requisitos Não-Funcionais*. Esses dois tipos estão, de fato, enraizados na literatura acerca de Engenharia de Requisitos e no dia-a-dia das empresas de software.

Dessa maneira, optou-se por considerar que, por princípio básico, um requisito é funcional ou não-funcional, como mostra a Figura 4.7 por meio dos conceitos *RequisitoFuncional* e *RequisitoNaoFuncional*. A partir daí, pode-se classificá-lo de acordo com a taxonomia definida pela organização. Essa taxonomia é definida segundo a modelagem do conceito *TipoRequisito*, que destaca que um subtipo pode ter vários supertipos e, para cada supertipo, é possível criar subtipos. Essa conceituação de tipos de requisitos objetiva responder à questão de competência *QC2*.

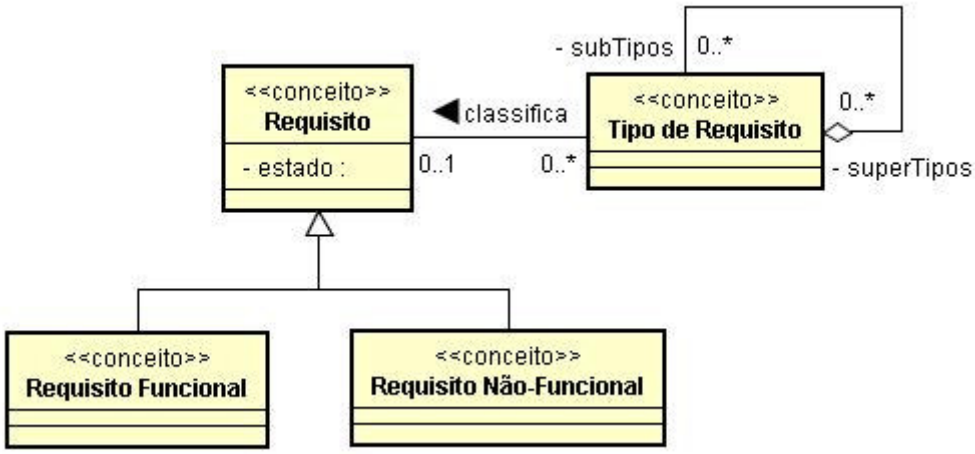


Figura 4.7 - Taxonomia de Requisito

Para a formalização da taxonomia de tipos de requisitos, foram utilizados os seguintes predicados: *subtipo(t1, t2)*, indicando que *t1* é um subtipo de *t2*; *supertipo(t2, ,t1)*, indicando que *t2* é um supertipo de *t1*. Esses predicados estão relacionados segundo a seguinte sentença:

$(\forall t1, t2)\ (subtipo(t1, t2) \leftrightarrow supertipo(t2, t1))$ (A5)

Os outros axiomas, implicitamente tratados pelo perfil UML utilizado, no que tange à relação todo-parte (axiomas AE3 a AE6 da Figura 4.1), garantem outras propriedades

importantes para a formação de uma taxonomia de tipos de requisito, tais como anti-simetria e transitividade.

Aprovação e Interesse em Requisitos

O processo de Engenharia de Requisitos envolve várias pessoas de diversas áreas e com perspectivas diferentes. Dessa forma, é interessante que se saiba os interessados em cada requisito, com o intuito de facilitar a localização dos mesmos para facilitar negociações, esclarecimentos e discutir impactos de mudanças (*QC6*).

Além disso, é imprescindível que existam pessoas responsáveis por um dado requisito. Tais pessoas podem, por exemplo, aprovar um requisito antes que ele seja tratado por atividades subseqüentes do processo de software (*QC5*).

A Figura 4.8 mostra a parte da ontologia que trata dessas questões. O conceito *Requisito* se relaciona com o conceito *Pessoa*, da Ontologia de Organizações de Software, por meio das relações *interesse* e *responsabilidade*. Como os nomes sugerem, a primeira relação representa o interesse de pessoas em um requisito e a segunda representa os responsáveis por um requisito. Ademais, pessoas podem compor equipes que estão alocadas a projetos.

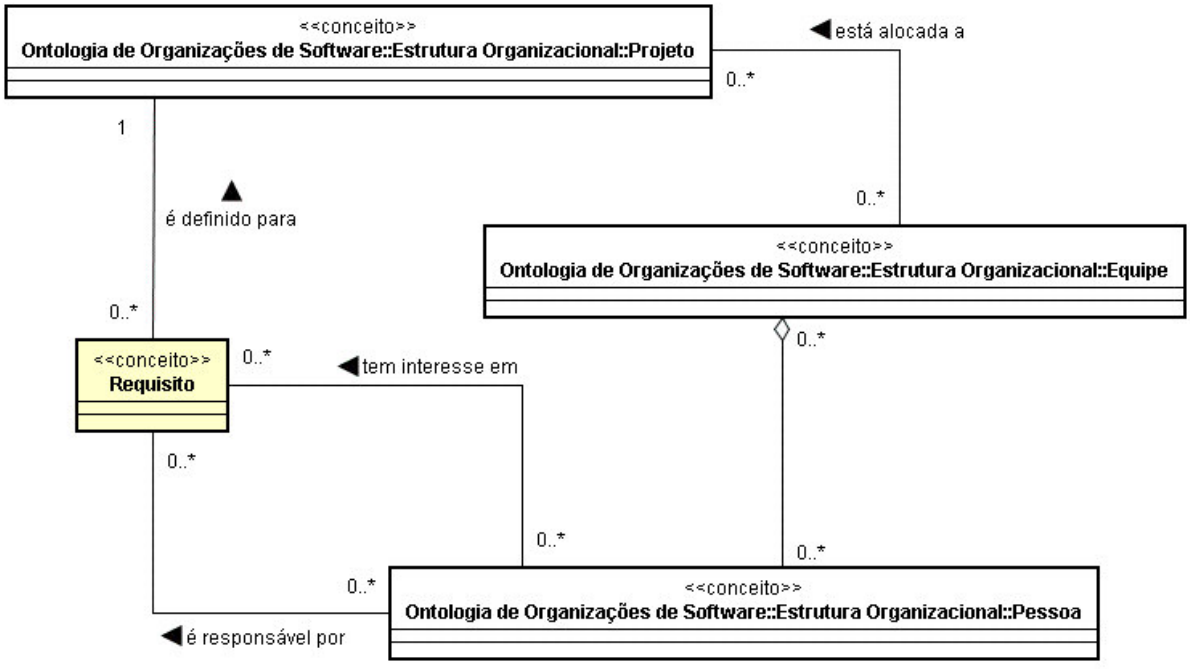


Figura 4.8 - Aprovação e Interesse em Requisito

Considere os seguintes predicados que são utilizados na construção de axiomas para formalizar restrições envolvidas no modelo da Figura 4.8: *ehResponsavelPor(p, r)* indica que uma pessoa *p* é responsável por um requisito *r*; *ehDefinidoPara(r, pr)* indica que um requisito

r é definido para o projeto *pr*; *parteDeEquipe(p, e)* indica que a pessoa *p* participa da equipe *e*; e *estahAlocadaA(e, pr)* indica que uma equipe *e* está alocada a um projeto *pr*. Com esses predicados pode-se escrever a seguinte sentença, que tem o objetivo de formalizar o fato de que, se uma pessoa é responsável por um requisito, então essa pessoa deve participar de uma equipe do projeto no qual o requisito foi definido.

$$(\forall r, p, pr)\,(ehResponsavelPor(p, r) \rightarrow (\exists e)\,(parteDeEquipe(p, e) \wedge estahAlocadaA(e, pr) \wedge ehDefinidoPara(r, pr))) \quad \text{(AC2)}$$

Gerência de Requisitos e Gerência de Configuração

Segundo (ROBERTSON et al., 1999), a gerência de requisitos compreende controle de mudanças, controle de versão, acompanhamento de estado de requisitos e rastreabilidade.

Em relação à rastreabilidade e ao controle de mudanças, é necessário estabelecer uma rede de ligações com o objetivo de detectar mais facilmente o impacto de uma mudança. Essas ligações, normalmente, buscam fornecer informações acerca de dependência (*QC10*), composição (*QC9*) e conflito (*CQ11*) entre requisitos, além de informações sobre onde um requisito está descrito, modelado e codificado (*QC13*) e qual a origem de um requisito (*QC7*).

A Figura 4.9 mostra a parte da ontologia que trata da interação entre requisitos. A relação de dependência permite estabelecer ligações de dependência entre requisitos (*QC10*), indicando que, se um requisito do qual outros requisitos dependem for alterado, é provável que os requisitos dele dependentes sofram algum impacto.

No caso da composição, a situação é análoga. Por meio dessa relação, é possível estabelecer a composição de requisitos (*QC9*). Além disso, se um requisito que é parte de outro for alterado, é provável que o requisito composto tenha de que ser revisado. Da mesma forma, se um requisito composto for alterado, é possível que os requisitos que o compõem tenham que ser revisados.

Finalmente, pode-se estabelecer relações de conflito entre requisitos (*QC11*), permitindo controlar conflitos e mantê-los aceitáveis ou detectar quando um conflito definitivamente não pode existir, levando à revisão dos requisitos envolvidos.

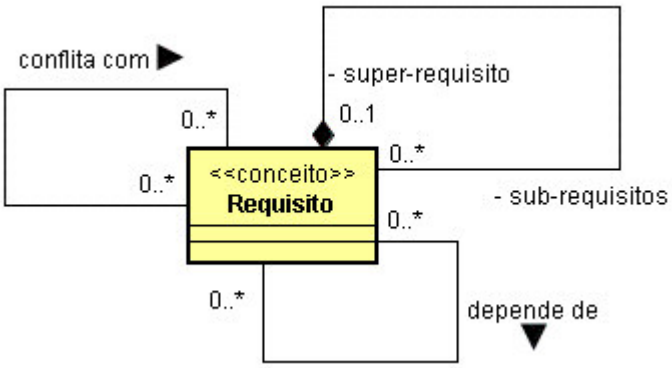


Figura 4.9 - Composição, Dependência e Conflito de Requisito

Com o objetivo de explicitar restrições ou verdades que o modelo exposto na Figura 4.8 não é capaz de capturar, foram definidos alguns axiomas, expostos a seguir.

O predicado *subrequisito(r1, r2)* indica que o requisito *r1* compõe do requisito *r2*, enquanto o predicado *superrequisito(r2, r1)* indica que o requisito *r2* é composto por *r1*. Esses predicados estão relacionados segundo a seguinte sentença:

$$(\forall r1,r2)\,(subrequisito(r1, r2) \leftrightarrow superrequisito(r2, r1)) \qquad \text{(A6)}$$

Vale lembrar que a relação de composição de requisitos é transitiva, assimétrica e anti-reflexiva. Contudo, como esses axiomas são implicitamente descritos pela notação utilizada, não serão aqui tratados novamente.

No que concerne à relação de dependência entre requisitos, é importante realçar que a mesma é transitiva, ou seja, se um requisito *r1* depende de um segundo requisito *r2* e este depende de um terceiro *r3*, então o primeiro depende do terceiro. Essa relação é expressa na sentença abaixo, na qual o predicado *dependeDe(r1, r2)* indica que o requisito *r1* depende de *r2*:

$$(\forall r1, r2, r3)\,(dependeDe(r1, r2) \wedge dependeDe(r2, r3) \rightarrow dependeDe(r1, r3)) \qquad \text{(A7)}$$

Os axiomas a seguir formalizam a dependência entre requisitos compostos. Considerando que *r1, r2* e *r3* são requisitos, o axioma (A8) destaca que se *r1* depende de *r2* e *r1* compõe *r3*, então *r3* depende de *r2*. O axioma (A9), por sua vez, destaca que se *r1* compõe *r2*, então *r2* depende de *r1*.

$$(\forall r1, r2, r3)\ (dependeDe(r1, r2) \wedge subrequisito(r1, r3) \rightarrow dependeDe(r3, r2)) \quad \text{(A8)}$$

$$(\forall r1, r2)\ (subrequisito(r1, r2) \rightarrow dependeDe(r2, r1)) \quad \text{(A9)}$$

A relação de conflito entre requisitos tem a propriedade de ser simétrica, ou seja, se um requisito *r1* está em conflito com *r2*, então *r2* está em conflito com *r1*. Para formalizar essa propriedade da relação de conflito, o seguinte axioma deve ser observado, no qual o predicado *conflitaCom(r1, r2)* indica que o requisito *r1* conflita com o requisito *r2*:

$$(\forall r1, r2)\ ((conflitaCom(r1, r2)) \rightarrow conflitaCom(r2,r1)) \quad \text{(A10)}$$

Com o exposto acima, são respondidas as questões de competência *QC9*, *QC10* e *QC11*.

Para se ter um controle de mudanças mais efetivo, é interessante que se possa avaliar o impacto das mudanças de um requisito nos demais artefatos produzidos e consumidos no processo de software. Sendo assim, saber quais documentos, tais como documentos de especificação de requisitos de software, de análise ou de projeto, bem como quais componentes serão afetados por uma mudança em um requisito é importante. Por outro lado, saber quais requisitos serão afetados, caso um componente ou diagrama seja alterado, também é de extrema valia (*QC13*). A Figura 4.10 mostra a relação *tratamento* entre *Requisito* e *Artefato*, a qual permite estabelecer essas ligações. Por meio dessa relação, é possível rastrear em quais artefatos do processo de software um requisito foi tratado, no que se convenciona chamar de rastreabilidade vertical.

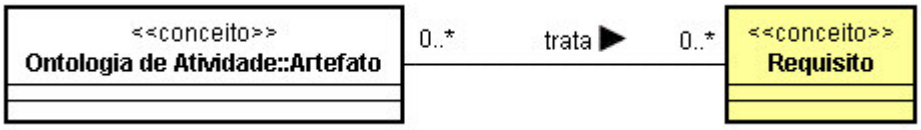


Figura 4.10 - Tratamento de Requisito

Para manter a rastreabilidade de um requisito desde a sua concepção até a sua implementação, se faz necessário ligar o requisito à sua origem, ou seja, ligá-lo ao contexto no qual ele se originou (*QC7*). Esse contexto pode descrever a utilização de artefatos, a interação entre pessoas, consulta a *sites* na Internet etc, e pode, ainda, se dar no âmbito de uma atividade do processo de software. De fato, várias são as possibilidades nas quais um requisito pode surgir e vários são os elementos que podem ser consultados na busca por um

requisito. Sendo assim, é importante dar ao Engenheiro de Requisitos uma maneira geral de descrever o contexto no qual um requisito surgiu. A Figura 4.11 mostra a porção da ontologia que trata da origem de um requisito, buscando responder a questão de competência *QC7*.

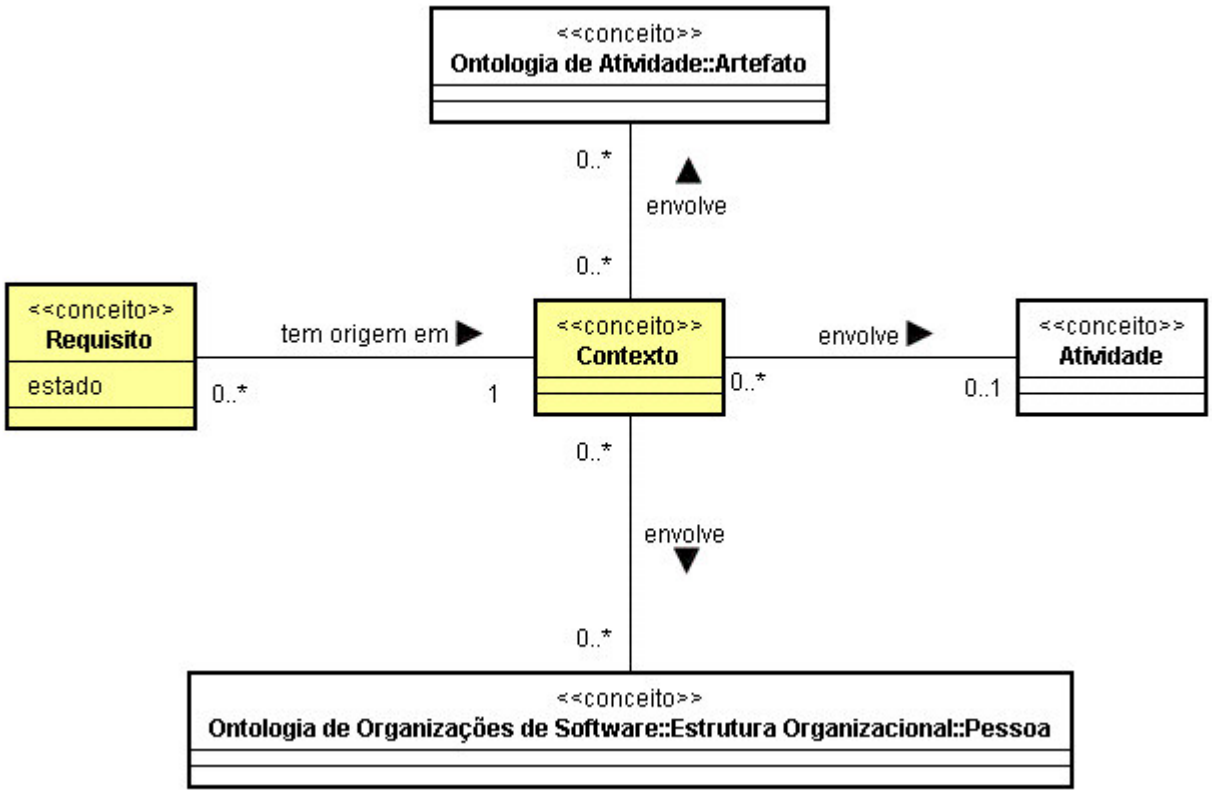


Figura 4.11 - Origem e Estado de Requisito

Finalmente, é importante controlar o estado de um requisito, definindo em um determinado momento, em que fase de seu ciclo de vida o requisito se encontra (*QC8*). Os estados possíveis de um requisito são dependentes de cada organização, ou seja, das práticas utilizadas por ela e podem ser, por exemplo: *Proposto*, *Em Alteração*, *Sob Verificação*, *Rejeitado* etc.

Outro ponto importante para a Gerência de Requisitos é o controle de versão de requisitos. Como destacado na seção anterior, a Ontologia de Gerência de Configuração de Software, trata do controle de versão de artefatos. Logo, o DERS já está contemplado por essa ontologia. Resta, portanto, estabelecer como gerenciar a configuração de um requisito.

O conceito central da Ontologia de Gerência de Configuração é o conceito de *Item de Configuração*, o qual representa uma entidade que faz parte da configuração de um software, identificada de forma única, que está sob gerência de configuração e, assim, só pode ser

alterada segundo um procedimento de controle de alteração formalmente estabelecido e documentado. Portanto, para permitir que a ontologia de requisitos fosse capaz de considerar a gerência de configuração de requisitos bastava tratá-los como *Itens de Configuração*. Reutilizando essa conceituação e introduzindo a relação *derivação* , esse aspecto pode ser tratado, como mostra a Figura 4.12. Além disso, teve-se o cuidado de determinar que as relações de derivação são exclusivas, ou seja, um item de configuração está relacionado com um requisito ou com um artefato de forma exclusiva.

No que tange à gerência de configuração, vale destacar que um *Item de Configuração* possui várias *Variações* (indica qual a versão ou variante de um determinado item de configuração), que podem ser tanto do tipo *Variante* quanto *Versão*. Cada *Variação* tem seu controle de alteração tratado pelas relações de submissão e resultado com *Alteração*, que capturam a solicitação de uma alteração e sua realização. Um *Recurso Humano* tem acesso a um *Item de Configuração*. Além disso, um *Recurso Humano* pode solicitar, autorizar e executar uma alteração em um *Item de Configuração*. Esses conceitos são tratados em mais detalhes na Ontologia de Gerência de Configuração de Software (NUNES, 2005).

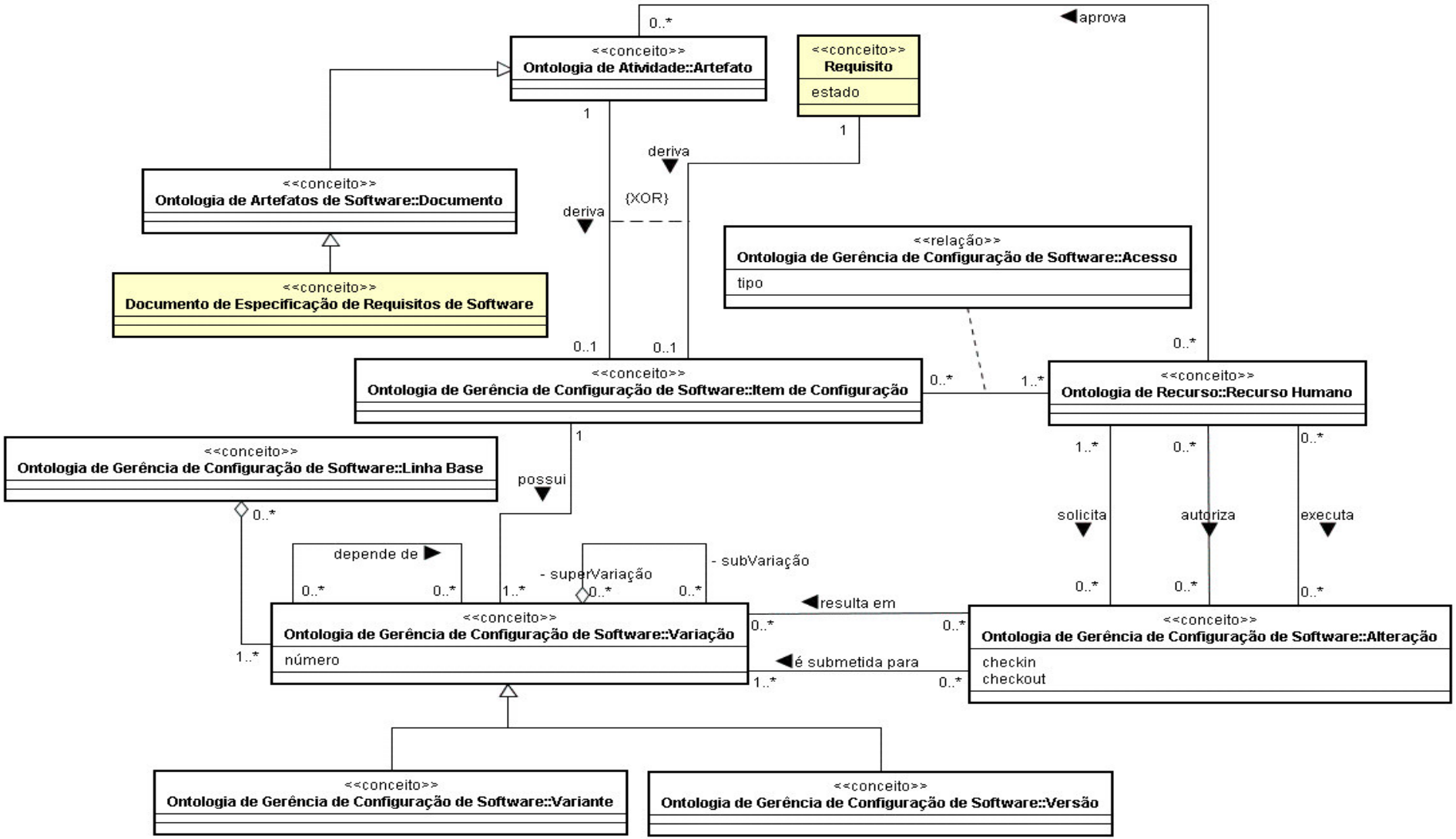


Figura 4.12 - Gerência de Configuração de Requisito e DERS

Vale ressaltar que a questão de competência *QC14* é respondida pelo conjunto da conceituação acima discutida, pois controlar mudanças nos requisitos e nos artefatos a eles relacionados tem ligação direta com as relações modeladas. Assim, para estabelecer esse controle, se faz necessário navegar por essas ligações de modo a detectar, por exemplo, o impacto de alterações.

Qualidade em Requisitos e DERS

A qualidade do produto de software pode se dar por meio da qualidade do seu processo de desenvolvimento e pela qualidade dos artefatos produzidos e consumidos ao longo desse processo. Dessa forma, considerando que um Documento de Especificação de Requisitos de Software é um artefato, avaliar a qualidade desse documento envolve aspectos relativos ao controle da qualidade dos artefatos de software produzidos no processo de Engenharia de Requisitos. Essa qualidade pode ser avaliada por meio da utilização de métricas e indicadores em uma análise crítica realizada através de inspeções (SAYÃO et al., 2003).

Duarte (2001), em sua Ontologia de Qualidade de Software, define características de qualidade para artefatos do processo de software. Uma vez que o conceito *Documento de Especificação de Requisitos de Software* foi considerado um *Documento*, que é um tipo de *Artefato*, a conceituação proposta por Duarte (2001) também é válida para o documento de requisitos. Além disso, a Ontologia de Qualidade contempla a utilização de métricas para a avaliação da qualidade de artefatos, respondendo à questão de competência *QC15*.

A Figura 4.13 mostra parte da conceituação da Ontologia de Qualidade, focando, principalmente, nas características de artefato, as quais são o foco da Ontologia de Requisitos.

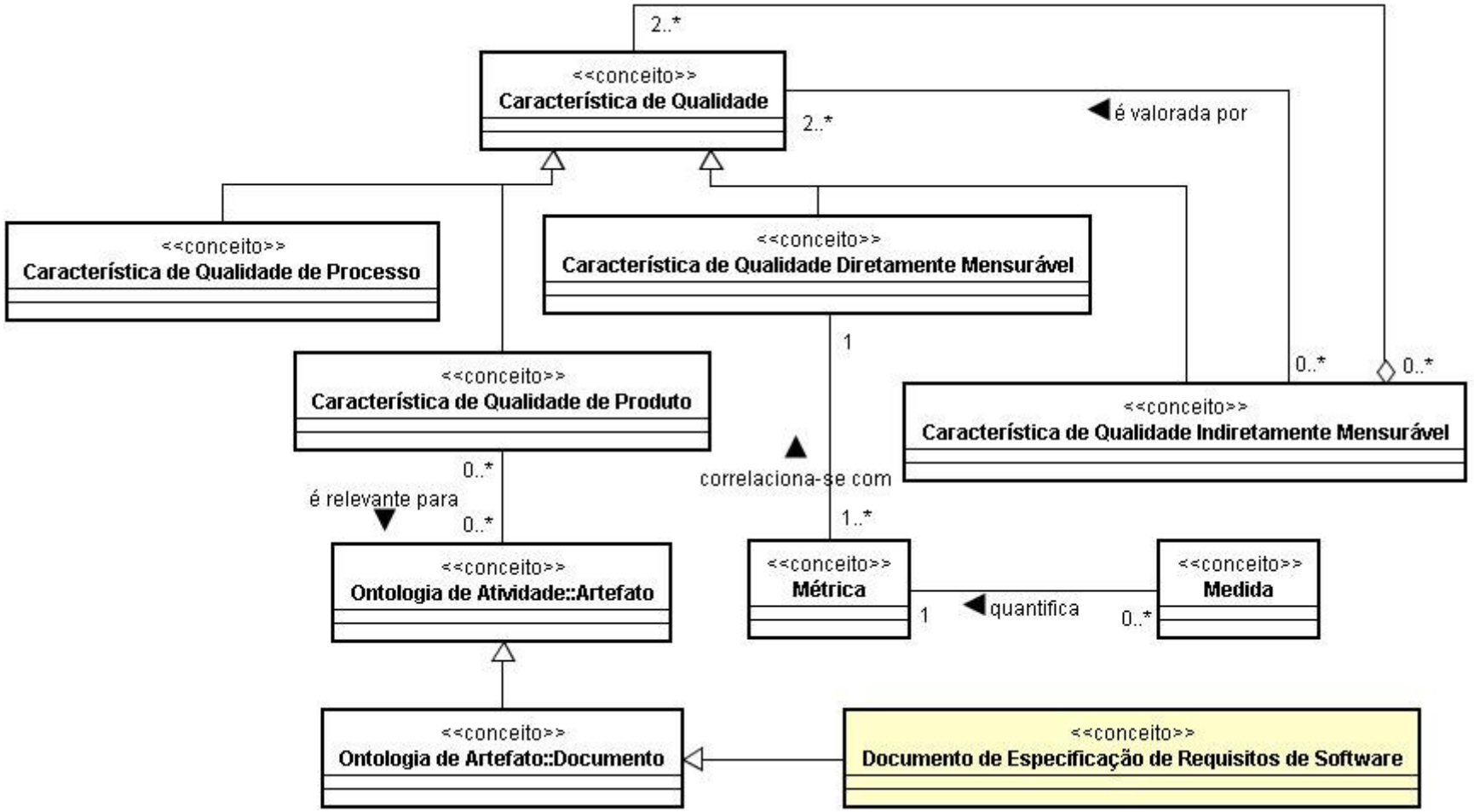


Figura 4.13 - Características de Qualidade de Requisito e ERS

Cada Artefato pode ter um conjunto de *Características de Qualidade de Produto* relevantes para a avaliação de sua qualidade. Essas características de qualidade podem ser direta ou indiretamente mensuráveis. Uma *Característica de Qualidade Diretamente Mensurável* tem correlação com uma *Métrica* que é usada para quantificar uma *Medida* para ela. Uma *Característica de Qualidade Indiretamente Mensurável* pode ser medida por meio da combinação de valores de suas sub-características, que podem ser direta ou indiretamente mensuráveis.

Para se ter uma idéia unificada do modelo da conceituação proposta, a Figura 4.14 mostra um diagrama que contém os conceitos e as relações diretamente envolvidos na conceituação da Ontologia de Requisitos. Para melhorar a legibilidade da figura, os espaços de nomes das ontologias de origem de cada conceito não são mostrados. Entretanto, conceitos da ontologia de requisitos são destacados em amarelo.

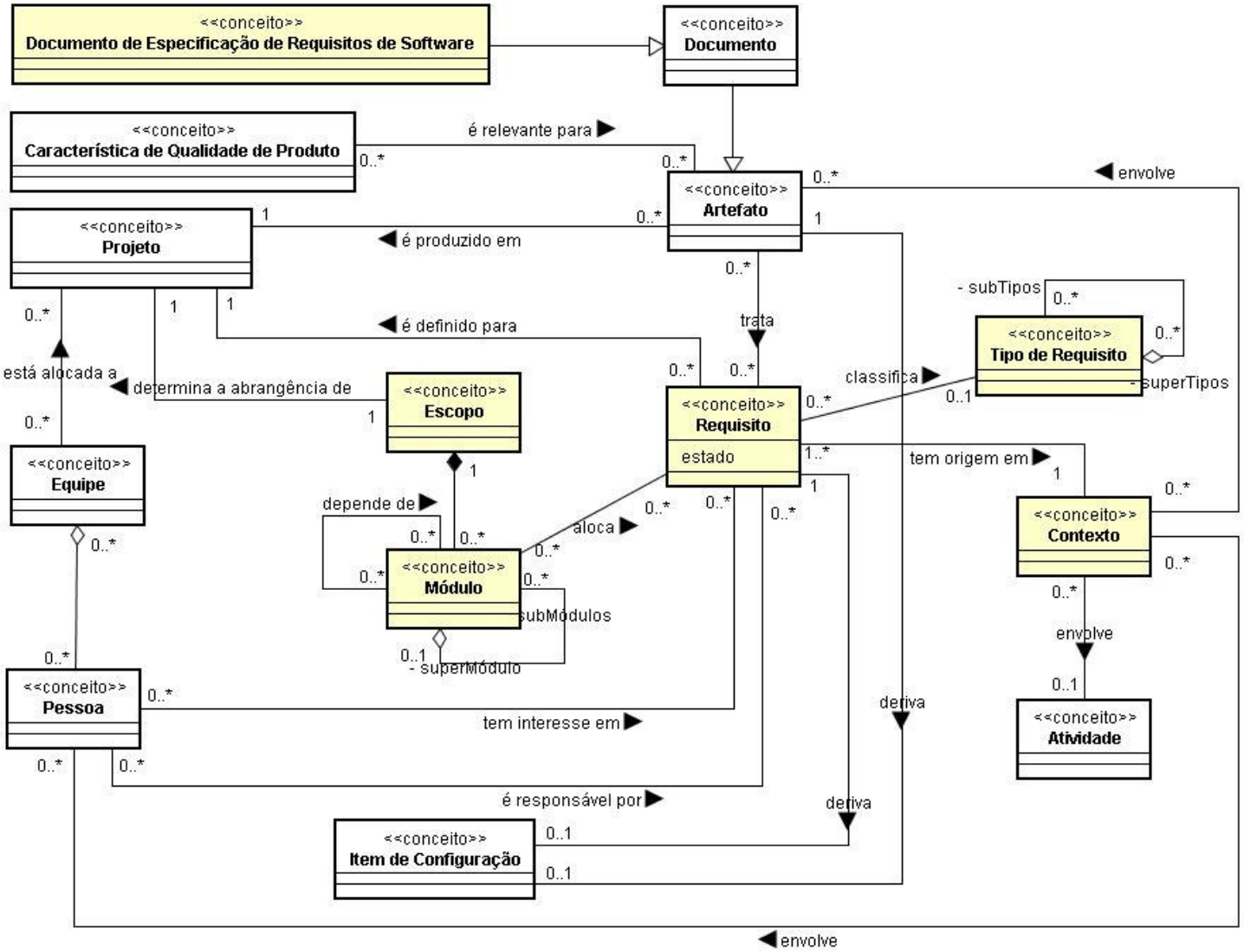


Figura 4.14 - Modelo completo da Ontologia de Requisitos de Software

Pelo método SABiO, é interessante utilizar, na atividade de Captura da Ontologia, um Dicionário de Termos para definir mais claramente os significados dos termos utilizados para nomear conceitos, propriedades, papéis e relações adotados na conceituação. Sendo assim, a Tabela 4.1 mostra o dicionário de termos da Ontologia de Requisitos.

Tabela 4.1 – Dicionário de Termos da Ontologia de Requisitos de Software.

aloca	Relação entre **requisito** e **módulo** que indica que um **requisito** está alocado a um **módulo** definido no **escopo** do **projeto**. Dessa forma, a construção do **módulo** deve se manter fiel às sentenças dos **requisitos** alocados a ele.
classifica	Relação entre **tipo de requisito** e **requisito,** utilizada para caracterizar um requisito, ou seja, definir qual é o tipo do requisito.
conflita com	Relação entre **requisitos** que descrevem características que o sistema deve possuir, mas que são antagônicas. Dessa forma, pode-se utilizar essa relação para controlar conflitos de **requisitos** de forma a mantê-los aceitáveis ou rejeitá-los.
Contexto	Caracteriza qual a situação na qual um **requisito** surgiu, procurando descrever quais as **pessoas** envolvidas, quais os **artefatos** analisados e em qual **atividade** do **processo** de software o requisito surgiu.
depende de (módulos)	Relação entre **módulos**, indicando quais **módulos** dependem de outros. Assim, é possível identificar quais **módulos** podem sofrer impacto quando da ocorrência de mudanças em um **módulo** específico.
depende de (requisitos)	Relação entre **requisitos**, indicando quais **requisitos** dependem de outros. Assim, é possível identificar quais **requisitos** podem sofrer impacto quando da ocorrência de mudanças em um **requisito** específico.
deriva	Relação entre **item de configuração** e **requisito,** indicando que um determinado **requisito** está sob gerência de configuração, ou seja, torna-se um **item de configuração**.
determina a abrangência de	Relação entre **escopo** e **projeto** que indica que um dado **escopo** determina e delimita a abrangência de um dado **projeto.**

Tabela 4.1 – Dicionário de Termos da Ontologia de Requisitos de Software (continuação).

Documento de Especificação de Requisitos de Software	É um **documento**, portanto um **artefato** de software, formal que é utilizado para comunicação dos requisitos aos clientes, aos usuários e demais interessados, servindo de base para diversas **atividades** do **processo** de software. Sendo um documento, é normalmente associado a padrões organizacionais (**roteiros**) que definem a forma como deve ser produzido.
é definido para	Relação entre **requisito** e **projeto**, indicando que um requisito é definido no âmbito de um projeto.
é responsável por	Relação entre **requisito** e **pessoa** que indica quais são os responsáveis por um requisito, ou seja, que **pessoas** responderão pela criação e por alterações nos requisitos associados a eles.
envolve (artefato)	Relação entre **contexto** e **artefato**, indicando quais foram os **artefatos** analisados em um dado **contexto** e podem ter sugerido a criação de um **requisito**, fazendo parte, portanto, do **contexto** de **origem** do **requisito**.
envolve (atividade)	Relação entre **contexto** e **atividade,** indicando qual **atividade** do **processo** de software em que se deu um determinado **contexto**, no qual um **requisito** pode ter sido originado, fazendo parte, portanto, do **contexto** de **origem** do **requisito**.
envolve (pessoa)	Relação entre **contexto** e **pessoa,** indicando quais **pessoas** estiveram envolvidas em um determinado **contexto**, no qual um **requisito** pode ter sido originado, fazendo parte, portanto, do **contexto** de **origem** do **requisito**.
Escopo	Determina os limites de um **projeto**, englobando o que faz parte do **projeto** e o que não faz. Um escopo pode ser decomposto em **módulos**.
estado	Propriedade de um **requisito** que indica qual a sua situação atual, ou seja, se está aprovado ou rejeitado, se está implementado etc. São exemplos de valores para essa propriedade: Cancelado, Rejeitado, Aprovado, Proposto, Implementado etc.

Tabela 4.1 – Dicionário de Termos da Ontologia de Requisitos de Software (continuação).

Módulo	É uma porção de sistema utilizada para organizar e agrupar, por exemplo, **artefatos**, **requisitos** e outros módulos que possuem certa afinidade.
Origem	Relação entre **contexto** e **requisito**, que indica qual foi o contexto no qual teve origem um requisito.
Requisito	Representa especificações de serviços que o sistema deve prover, restrições do sistema e do processo de software e conhecimento necessário para a construção do sistema.
sub-módulos	Papel da relação de agregação entre **módulos**, indicando quais os **módulos** são parte de (ou foram decompostos) um de dado **módulo**.
sub-tipos	Papel da relação de agregação entre **tipos de requisitos**, indicando quais os **tipos de requisitos** são parte de um de dado **tipo de requisito**.
super-módulo	Papel da relação de agregação entre **módulos**, indicando a partir de qual **módulo**, um dado **módulo** foi decomposto.
super-tipos	Papel da relação de agregação entre **tipos de requisitos**, indicando a partir de quais **tipos de requisitos**, um dado **tipo de requisito** foi decomposto.
tem interesse em	Relação entre **requisito** e **pessoa** que indica quais são as pessoas interessadas em um requisito, de forma que, se alguma alteração ou negociação for necessária, eles deverão ser contatados.
tem origem em	Relação entre **requisito** e **contexto** que indica em qual **contexto** (situação, circustância) foi identificado um dado **requisito**.
Tipo de Requisito	Elemento utilizado na caracterização de um requisito, buscando retratar a dimensão que o requisito procura abordar, tal como requisito de usuário e de sistema, requisito de segurança, desempenho etc.
trata	Relação entre **requisito** e **artefato**, indicando que um requisito é descrito, modelado, implementado etc em um **artefato**. Por meio dessa relação pode-se também trabalhar rastreabilidade de **requisitos** nos **artefatos** relacionados.

As Tabelas 4.2, 4.3, 4.4, 4.5 e 4.6 apresentam o significado de alguns dos termos das ontologias reutilizadas. Apenas os termos utilizados diretamente na construção da Ontologia de Requisitos são apresentados.

Tabela 4.2 – Dicionário de Termos Parcial da Ontologia de Processo de Software (BERTOLLO, 2006)

Artefato	Um **insumo** para, ou um **produto** de, uma **atividade**, no sentido de ser um objeto de transformação da atividade. Em função de sua natureza, artefatos podem ser classificados em: **documentos**, **diagramas**, **artefatos de código**, **componentes de software** ou **produtos de software**.
Atividade	Ação que transforma **artefatos** de entrada (**insumos**) em **artefatos** de saída (**produtos**). Em função de sua natureza, atividades podem ser classificadas em **atividades de gerência**, **atividades de construção** e **atividades de avaliação da qualidade**.

Tabela 4.3 – Dicionário de Termos Parcial da Ontologia de Artefato (NUNES, 2005)

Documento	**Artefato** de software não passível de execução, constituído tipicamente de declarações textuais, normalmente associado a padrões organizacionais (**roteiros**) que definem a forma em que eles devem ser produzidos. Ex.: documento de especificação de requisitos, plano de projeto, plano de qualidade, relatório de avaliação da qualidade, etc.

Tabela 4.4 – Dicionário de Termos Parcial da Ontologia de Gerência de Configuração de Software (NUNES, 2005)

Acesso	Relação entre **item de configuração** e **recurso humano**, indicando qual **tipo** de acesso (leitura, alteração, exclusão, etc.) determinado **recurso humano** tem sobre determinado **item de configuração**.
Alteração	Representa uma solicitação de alteração efetuada por um **recurso humano**, indicando as **variações de itens de configuração** que poderão ser alteradas. Esta alteração, sendo aprovada, deverá ser executada por **recursos humanos**, dando origem a novas **variações**. Para um controle formal, deverão ser executados procedimentos de ***check-out*** e ***check-in***.
Check-in	Propriedade de **alteração** que indica quando as **variações** submetidas para uma **alteração** foram devolvidas ao repositório central, estando disponíveis para novas **alterações**. Contém a data/hora em que as **variações** tornaram-se disponíveis novamente.
Check-out	Propriedade de **alteração** que indica quando cópias das **variações** submetidas para **alteração** foram retiradas do repositório central e disponibilizadas na área de trabalho do desenvolvedor, para que sejam manipuladas. Contém a data/hora em que as **variações** foram retiradas do repositório central. A partir deste momento, nenhum outro **recurso humano** poderá alterar estas **variações** até que se tenha realizado um procedimento de ***check-in***.
depende de	Relação entre **variações** indicando as **variações** que são dependentes de uma determinada **variação**. Assim, é possível identificar todas as **variações** que podem ser afetadas em uma determinada **alteração** e não apenas as solicitadas.
deriva	Relação entre **item de configuração** e **artefato** (ou entre **item de configuração** e **ferramenta de software** ou entre **item de configuração** e **requisito**), indicando que um determinado **artefato** (ou **ferramenta de software** ou **requisito**) está sob gerência de configuração, ou seja, torna-se um **item de configuração**.
Estado	Relação entre **item de configuração** e **variação** que indica quais as **variações** de um certo **item de configuração**.

Tabela 4.4 – Dicionário de Termos Parcial da Ontologia de Gerência de Configuração de Software (NUNES, 2005) (continuação)

é submetida para	Relação entre **variação** e **alteração**, indicando quais **variações** foram submetidas para modificação em uma determinada **alteração**.
executa	Relação entre **recurso humano** e **alteração,** indicando quais **recursos humanos** são responsáveis pela execução de uma determinada **alteração**, ou seja, quais **recursos humanos** estão (ou estiveram) de posse de determinadas **variações** para, possivelmente, realizar **alterações**.
Item de Configuração	Item que está sob gerência de configuração e, assim, só pode ser alterado segundo um procedimento de controle de alteração formalmente estabelecido e documentado. Pode possuir várias **variações**. Pode ser uma **ferramenta de software** ou um **artefato**, como, por exemplo, um determinado plano de projeto ou um certo artefato de código.
Linha Base	Conjunto de **itens de configuração** em determinadas **variações**, que serve de base para o desenvolvimento ulterior. Ex.: uma linha base formada pela porção de código X (variação 2.0), pelo documento de plano de projeto Z (variação 1.1.1), pelo diagrama de casos de uso (variação 1.0) etc.
resulta em	Relação entre **variação** e **alteração**, indicando quais **variações** foram produzidas em uma determinada **alteração**, ou seja, quais as **variações** resultantes de uma determinada **alteração**.
solicita	Relação entre **recurso humano** e **alteração**, indicando quais **recursos humanos** fizeram a solicitação de uma determinada **alteração**.
Sub-variação	Papel da relação de agregação entre duas **variações** v_1 e v_2. Se v_2 é parte de v_1, então v_2 é dita uma sub-variação de v_1
Super-variação	Papel da relação de agregação entre duas **variações** v_1 e v_2. Se v_1 é decomposta em outras **variações**, dentre eles v_2, então v_1 é dita um super-variação de v_2.
Variação	Indica qual a **versão** ou **variante** de um determinado **item de configuração**. É caracterizada por um **número** único para cada **variação** de um mesmo **item de configuração**.

Tabela 4.4 – Dicionário de Termos Parcial da Ontologia de Gerência de Configuração de Software (NUNES, 2005) (continuação).

Variante	Estado em que um **item de configuração** existe simultaneamente em duas ou mais formas diferentes que atendam a requisitos similares. Ex.: Um documento estava na versão 1.0 e após alterações passou para a versão 1.1. Porém, foi criada a variante 2.0 que, apesar de atender aos mesmos requisitos que a versão 1.1, foi construída de forma diferente.
Versão	Estado em que se encontra um **item de configuração**. Cada **alteração** realizada em um **item de configuração** gera uma nova versão deste **item**. Ex.: Um documento que estava na versão 1.0 e após algumas alterações passou para versão 1.1.

Tabela 4.5 – Dicionário de Termos Parcial da Ontologia de Qualidade de Software (DUARTE, 2001)

Característica de qualidade	Atributos de um **artefato** ou de um **processo de software** através dos quais a qualidade de um software ou de um processo pode ser avaliada.
Característica de qualidade de produto	**Característica de qualidade** relevante para a qualidade de um **artefato**. Quando for uma **característica indiretamente mensurável**, só pode ser medida a partir de outras características de produto. **Ex**: Tamanho, legibilidade, adequabilidade.
Característica de qualidade diretamente mensurável	**Característica de qualidade** que pode ser medida diretamente através de uma **métrica**, sem a necessidade de combinar medidas de sub-características. **Ex**: tamanho, velocidade.
Característica de qualidade indiretamente mensurável	**Característica de qualidade** que só pode ser medida combinando medidas de sub-características. **Ex**: funcionalidade, manutenibilidade.

Tabela 4.5 – Dicionário de Termos Parcial da Ontologia de Qualidade de Software (DUARTE, 2001) (continuação)

correlaciona-se com	Relação entre os conceitos **métrica** e **característica diretamente mensurável**, indicando que uma **métrica** está correlacionada a uma **característica diretamente mensurável**.
é valorada por	Relação entre **características indiretamente mensuráveis** e **características de qualidade**, indicando que uma **característica indiretamente mensurável** pode ser valorada através de outras **características de qualidade**, direta ou indiretamente mensuráveis.
é relevante para	Relação entre os conceitos **característica de qualidade de produto** e **artefato**, indicando que uma determinada **característica de qualidade** é relevante para a qualidade de um dado **artefato**.
Medida	Resultado de uma medição de uma **métrica**. **Ex**: 300 linhas de código.
Métrica	É o que se deseja medir. **Ex**: número de linhas de código.
quantifica	Relação entre os conceitos **medida** e **métrica**, indicando que uma **medida** quantifica uma **métrica**.

Tabela 4.6 – Dicionário de Termos Parcial da Ontologia de Organizações de Software (RUY, 2006)

Pessoa	Indivíduo fundamental ao funcionamento de uma organização, atuando na execução de atividades necessárias ao comprimento da missão da organização. Ao longo de sua trajetória profissional, pessoas acumulam competências, disponibilizando-as para a organização em que trabalham. As competências possuídas pelos profissionais de uma organização são de grande importância para os próprios profissionais, pois são utilizadas para estabelecer o seu papel e seu valor na organização, e também para a organização, pois representam o seu capital intelectual.

Tabela 4.6 – Dicionário de Termos Parcial da Ontologia de Organizações de Software (RUY, 2006) (continuação)

Projeto	Empreendimento requerendo esforço organizado (**processo**) visando ao desenvolvimento ou manutenção de um **produto de software** específico. Compreende um conjunto gerenciado de **recursos** inter-relacionados que entrega **artefatos** a um cliente ou usuário final e tipicamente opera de acordo com um plano. Ex: projeto da vídeo locadora Passatempo.
Equipe	Agrupamento de pessoas com finalidade determinada e, normalmente, por período limitado. Equipes podem estar relacionadas a projetos (por exemplo: equipe de desenvolvimento, equipe de analistas) ou somente à organização (equipe administrativa, equipe de testes, equipe de auditores).

4.6 Considerações do Capítulo

Este capítulo apresentou a Ontologia de Requisitos de Software desenvolvida visando à formalização do conhecimento acerca do domínio de requisitos, sobretudo, para apoiar o desenvolvimento de ferramentas para o ambiente ODE. Essa ontologia foi definida buscando seguir o critério de compromissos ontológicos mínimos (GRUBER, 1995), de modo que seja possível reutilizar esse conhecimento de várias formas no ambiente, tais como na comunicação entre pessoas e o ambiente, entre agentes de software atuando no ambiente, na construção e integração de ferramentas, no aprendizado de Engenharia de Requisitos dentro do ambiente etc. Além disso, uma vez que diversas ontologias já foram construídas no contexto do projeto ODE, procurou-se sempre reutilizar conceituações previamente estabelecidas, favorecendo a formação de uma rede efetiva de conceitos e relações que possa ser explorada de várias formas.

Capítulo 5

Apoio à Engenharia de Requisitos e ao Reúso de Itens de Conhecimento

O objetivo deste capítulo é reunir o que foi discutido nos capítulos anteriores e propor uma abordagem de reúso do conhecimento envolvido na Engenharia de Requisitos (ER), no contexto de ODE (*Ontology-based Software Development Environment*), um ambiente de desenvolvimento de software centrado em processo, baseado em ontologias e que possui uma infra-estrutura de Gerência de Conhecimento integrada. Dessa maneira, objetiva-se mostrar como foi utilizada a Ontologia de Requisitos de Software para a construção de uma ferramenta de apoio à Engenharia de Requisitos (*ReqODE*) e como a infra-estrutura de Gerência de Conhecimento de ODE foi utilizada para a construção de serviços capazes de apoiar a implementação da abordagem proposta. Ademais, em relação ao reúso de itens de conhecimento, este capítulo destaca a reutilização de ontologias, padrões de análise e modelos de objetos de projeto similares no contexto da ER.

5.1 Introdução

Ao longo deste trabalho se falou sobre Engenharia de Requisitos (ER), reutilização, ontologias, padrões de análise, Gerência de Conhecimento (GC) e Ambientes de Desenvolvimento de Software (ADS). Tais assuntos foram tratados tanto isoladamente quanto de forma integrada, procurando explorar como eles poderiam ser combinados com o objetivo de facilitar o desenvolvimento de software.

Este capítulo, por sua vez, se propõe a reunir esses elementos em uma abordagem de apoio à Engenharia de Requisitos baseada em reutilização, fundamentada sobre dois pilares: (i) o apoio automatizado a atividades do processo de ER e (ii) serviços de Gerência de Conhecimento visando ao reúso de itens de conhecimento no contexto da ER.

O primeiro aspecto é tratado por meio de uma ferramenta de requisitos chamada *ReqODE*, a qual é construída com base na Ontologia de Requisitos de Software apresentada no Capítulo 4. Vale destacar que, conforme discutido no Capítulo 3, a integração de ferramentas em um ADS é um aspecto essencial. Tal fato foi trabalhado no sentido de garantir que *ReqODE* estivesse integrada às outras ferramentas de ODE, dentre elas as ferramentas de alocação de recursos e de apoio ao planejamento da documentação. Na busca por essa

integração, as ontologias do ambiente, as quais servem de base conceitual para os modelos das ferramentas, tiveram um papel muito importante.

No que se refere ao apoio de serviços focando o reúso no contexto da ER, utilizou-se a infra-estrutura de GC de ODE. Uma vez que tal infra-estrutura faz parte do ambiente, a integração desses serviços a *ReqODE* foi facilitada. Assim, no contexto da ER, foram abordados itens de conhecimento já reutilizados em ODE (artefatos, lições aprendidas e pacotes de mensagens) e, além disso, outros itens de conhecimento passaram a ser considerados, a saber ontologias, padrões de análise e modelos de objetos, de modo que os engenheiros de requisitos passassem a ter a possibilidade de reutilizar tais itens.

Ademais, a relevância dessa forma de reúso foi reforçada quando, neste capítulo (seção 5.2), tratou-se da construção de *ReqODE*.

Este capítulo apresenta a abordagem proposta neste trabalho, sendo composto de outras duas seções. A seção 5.2 trata da construção de *ReqODE*. Inicialmente, descreve-se, passo-a-passo, o processo utilizado para derivar, a partir da Ontologia de Requisitos de Software, os modelos de classes e de casos de uso utilizados na construção da ferramenta. Na seqüência, a ferramenta é apresentada, por meio de suas telas, dando uma idéia geral de seu funcionamento. A seção 5.3 trata do apoio ao reúso de itens de conhecimento no contexto da Engenharia de Requisitos, destacando os modelos relacionados à construção das funcionalidades desenvolvidas para implementar alguns serviços de GC especializados para a ER. São apresentadas, ainda, algumas telas com o objetivo de ilustrar o funcionamento de tais funcionalidades, destacando a integração com *ReqODE*.

5.2 *ReqODE* – Uma Ferramenta de Apoio à Engenharia de Requisitos

Muitos dos modelos de qualidade (CMMI e MPS.BR, por exemplo) apontam que um dos primeiros aspectos a serem tratados na busca pela qualidade do processo é a gerência de requisitos. Assim, para que ODE pudesse apoiar efetivamente as organizações de software em seus esforços de melhoria de processos, ele deveria fornecer apoio ao trabalho com requisitos. A partir daí, surgiu a necessidade de se construir uma ferramenta de apoio à Engenharia de Requisitos, denominada *ReqODE*.

Para que *ReqODE* fosse construída segundo os princípios de ODE, ela deveria ser baseada nas ontologias do ambiente e estruturada segundo a Arquitetura Conceitual de ODE, discutida no Capítulo 3. Para tanto, foi construída a Ontologia de Requisitos de Software, apresentada no Capítulo 4, a qual está integrada às outras ontologias do ambiente e que serviu de base para a construção de *ReqODE*. Além disso, para estar em conformidade com a Arquitetura Conceitual de ODE, foi necessário organizar os elementos de *ReqODE* segundo o nível Ontológico, o nível de Conhecimento e o nível Base. Assim, iniciou-se o trabalho de construção da ferramenta.

5.2.1 A Construção de *ReqODE*

A elaboração dos modelos de análise e de requisitos de *ReqODE* teve como base a Ontologia de Requisitos de Software. Essa ontologia forneceu uma conceituação inicial, possibilitando uma modelagem preliminar das classes e dos casos de uso. Além disso, vale ressaltar que as restrições existentes nos modelos de classe foram derivadas dos axiomas definidos na Ontologia de Requisitos de Software.

Dessa forma, a modelagem de *ReqODE* foi construída em três etapas: (1) Geração do Modelo de Classes Preliminar, (2) Geração do Modelo de Casos de Uso Preliminar e (3) Refinamento do Modelo de Classe e do Modelo de Caso de Uso. Tais etapas são tratadas em mais detalhes a seguir.

Etapa 1 - Geração do Modelo de Classes Preliminar

Uma vez que *ReqODE* está no contexto de ODE e que a Ontologia de Requisitos de Software foi construída de forma integrada às outras ontologias do ambiente, na geração de um modelo de classes preliminar não basta apenas derivar classes e associações a partir de conceitos e relações, respectivamente. É necessário derivar esses elementos com base na Arquitetura Conceitual de ODE. Para tanto, se faz imprescindível atentar para duas questões: (i) é necessário fazer o mapeamento dos novos conceitos, relações, propriedades e axiomas da Ontologia de Requisitos de Software para a Arquitetura de ODE; (ii) como a Ontologia de Requisitos de Software reutiliza conceitos de outras ontologias de ODE que já foram derivadas em modelos de classes, é necessário identificar em quais níveis (Nível de Conhecimento ou Nível Base) cada uma dessas classes foi derivada e integrá-las com as novas classes a serem derivadas da Ontologia de Requisitos de Software.

Inicialmente, identificaram-se quais conceitos da Ontologia de Requisitos de Software poderiam fazer parte do Nível de Conhecimento. Após esse trabalho, constatou-se que apenas o conceito *TipoRequisito* teria essa propriedade de representar um conhecimento, ou seja, uma organização de software poderia cadastrar em sua base de conhecimento quais são os tipos de requisitos a serem utilizados em seus projetos. Sendo assim, modelou-se a classe *KTipoRequisito*[6.1], como mostra a Figura 5.1. A classe *KTipoRequisito* herda de Conhecimento, que é super-classe de toda classe de ODE que representa um objeto de conhecimento.

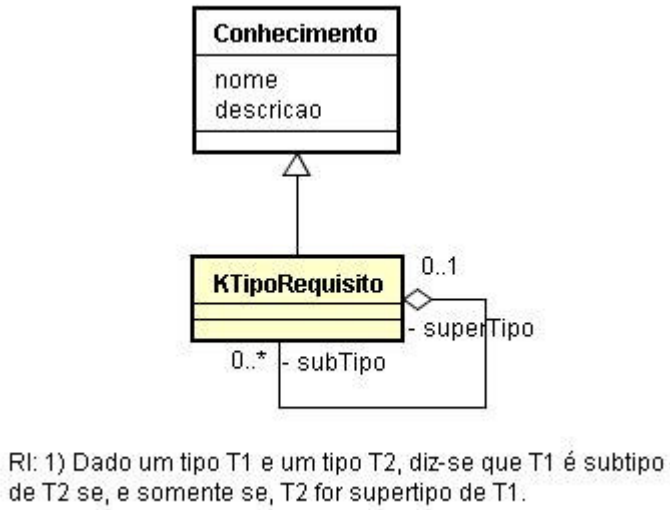


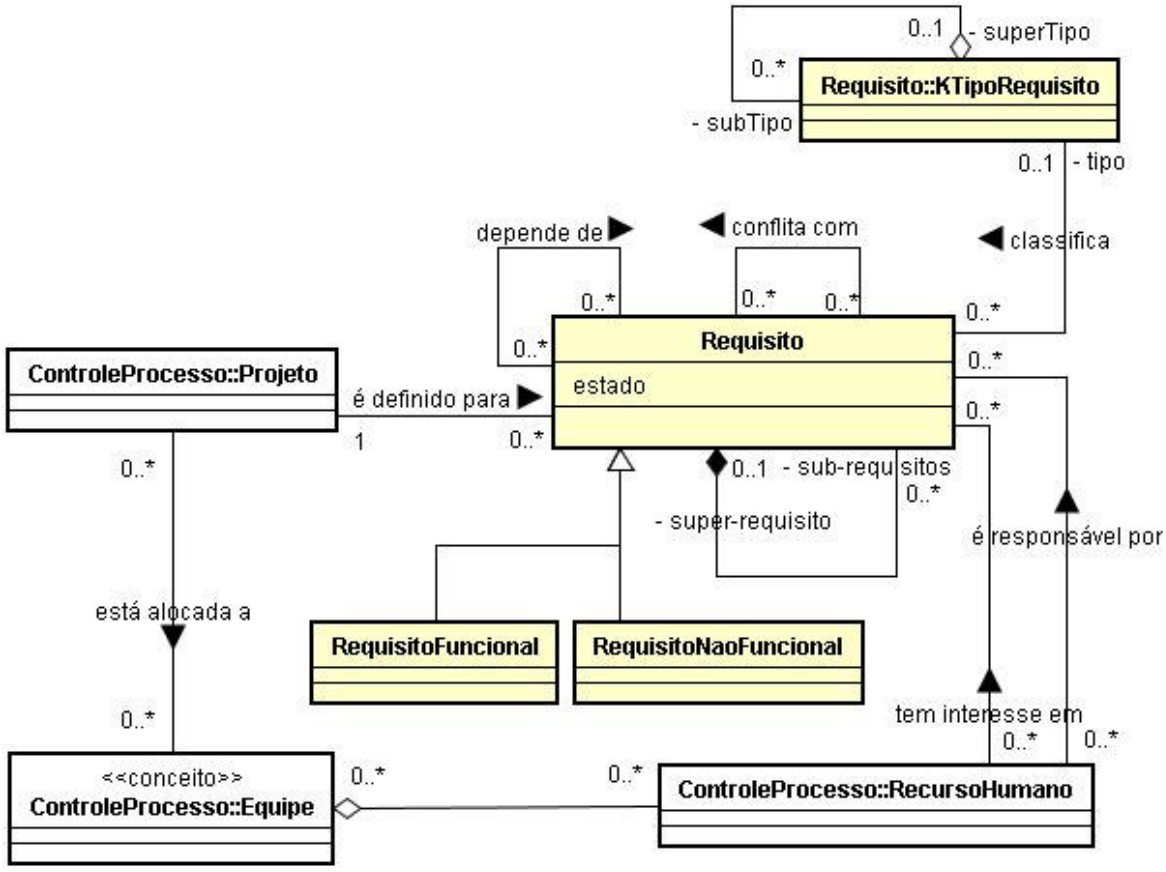
Figura 5.1 - Modelagem de *KTipoRequisito*

Depois de definido o que é conhecimento, passou-se à derivação das classes do Nível Base. Sendo assim, iniciou-se pelo conceito *Requisito* que deu origem à classe de mesmo nome.

As relações de *composição* (relação todo-parte entre requisitos)*, dependência* (relação *depende de*) e *conflito* (relação *conflita com*) foram mapeadas no modelo de classes assim como estavam na Ontologia de Requisitos de Software. O mesmo aconteceu com as relações com *Projeto* (relação *é definido para*) *e TipoRequisito* - agora *KTipoRequisito* – (relação *classifica*). No entanto, o conceito *Pessoa*, oriundo da Ontologia de Organizações de Software (RUY, 2006), foi mapeado para a classe *RecursoHumano*, a qual representa pessoas desempenhando papéis de recursos humanos no processo de software. Essa decisão foi tomada por RUY (2006), porque no contexto de ODE, esse mapeamento já era feito dessa forma em uma versão anterior, tomando por base apenas a ontologia de processo de software,

[6.1] Em ODE utiliza-se o prefixo K para as classes de conhecimento, a fim de facilitar a identificação.

evitando-se a necessidade de muitas alterações. Dessa forma, tem-se que os interessados e o responsável por um requisito são recursos humanos relacionados à equipe do projeto no qual o requisito foi definido. Esse modelo é mostrado na Figura 5.2.



RI : 1) O recurso humano responsável por um requisito deve estar alocado a uma equipe do projeto no qual o requisito foi definido.

RI : 2) Os recursos humanos interessados em um requisito devem estar alocados a um equipe do projeto no qual o requisito foi definido.

RI : 3) Dados três requisitos R1, R2 e R3, se R1 depende de R2 e R2 depende de R3, então R1 depende de R3.

RI : 4) Dados dois requisitos R1 e R2, se R1 conflita com R2, então, R2 conflita com R1.

RI : 5) Dados três requisitos R1, R2 e R3, se R1 depende de R2 e R1 compõe R3, então R3 depende de R2.

RI : 6) Dados dois requisitos R1 e R2, se R1 compõe R2, então, R2 depende de R1.

Figura 5.2 - Modelagem inicial de Requisito no Nível Controle

Quanto às classes derivadas de *Escopo* e *Modulo* e seus relacionamentos, os mesmos se mantiveram como os respectivos conceitos e as relações da Ontologia de Requisito de Software, como mostra a Figura 5.3.

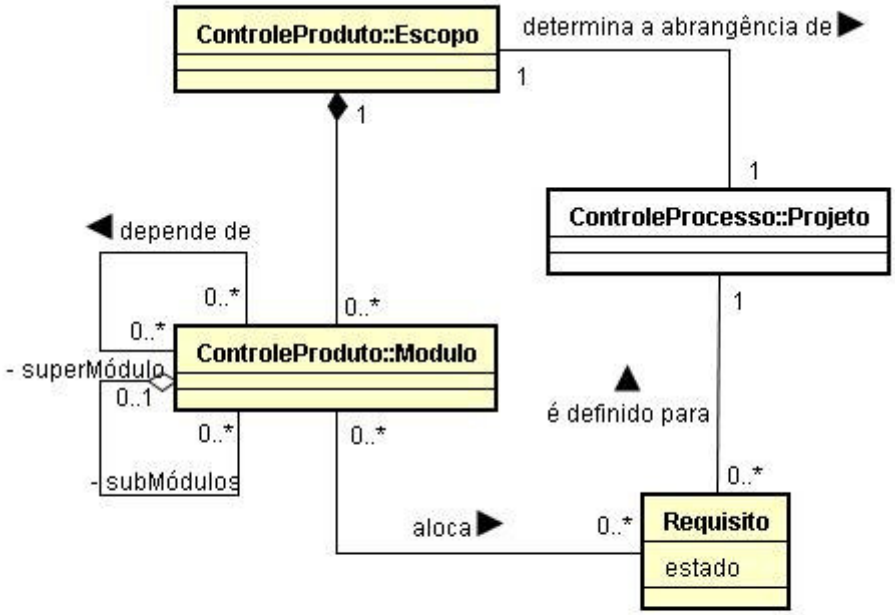


Figura 5.3 - Modelagem de Requisito e Módulo

O modelo tratando da origem de um requisito é mostrado na Figura 5.4. Ele se manteve bem próximo ao correspondente na Ontologia de Requisitos de Software, exceto pelo fato do conceito *Pessoa* ter sido mapeado para a classe *RecursoHumano*, conforme citado anteriormente.

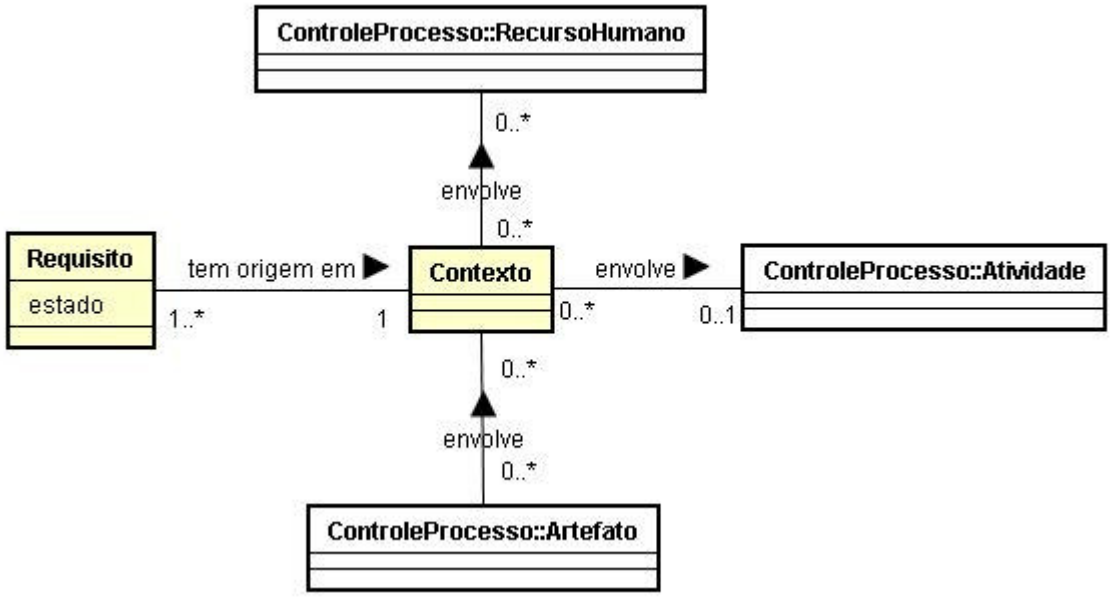


Figura 5.4 - Contexto de Surgimento de um Requisito

No que tange ao tratamento de requisitos em artefatos, da mesma forma que foi modelado na Ontologia de Requisitos de Software, manteve-se no modelo de classes, como mostra a Figura 5.5. Tem-se, portanto, que um requisito pode ser tratado por um Documento de Especificação de Análise, um Documento de Especificação de Requisitos, um Diagrama de Casos de Uso, um Plano de Riscos etc, os quais são todos artefatos.

Finalmente, em relação aos aspectos de Gerência de Configuração e de Qualidade, tratados pela Ontologia de Requisitos de Software, derivaram-se modelos de classes muito semelhantes aos modelos dessa ontologia. No entanto, tais modelos não constam nesta seção pelo fato de não terem sido contemplados na construção de *ReqODE*. Isso porque a ferramenta de Gerência de Configuração e as funcionalidades relativas a características de qualidade não estão integradas à versão atual do ambiente, o que impossibilita sua utilização no contexto deste trabalho.

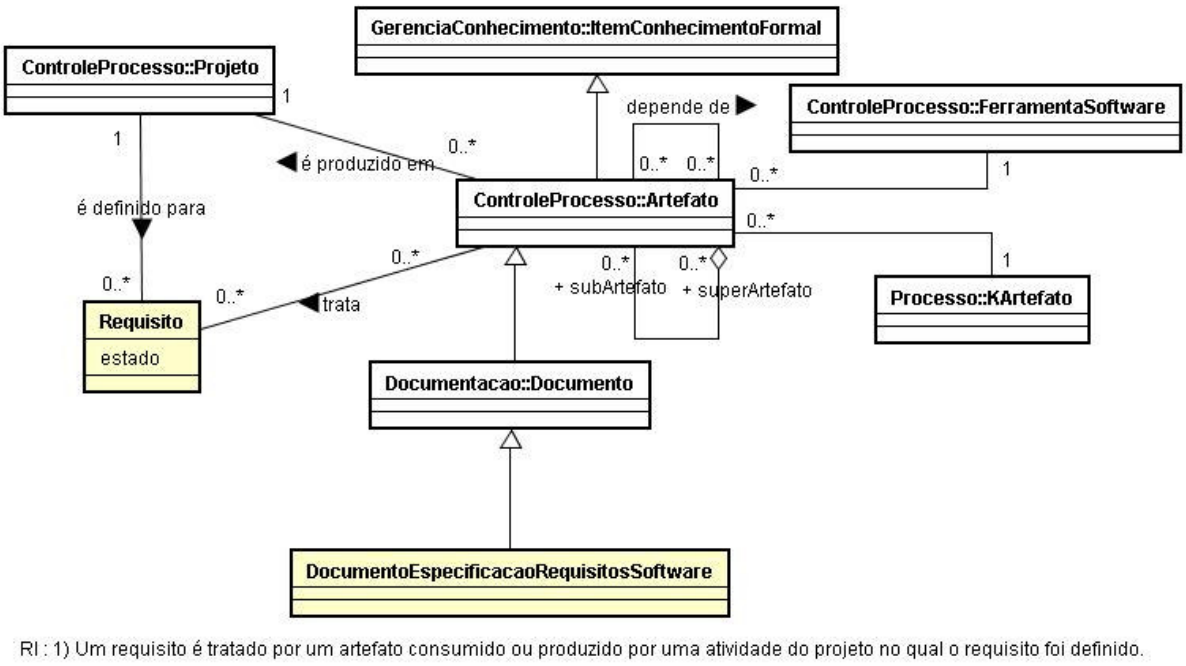


Figura 5.5 - Modelagem de Tratamento de Requisito em Artefatos

Etapa 2 - Geração do Modelo de Casos de Uso Preliminar

A partir do modelo de classes preliminar foi possível listar alguns casos de uso que teriam responsabilidades básicas de manipular objetos das classes derivadas e seus relacionamentos. Daí gerou-se o diagrama de casos de uso da Figura 5.6. Nesse diagrama são contemplados dois atores: Engenheiro de Conhecimento, que lida com os itens de conhecimento existentes no ambiente e o Engenheiro de Requisitos, que atua trabalhando com requisitos.

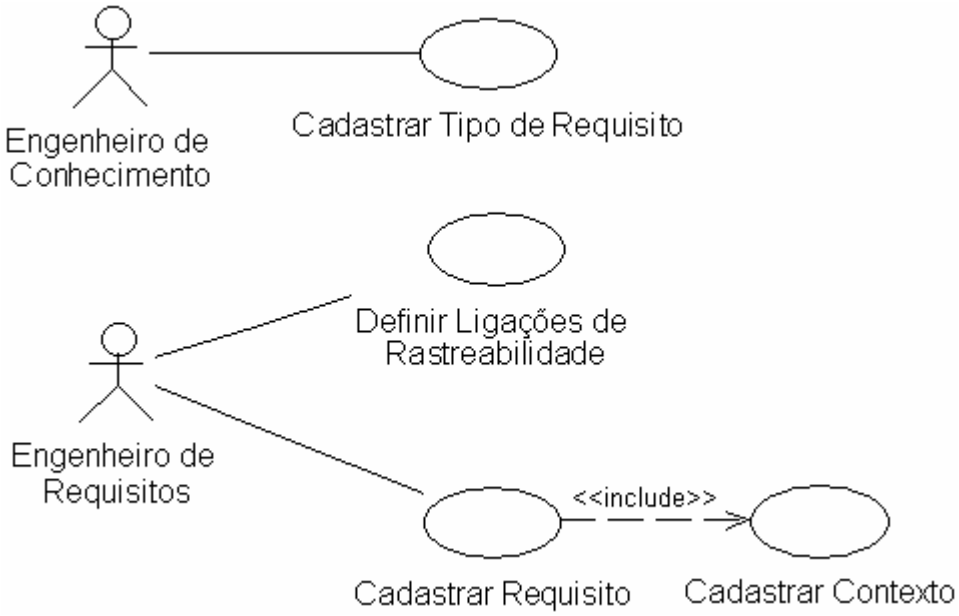


Figura 5.6 – Modelo de Casos de Uso Preliminar

Os casos de uso Cadastrar Tipo de Requisito, Cadastrar Contexto, Cadastrar Requisito e Definir Ligações de Rastreabilidade são descritos a seguir:

- **Cadastrar Tipo de Requisito**

Este caso de uso permite que o Gerente de Conhecimento cadastre tipos de requisitos que são utilizados na caracterização dos requisitos. Esses tipos são, portanto, conhecimento organizacional que é utilizado na condução dos projetos.

- **Cadastrar Requisito**

Este caso de uso permite controlar os requisitos de um projeto, oferecendo serviços para criar um requisito, alterar dados de um requisito, definir a origem de um requisito a partir de um contexto e excluir um requisito.

- **Cadastrar Contexto**

Este caso de uso permite definir contextos, incluindo serviços para criar, alterar e excluir um contexto. Uma vez definido um contexto, ele pode, por exemplo, ser utilizado na definição da origem de um requisito.

- **Definir Ligações de Rastreabilidade**

Este caso de uso permite definir ligações a serem utilizadas para manter a rastreabilidade tanto entre requisitos quanto entre requisitos e artefatos gerados ao longo do processo de software.

Etapa 3 - Refinamento do Modelo de Classes e do Modelo de Casos de Uso

Nesta etapa, aspectos específicos relacionados ao modo de atuação da ferramenta foram abordados. Nesse sentido, as multiplicidades dos relacionamentos foram revistas, os atributos das classes foram definidos, a modelagem de casos de uso foi reavaliada, os casos de uso foram descritos e tanto os casos de uso quanto as classes foram organizados em pacotes segundo a estrutura do ambiente.

Neste momento, contou-se com o apoio da VixTeam Consultoria & Sistemas, empresa parceira do LabES (Laboratório de Engenharia de Software), que atuou como cliente na modelagem de *ReqODE* fornecendo requisitos para a ferramenta de modo que ela fosse capaz de apoiar o processo de Gerência de Requisitos segundo o modelo de qualidade CMMI, uma vez que tal empresa está buscando tal certificação e espera contar com o apoio de *ReqODE* para tanto.

Em relação ao refinamento do modelo de classes, pode-se citar que a alteração da multiplicidade do papel *superTipo* da auto-associação da classe *KTipoRequisito* foi alterada, passando de 0..n a 0..1. Isso porque chegou-se à conclusão de que, na prática, não são muito usados tipos de requisitos derivados de mais de um tipo, ou seja, todo tipo de requisito passa a ter no máximo um supertipo.

Além disso, outras multiplicidades foram alteradas, como aquelas da associação entre as classes *Requisito* e *Contexto*, que passaram de 1 para 0..1 e de 1..n para 0..n. No primeiro caso, objetiva-se permitir que um requisito possa ou não ter definido o contexto no qual ele foi identificado. Já no segundo caso, objetiva-se representar que um contexto pode ter sido criado e não ter sido relacionado a um requisito. Essas decisões foram tomadas para dar mais flexibilidade na utilização de *ReqODE*, pois nem sempre o Engenheiro de Requisitos deseja definir o contexto de surgimento de um requisito no momento em que o requisito é criado ou nem sempre um contexto é relacionado a um requisito no momento em que esse contexto é criado.

No que tange à definição de atributos, destacam-se a definição dos atributos *nome* e *descrição* da classe *Módulo*, do atributo *dscMiniMundo* (descrição do contexto geral referente ao escopo do sistema a ser construído) da classe *Escopo* e dos atributos *id, sentença, dtCriacao, prioridade, estado, razoesCriacao* e *comentariosGerais* na classe *Requisito.*

Na construção da Ontologia de Requisitos de Software procurou-se não modelar aspectos relacionados a uma tecnologia ou técnica específica. Isso com a finalidade de deixá-

la o mais reutilizável possível, mesmo porque, decisões de quais tecnologias ou procedimentos utilizar concerne ao Gerente de Projeto durante a instanciação de um processo para um projeto específico. Ou seja, tais decisões podem não valer para todos os processos.

No entanto, é fato que algumas técnicas são utilizadas largamente por uma porção considerável de projetos. Um exemplo disso é a Técnica de Modelagem de Casos de Uso. Grande parte das empresas de software utiliza casos de uso para refinar requisitos funcionais e para relacioná-los a requisitos não-funcionais. Dessa forma, optou-se por criar um relacionamento entre a classe *Requisito* e a classe *CasoUso*, a qual já existia no contexto de ODE, como mostra a Figura 5.7. Assim, um requisito pode ser relacionado a vários casos de uso que, por sua vez, podem estar relacionados a mais de um requisito.

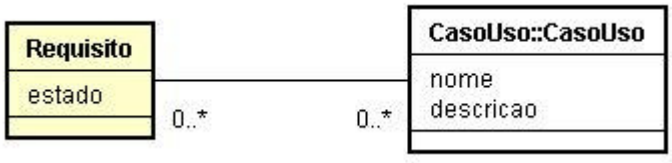


Figura 5.7 – Modelagem de Requisito e Caso de Uso

Como um dos objetivos deste trabalho é favorecer a reutilização no contexto da Engenharia de Requisitos, outra decisão importante foi o fato de se considerar um requisito como um Item de Conhecimento Formal. Um Item de Conhecimento Formal na infra-estrutura de Gerência de Conhecimento de ODE representa um elemento produzido ao longo do processo de software de maneira sistemática. Dessa forma, por meio de uma especialização entre as classes *Requisito* e *ItemConhecimentoFormal* foi possível permitir que um requisito seja trabalhado pelos serviços de Gerência de Configuração de ODE, visando, assim, ao seu reúso. Tal modelo é exibido na Figura 5.8.

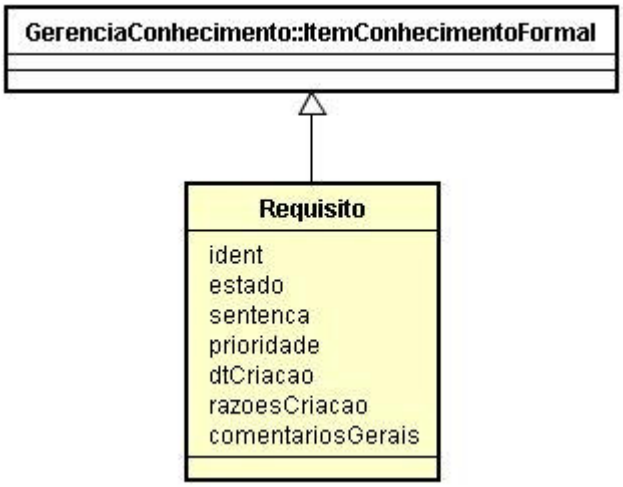


Figura 5.8 - Modelagem de Requisito com Item de Conhecimento Formal

Em relação ao refinamento do modelo de casos de uso previamente elaborado para *ReqODE*, foi criado mais um caso de uso que tem por objetivo gerar relatórios de rastreabilidade dos requisitos de um projeto. Esses relatórios são de extrema importância no momento de uma alteração, pois fornecem uma visão geral de quais elementos podem ser impactados pela alteração. O modelo final de casos de uso de *ReqODE* é mostrado, então, na Figura 5.9. O caso de uso *Gerar Relatório de Rastreabilidade* é descrito a seguir.

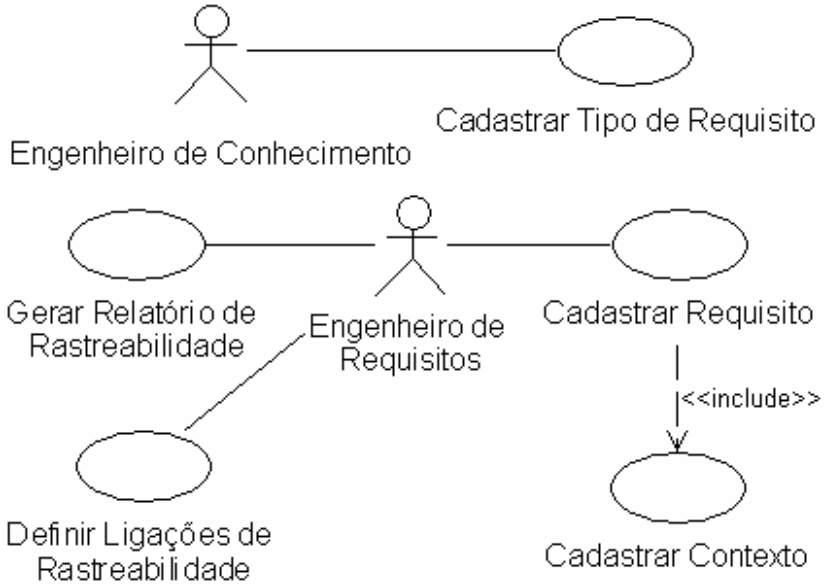


Figura 5.9 – Modelagem Final de Casos de Uso de *ReqODE*

- **Gerar Relatórios de Rastreabilidade**

Este caso de uso permite a geração de relatórios de rastreabilidade fornecendo informações acerca das ligações existentes tanto entre requisitos quanto entre requisitos e outros elementos

do processo de desenvolvimento, permitindo que análises de impacto de alterações sejam feitas.

Para facilitar a visualização das semelhanças entre a modelagem ontológica e a modelagem de classes final, nas Figuras 5.10 e 5.11 são exibidos, respectivamente, um diagrama simplificado da Ontologia de Requisitos de Software e o Modelo de Classes de *ReqODE*.

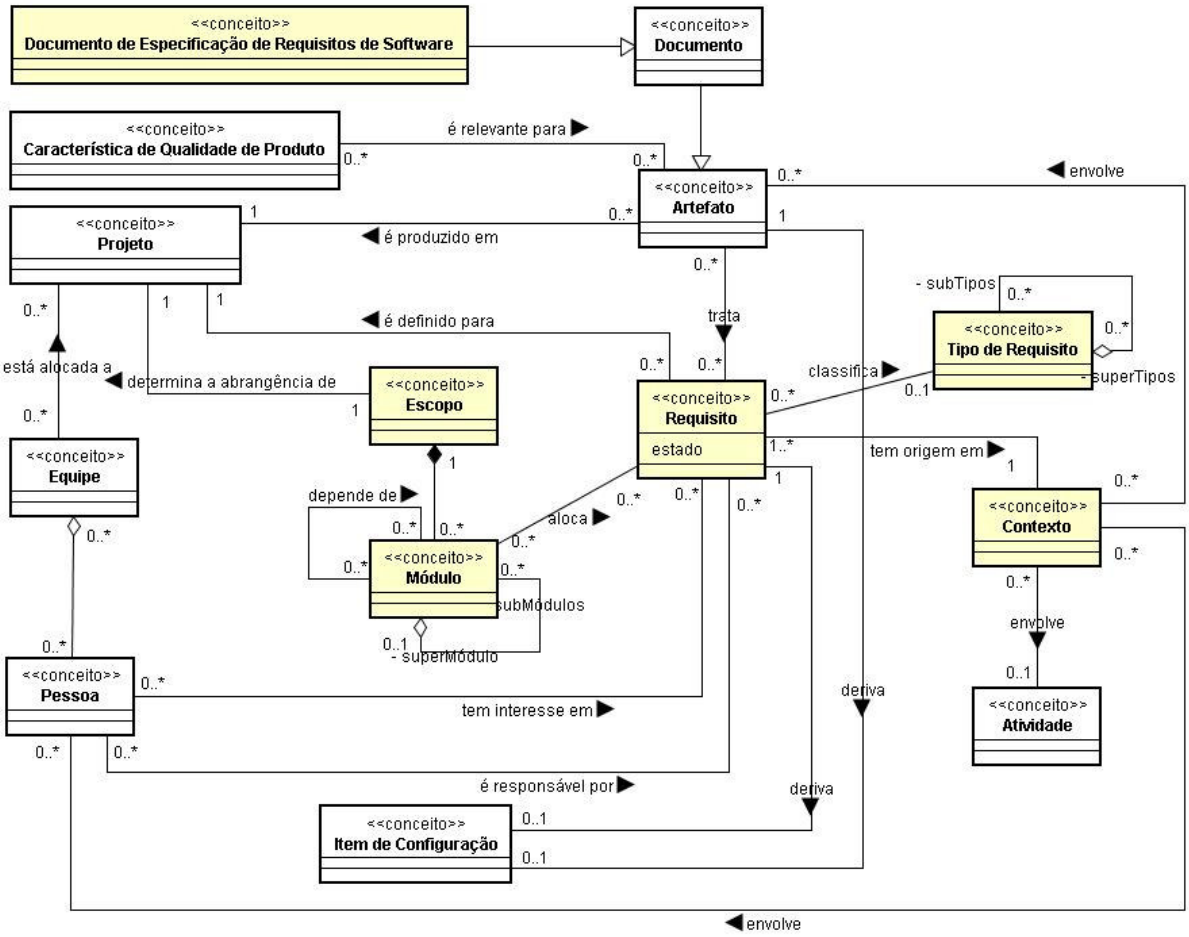


Figura 5.10 – Ontologia de Requisitos de Software

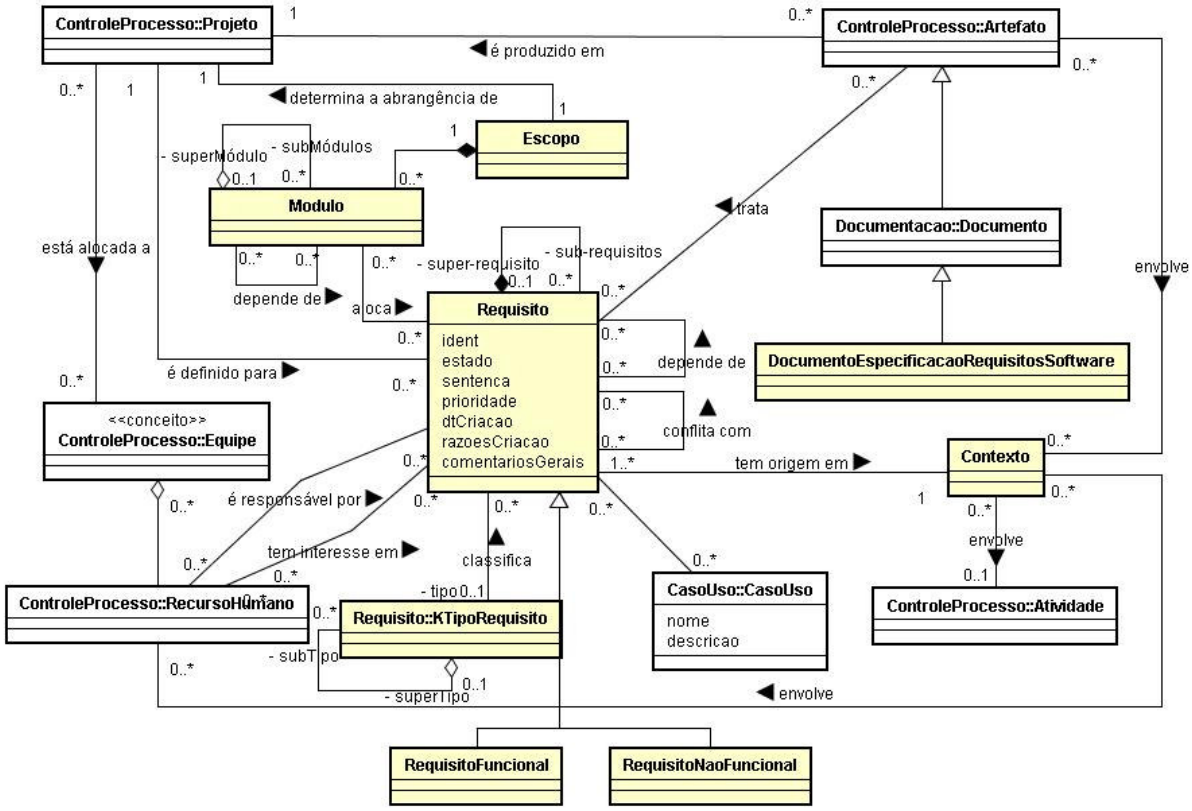


Figura 5.11 – Modelo de Classes de ReqODE

Para que a Arquitetura Conceitual de ODE fosse respeitada, os casos de uso modelados deveriam estar organizados segundo os níveis de Conhecimento e de Controle. Dessa forma, foram considerados dois sub-sistemas: *Conhecimento* e *EngenhariaRequisito*.

O subsistema *Conhecimento*, já existente em ODE, contém os cadastros de conhecimento de todo o ambiente. Assim, passou a contemplar também o caso de uso *Cadastrar Tipo de Requisito*. Já o pacote *EngenhariaRequisito* foi modelado com o propósito de organizar os casos de uso específicos de *ReqODE* e que dizem respeito ao nível de Controle. Dessa forma, ele contempla os casos de uso *Cadastrar Requisito, Definir Ligações de Rastreabilidade, Cadastrar Contexto e Gerar Relatórios de Rastreabilidade*. A Figura 5.12 mostra a dependência entre esses dois sub-sistemas. Pela Arquitetura Conceitual de ODE, o nível de Controle tem uma dependência do nível de Conhecimento, mas o contrário nunca pode existir. Tal fato foi observado e mantido o também em *ReqODE*.

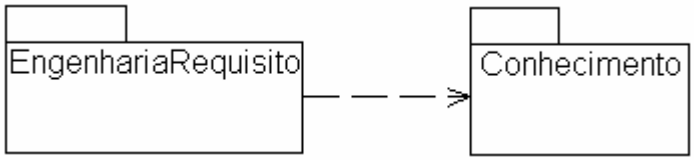


Figura 5.12 - Sub-Sistemas no contexto de *ReqODE*

Além disso, todas essas classes modeladas foram organizadas segundo a arquitetura de pacotes do ambiente. Uma visão geral do diagrama de pacotes resultante é dada na Figura 5.13.

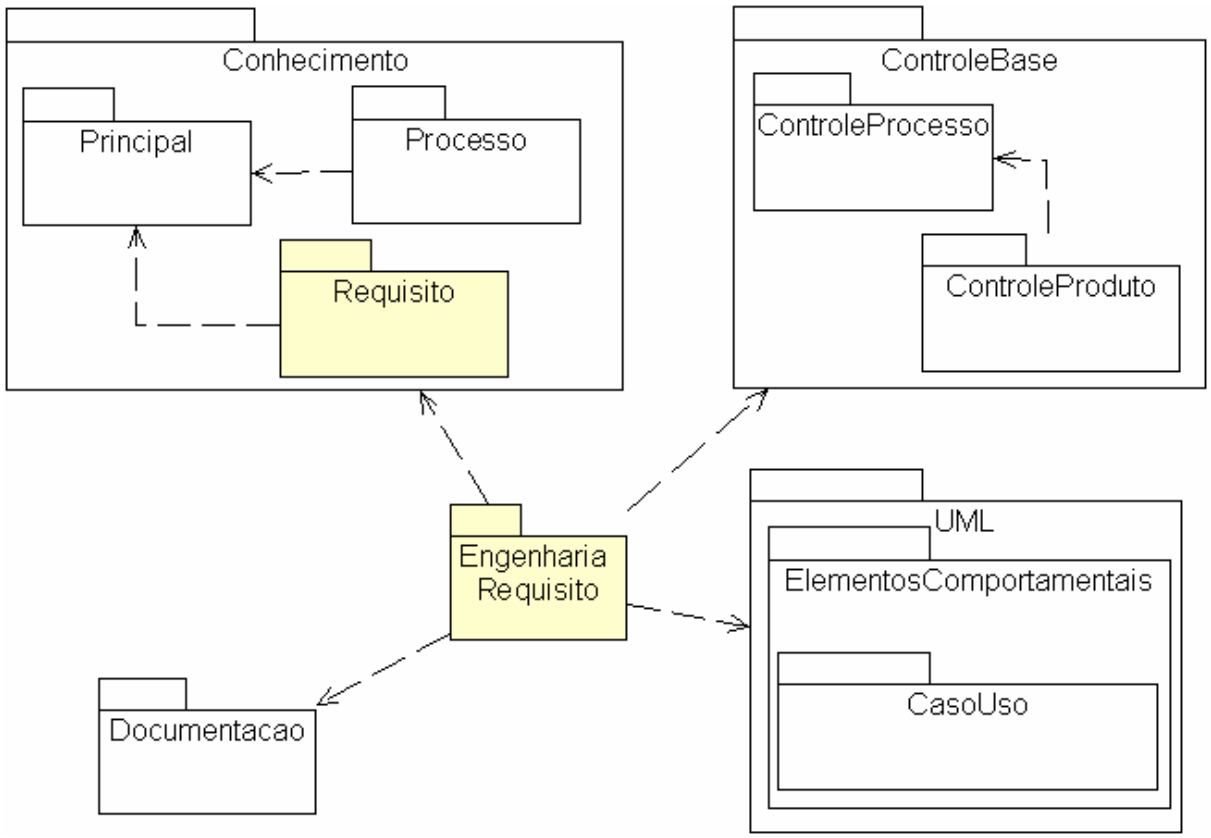


Figura 5.13 - Estrutura de pacotes de Análise (Uma visão geral)

No centro do diagrama tem-se o pacote *EngenhariaRequisito*. Optou-se por um nome mais abrangente, uma vez que ele tem o objetivo de organizar questões a serem exploradas que extrapolam os limites do domínio de requisitos, puramente. Por hora, esse pacote organiza as novas classes derivadas da Ontologia de Requisitos de Software que são: *Requisito, Contexto e DocumentoEspecificacaoRequisitoSoftware.*

O pacote *Conhecimento* concentra todas as classes de ODE responsáveis pela modelagem de conhecimento no ambiente. Para organizar conhecimento acerca de requisitos foi criado o pacote *Requisito.*

O pacote *ControleBase* é o núcleo do Nível Base. Ele concentra as classes derivadas da Ontologia de Processo de Software de ODE, a qual é a ontologia base do ambiente. Nesse pacote estão as classes *Artefato, FerramentaSoftware, Projeto, RecursoHumano, Atividade* etc (sub-pacote *ControleProcesso*) e *Escopo e Modulo,* que foram derivadas dos conceitos de mesmo nome da Ontologia de Requisitos de Software (sub-pacote *ControleProduto*).

O pacote *CasoUso*, sub-pacote de *UML,* concentra a classe *CasoUso* e outras relacionadas à modelagem de casos de uso (SILVA, 2003). Já o pacote *Documentacao* organiza os elementos relacionados à ferramenta de apoio à documentação em ODE (SILVA, 2004). Desse pacote é utilizada a classe *Documento*.

5.2.2 Apresentando *ReqODE*

Como já citado, *ReqODE* é uma ferramenta de ODE que objetiva apoiar a condução das atividades do processo de Engenharia de Requisitos (ER). Para ser utilizada no ambiente, deve estar contemplada no processo de software definido utilizando-se a ferramenta de definição de processo de ODE (BERTOLLO, 2006) e alocada a atividades desse processo, por meio da ferramenta de alocação de recursos de ODE. Feito isso, ODE a disponibiliza aos desenvolvedores responsáveis pela realização das atividades do processo para as quais *ReqODE* está alocada. Deste modo, trata-se a dimensão de integração de processo.

Ao iniciar *ReqODE*, o desenvolvedor se depara com a tela exibida na Figura 5.14. Essa é a janela principal da ferramenta. À esquerda são exibidos os requisitos, funcionais e não funcionais de um dado sistema em desenvolvimento, organizados numa estrutura de árvore para facilitar a organização de requisitos compostos. À direita, tem-se uma série de painéis que fornecem acesso às propriedades do requisito selecionado na árvore à esquerda. Nesses painéis é possível, dentre outros, editar dados gerais (prioridade, responsável, descrição e interessados), dependências, conflitos, relacionamentos com casos de uso etc.

Em relação aos interessados, cabe uma outra colocação em relação à integração de *ReqODE* com as demais ferramentas de ODE, mais especificamente, com a ferramenta de alocação de recursos. Os interessados em um requisito devem ser recursos humanos alocados à equipe do projeto no qual esse requisito foi definido. Assim, se alguma alteração for feita em alguma das alocações do projeto, *ReqODE* deve estar ciente a fim de garantir a consistência nas relações.

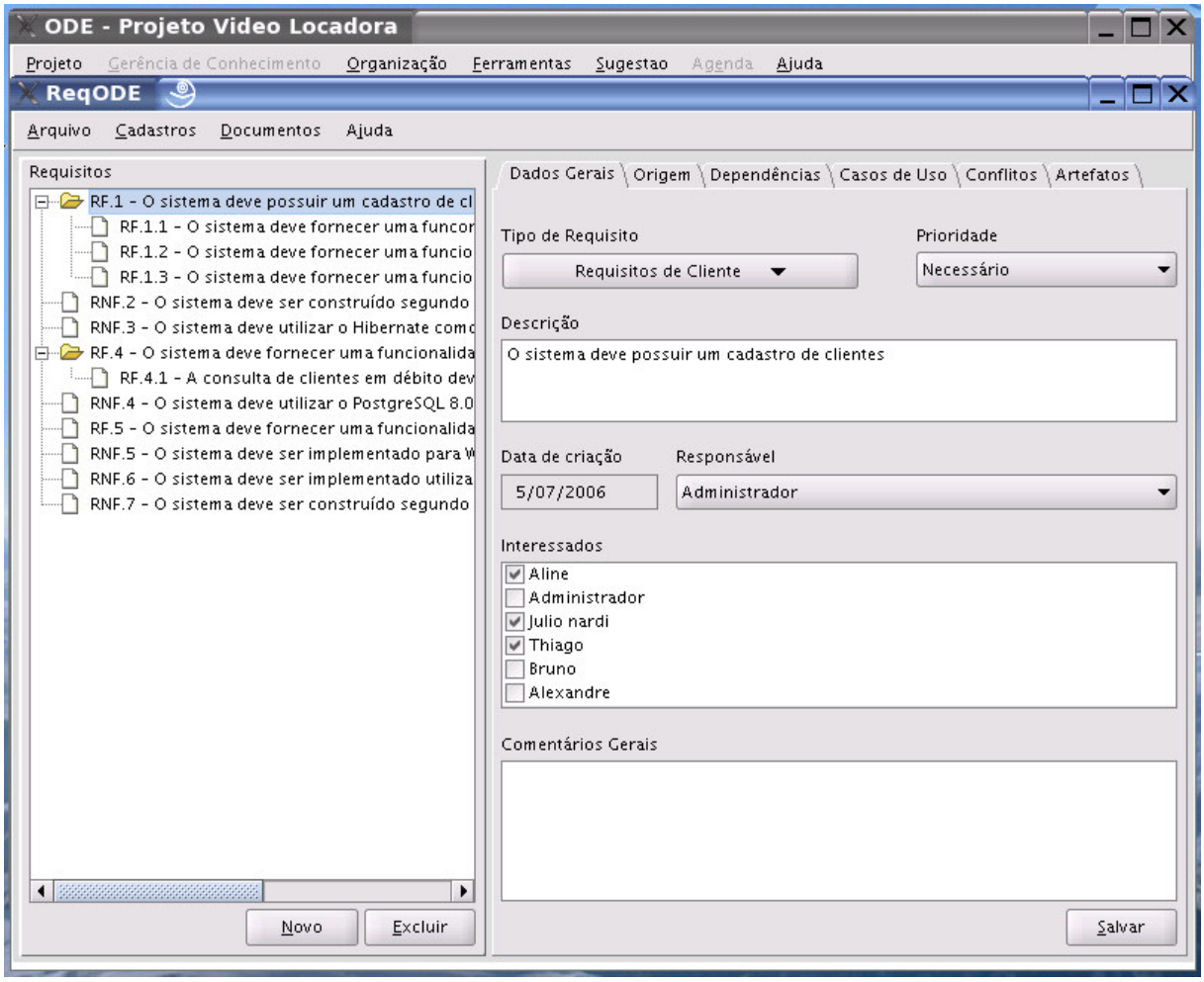


Figura 5.14 – A Edição de Dados Gerais de um Requisito em ReqODE.

Para editar as dependências de um requisito, pode-se acessar o painel *Dependências* que está à direita da tela exibida na Figura 5.15. Esse painel possui uma árvore com todos os requisitos e, a partir dessa árvore, o desenvolvedor pode selecionar as dependências de um requisito. Vale realçar que essa árvore está implementada de forma a garantir algumas restrições automaticamente. Por exemplo, se o desenvolvedor informa que um requisito (R1) depende de outro (R2), sendo que esse (R2) é composto por um terceiro (R3), então o primeiro (R1) também depende do terceiro (R3), pois essa restrição foi definida na ontologia de requisitos. Assim, tais restrições são implementadas na árvore de dependências e ela garante a seleção dos demais requisitos automaticamente. O mesmo acontece com o painel de *Conflitos*, o qual tem uma interface semelhante à do painel *Dependências*, também em uma estrutura de árvore, na qual restrições estabelecidas como axiomas na ontologia também são garantidas.

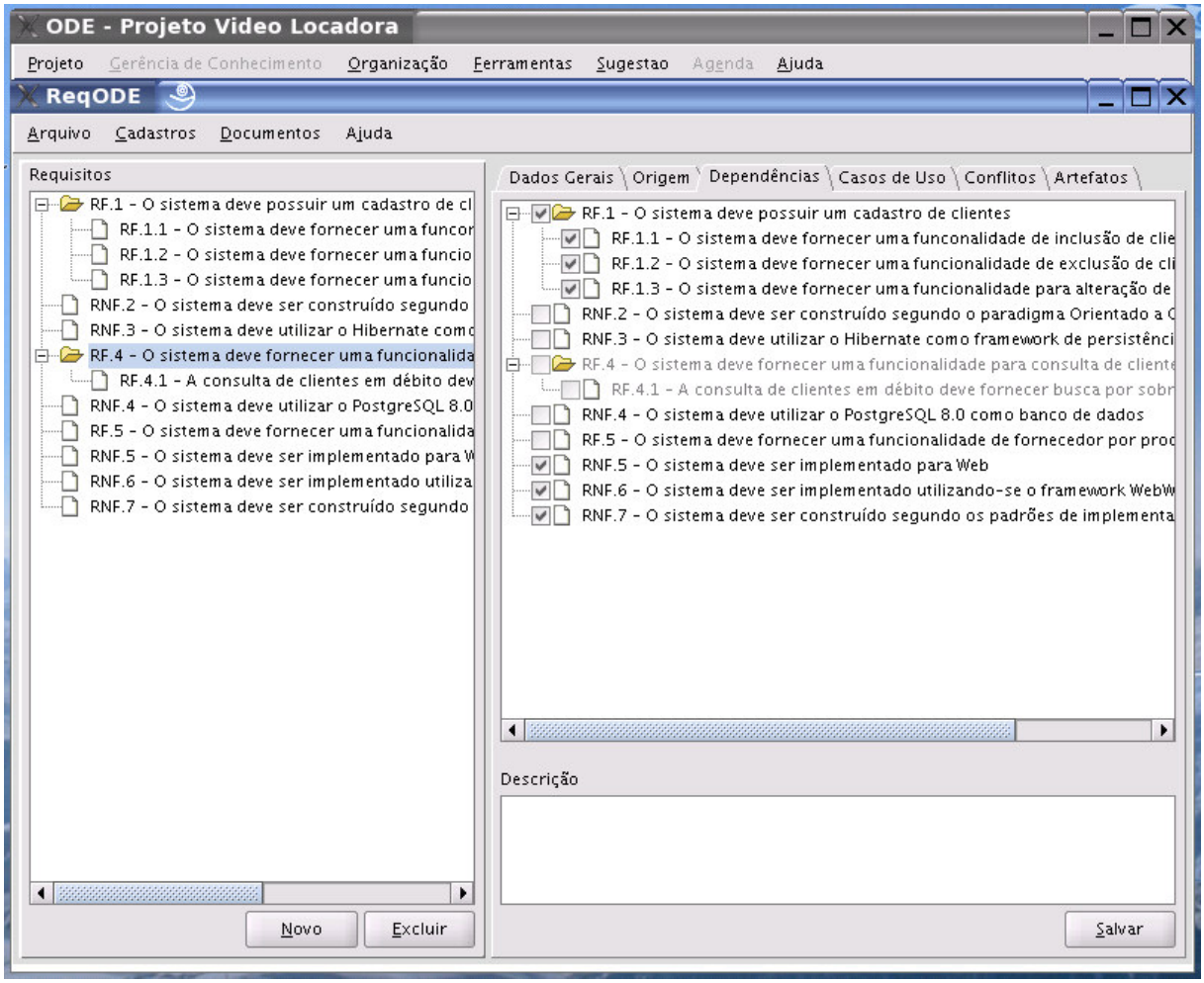


Figura 5.15 – A Edição de Dependências em ReqODE.

A fim de possibilitar o relacionamento dos requisitos com os casos de uso que os modelam, criou-se o painel *Casos de Uso*, a partir do qual pode-se selecionar em quais casos de uso um determinado requisito deverá ser tratado. O *layout* desse painel é exibido na Figura 5.16.

Além disso, ainda visando a manter relacionamento de rastreabilidade, criou-se o painel *Artefato*, o qual é exibido na Figura 5.17. Por meio desse painel é possível relacionar um requisito aos vários artefatos gerados ao longo do processo de desenvolvimento, considerando tanto artefatos internos (gerados pelas ferramentas de ODE) quanto artefatos externos (gerados por ferramentas externas à ODE, como, por exemplo, um editor de textos). Nesse caso, cabe outra colocação no que tange à integração de *ReqODE*. Tais artefatos (externos ou internos) devem ser, de alguma forma, gerenciados pelas ferramentas que os produziram (no caso dos artefatos internos) ou que os disponibilizaram em ODE (no caso de artefatos externos). Assim, uma integração de *ReqODE* com tais ferramentas é necessária para garantir a consistência das relações.

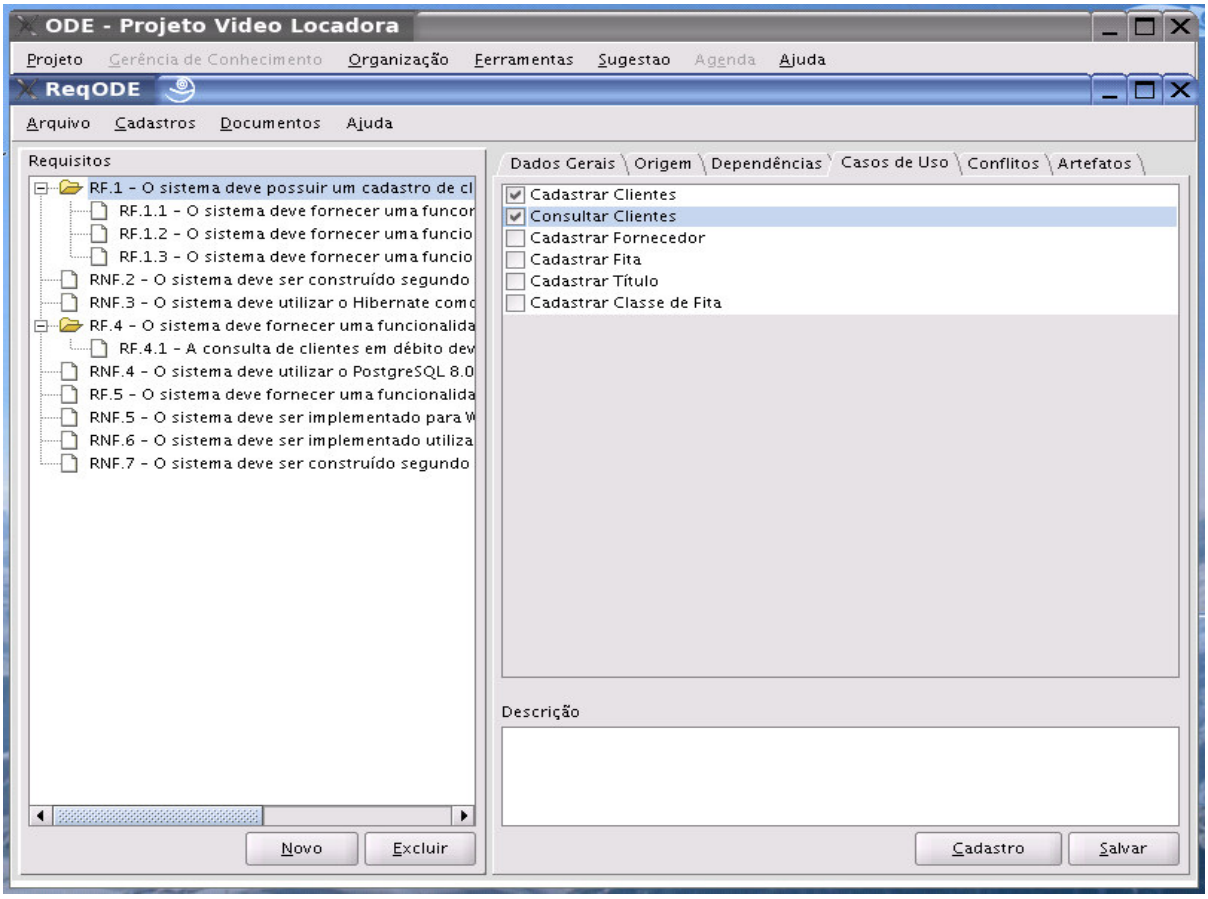


Figura 5.16 – Relacionando Requisitos a Casos de Uso em ReqODE.

Uma vez que em ODE podem-se utilizar vários processos que são, por sua vez, compostos de atividades, as quais produzem artefatos, o painel da Figura 5.17 ilustra que para ter acesso aos artefatos de um projeto é necessário selecionar o processo, a atividade e o tipo de artefato que se deseja relacionar a um dado requisito.

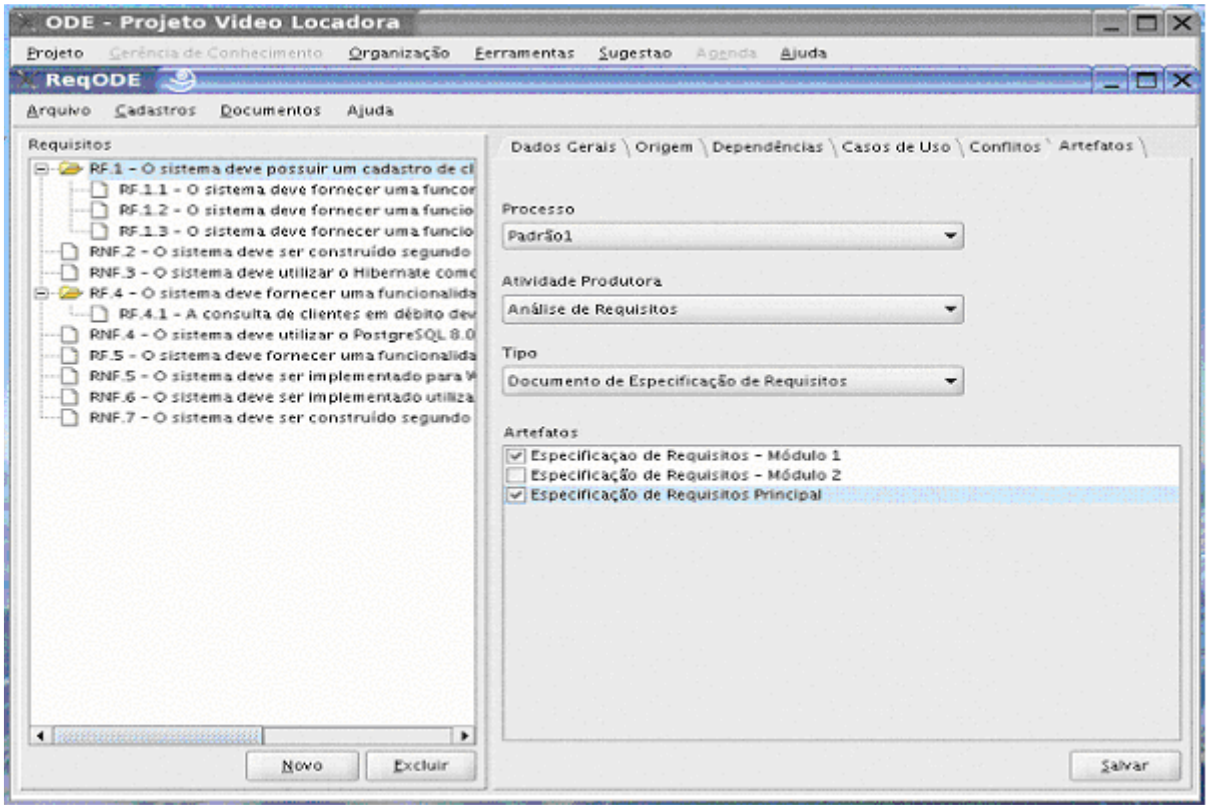


Figura 5.17 – Relacionando Requisitos a Artefatos em ReqODE.

Para que o desenvolvedor possa visualizar, de maneira mais concisa, as ligações de rastreabilidade dos requisitos de um projeto, um relatório é gerado, como o mostrado na Figura 5.18. Nesse relatório é informado, para cada requisito do projeto, os conflitos existentes, os casos de uso com os quais se relaciona, os seus requisitos dependentes e os artefatos pelos quais é tratado. Para facilitar o manuseio das informações, conta-se, também, com a função de impressão de relatório.

Figura 5.18 – Relatório de Rastreabilidade

5.3 Apoio ao Reúso de Itens de Conhecimento no Contexto da Engenharia de Requisitos

A seção anterior apresentou *ReqODE*, a ferramenta construída no contexto deste trabalho que objetiva apoiar a execução de algumas atividades do processo de Engenharia de Requisitos. Agora, nesta seção, objetiva-se mostrar como foi estabelecido o apoio ao reúso de itens de conhecimento no contexto da Engenharia de Requisitos, utilizando a infra-estrutura de Gerência de Conhecimento de ODE.

5.3.1 Buscando apoio na Infra-estrutura de Gerência de Conhecimento existente

Conforme apresentado no Capítulo 3, a infra-estrutura de Gerência de Conhecimento de ODE define cinco serviços - Criação e Captura, Recuperação e Acesso, Disseminação, Uso e Manutenção - a partir dos quais funcionalidades são implementadas. Ou seja, o conceito de serviço nessa infra-estrutura é algo abstrato que se torna concreto na figura de funcionalidades

das ferramentas do ambiente que implementam tais serviços. A Figura 5.19 mostra o diagrama de casos de uso referente aos serviços de Gerência de Conhecimento (GC) de ODE. Como mostra essa figura, o Desenvolvedor pode criar, coletar e utilizar conhecimento, assim como, ao realizar uma atividade do processo de software, pode recuperar e acessar itens de conhecimento ou receber sugestões de itens por meio de um serviço de disseminação pró-ativa. Já o Gerente de Conhecimento pode fazer as mesmas coisas que o Desenvolvedor, mas ainda tem a responsabilidade de fazer a manutenção dos itens de conhecimento da memória organizacional. É importante, no entanto, destacar que os atores Gerente de Conhecimento e Desenvolvedor representam papéis que usuários de ODE podem desempenhados em ODE, ou seja, pode-se ter mais de um gerente de conhecimento atuando no ambiente de modo a descentralizar a manutenção de conhecimento, tornando-a, assim, mais efetiva.

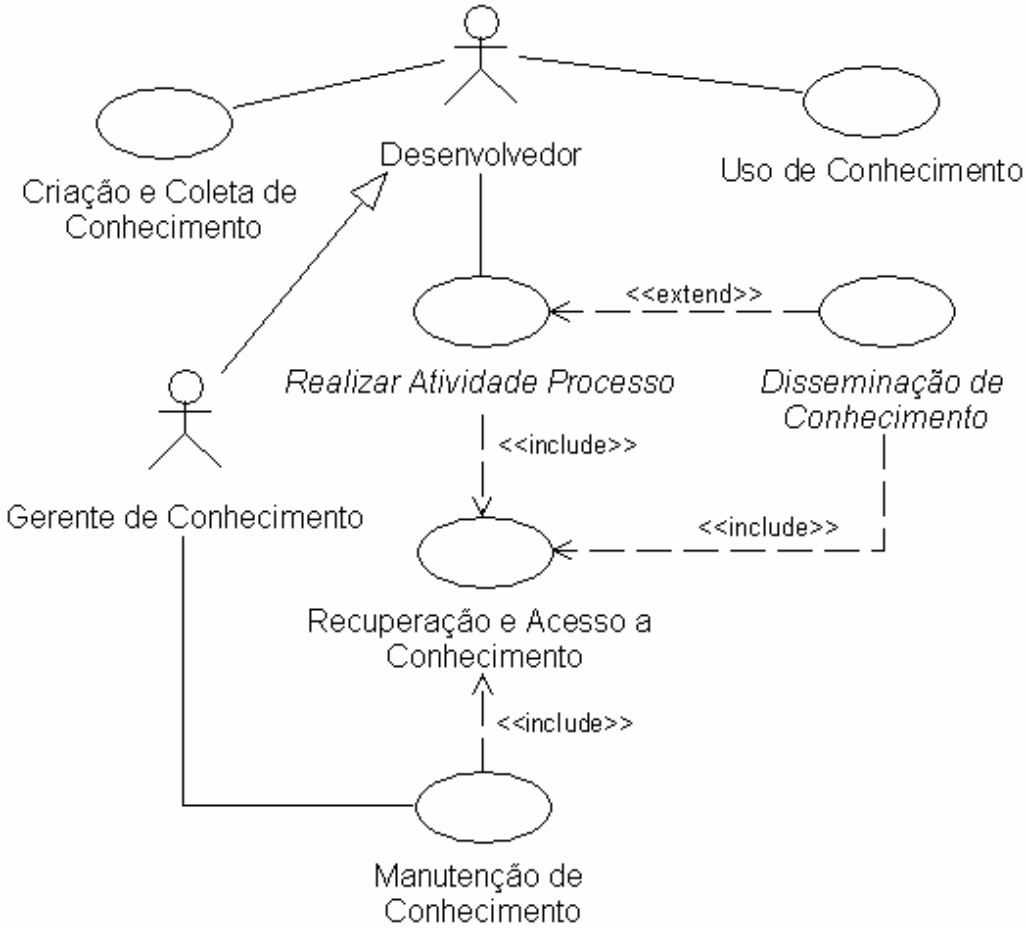


Figura 5.19- Modelo de Casos de Uso dos Serviços de Gerência de Conhecimento (NATALI, 2003)

No contexto desses serviços, existem em ODE funcionalidades como o cadastro de lições aprendidas, o empacotamento de mensagens, a recuperação de artefatos para consulta e a criação e instanciação de ontologias. Assim, de posse desses recursos, o primeiro passo no

sentido de estabelecer o apoio baseado em GC à Engenharia de Requisitos (ER) em ODE foi a definição de quais funcionalidades ligadas ao reúso de conhecimento poderiam ser aproveitadas e integradas a *ReqODE*. Nesse sentido, entendeu-se que o cadastro e reúso de lições aprendidas, o acesso às mensagens trocadas entre os membros da organização durante a resolução de problemas e a consulta a artefatos produzidos em projetos similares seriam muito importantes, pois tais itens de conhecimentos poderiam ajudar os desenvolvedores em negociações com o cliente, na identificação de requisitos, na validação de Documentos de Especificação de Requisitos, na construção de diagramas de classes etc. Dessa maneira, as funcionalidades que lidam com cadastro e aprovação de lições aprendidas, empacotamento e alteração de mensagens e recuperação de artefatos foram contempladas no apoio ao reúso de conhecimento no contexto da ER.

No entanto, a despeito da existência dessas funcionalidades de Gerência de Conhecimento em ODE, era necessário ampliar as possibilidades de reúso, ou melhor, era necessário ampliar os tipos de itens de conhecimento reutilizados e fornecer funcionalidades que dessem suporte à gerência desses itens.

5.3.2 Estabelecendo novas formas de Reúso de Itens de Conhecimento em ODE

Como já discutido neste capítulo, modelos de domínio, gerados por atividades de Análise de Domínio, são de grande valia para a Engenharia de Requisitos, uma vez que tais modelos fornecem conhecimento acerca de domínios que são alvo das atividades da ER. Assim, fazer um estudo sobre o domínio a ser trabalhado antes de se iniciar uma atividade de Levantamento de Requisitos pode ajudar a elucidar questões ou direcionar melhor as perguntas no âmbito de uma entrevista com o cliente. Além disso, modelos preliminares de domínio podem tornar a atividade de modelagem mais fácil, uma vez que fornecem um ponto de partida. Neste sentido, o apoio à Engenharia de Requisitos foi estendido de forma a contemplar o reúso de modelos. Assim, a seção 5.3.2.1 discute como eram modelados e utilizados os modelos de ODE e como passaram a ser tratados com vistas ao reúso. A seção 5.3.2.2 apresenta os novos casos de uso que foram modelados com base nos serviços da infra-estrutura de Gerência de Conhecimento e que objetivam fornecer meios para que esses itens de conhecimento formais sejam reutilizados.

5.3.2.1 Entendendo as Mudanças no Trato com os Modelos de ODE

Em ODE existiam dois tipos de modelo, Modelo de Objetos e Ontologias, como mostrado na Figura 5.20.

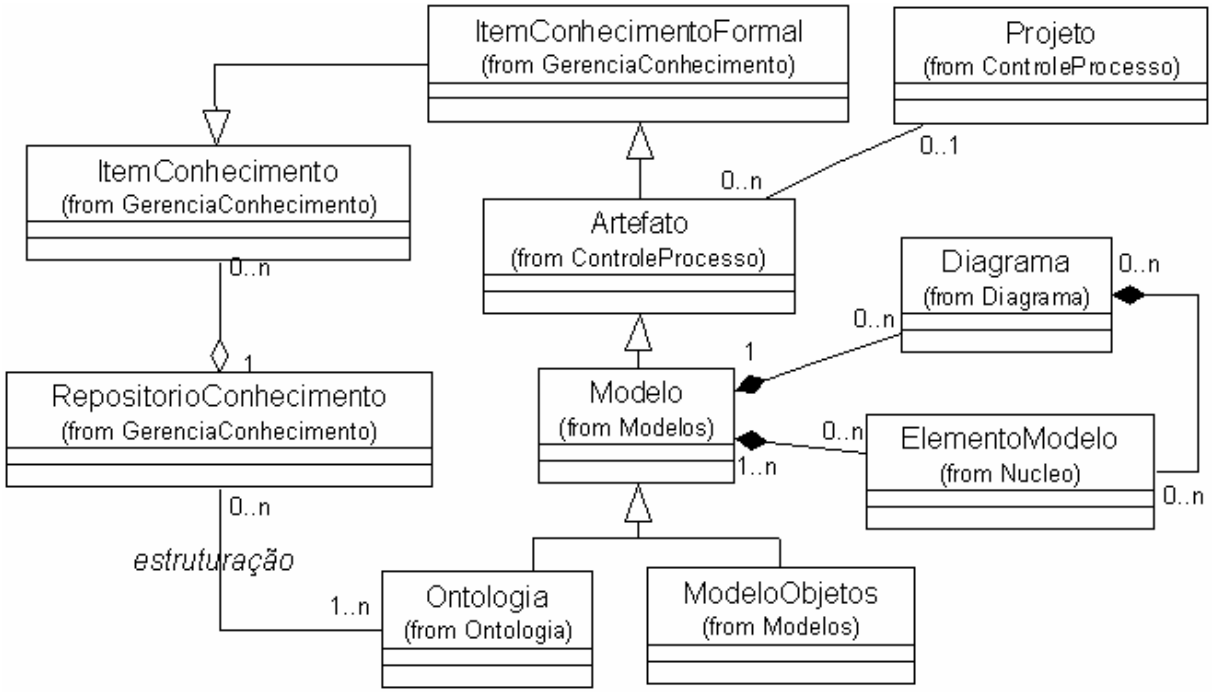


Figura 5.20 - Modelos de ODE em uma versão anterior a este trabalho.

A classe *ModeloObjetos* representava os modelos gerados pelas atividades de Análise e Projeto do processo de desenvolvimento de software, os quais eram compostos por diagramas (Diagramas de Classes, Diagramas de Casos de Uso, Diagramas de Estados etc) e elementos de modelo (classes, associações, casos de uso etc). Tais modelos podiam ser construídos por meio de OODE (SILVA, 2003), a ferramenta de ODE para modelagem UML.

As ontologias de ODE, modeladas pela classe *Ontologia*, eram utilizadas para estruturar o repositório de conhecimento de ODE, ou seja, objetivavam fornecer uma base formal na qual as informações do repositório podiam ser mais facilmente estruturadas e manipuladas pelos serviços de Gerência de Conhecimento. A modelagem de ontologias era feita pela ferramenta ODEd (MIAN, 2003) (SOUZA, 2004), a qual utilizava LINGO (FALBO, 1998) como linguagem gráfica de modelagem de ontologias.

Tais modelos, ainda que fossem considerados Artefatos e, por conseguinte, itens formais de conhecimento, como mostrado na Figura 5.20, não eram tratados propriamente como se fossem conhecimento organizacional, ou seja, conhecimento a ser reutilizado pelos desenvolvedores na condução de seus projetos.

Dessa maneira o primeiro passo foi considerar que tais modelos seriam disponibilizados pelos serviços de Gerência de Conhecimento. Após isso, uma vez que Padrões de Análise fornecem soluções de problemas recorrentes para um dado domínio de aplicação e pelo fato de estarem intimamente relacionados à Análise de Requisitos, decidiu-se considerar, também, esse tipo de modelo e incorporá-lo como um item de conhecimento formal de ODE. A Figura 5.21 mostra, dentre outros, a nova estrutura de itens de conhecimento formais de ODE.

Além da nova estrutura de modelos de ODE contemplar padrões de análise como modelos de objetos, a Figura 5.21 fornece uma visão geral dos itens de conhecimento formais que passaram a ser trabalhados no apoio à Engenharia de Requisitos, os quais são: Requisitos, Modelos de Objetos, Padrões de Análise e Ontologias.

Além disso, com vistas ao reúso de modelos e para que não seja feita a utilização de uma informação em certo nível, sem que essa seja precedida da reutilização da informação correspondente nos níveis mais abstratos, como recomendado por Cima et al. (1997), criou-se um auto-relacionamento na classe *Modelo*, nomeado *reutilização*, que visa a manter uma ligação informando a partir de que modelos um modelo foi derivado. Isso é importante, pois um desenvolvedor, ao reutilizar um determinado modelo, pode ter a necessidade de chegar até a sua origem e consultar lições aprendidas, comentários informados ou, até mesmo, explorar o modelo reutilizado originalmente para obter mais detalhes.

Criou-se, ainda, uma associação entre a classe *Modelo* e *DominioAplicacao* com o objetivo de registrar sobre qual domínio de aplicação trata um modelo e facilitar a operação dos serviços de busca e disseminação de conhecimento, permitindo trabalhar sobre modelos de um mesmo domínio.

Esclarecidas as decisões tomadas acerca dos novos itens de conhecimento que passaram a ser trabalhados pelos serviços de Gerência de Conhecimento, a partir deste ponto, passa-se a apresentar as funcionalidades que visam a fornecer um apoio baseado em gerência de conhecimento aos desenvolvedores no contexto de *ReqODE*.

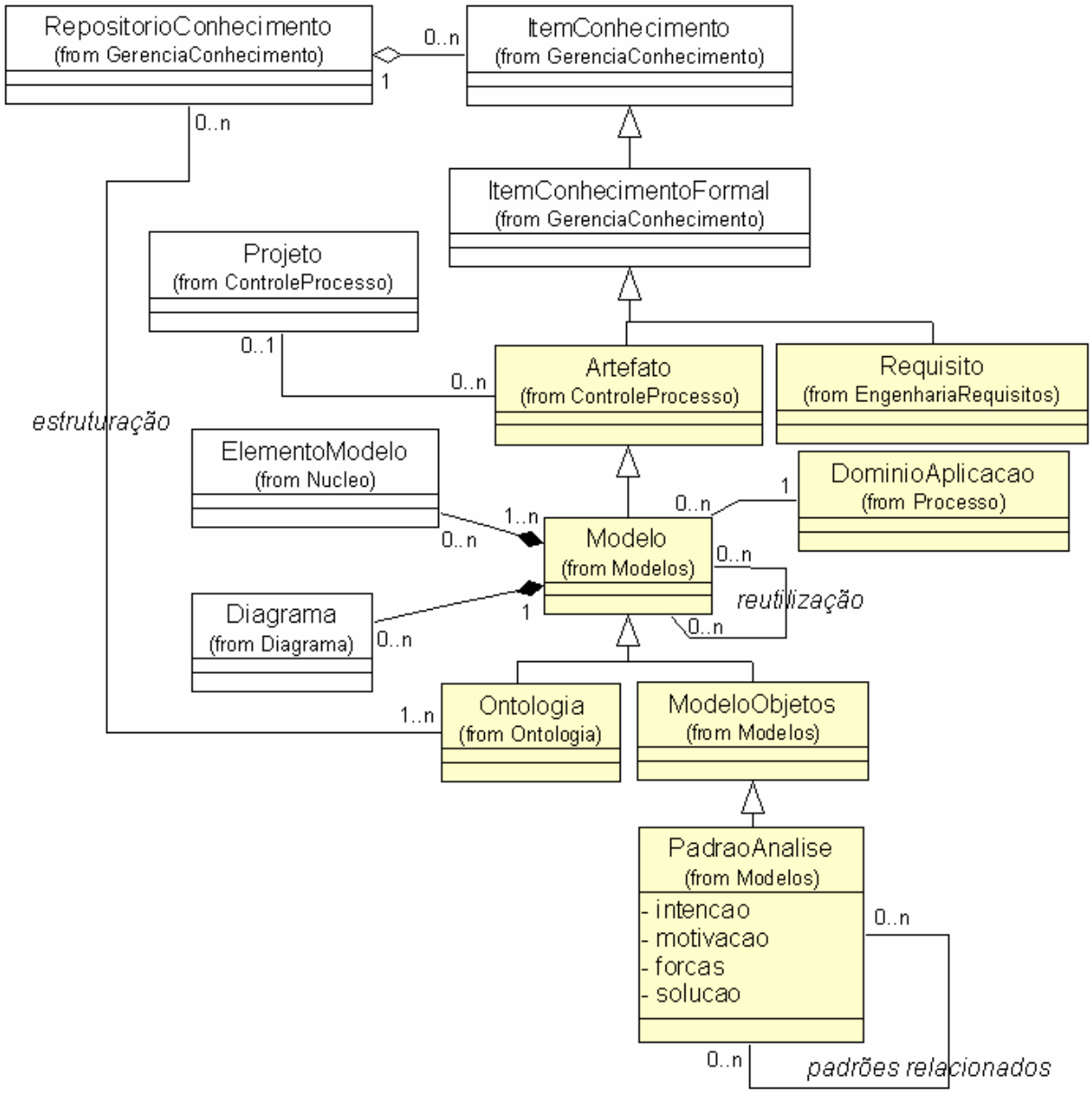


Figura 5.21 – Modelagem de Itens de Conhecimento Formal para Reúso

5.3.2.2 Modelando Novos Casos de Uso visando à Reutilização de Itens de Conhecimento Formais

Conforme discutido na seção 5.3.1, a infra-estrutura de gerência de conhecimento de ODE é utilizada por dois atores: Desenvolvedor e Gerente de Conhecimento. Ao trabalhar o apoio de gerência de conhecimento à ER no contexto de *ReqODE*, definiu-se mais um ator, o Engenheiro de Requisitos, o qual é uma especialização do ator Desenvolvedor, ou seja, um desenvolvedor com características peculiares. Esse ator é o contemplado com as novas funcionalidades de Gerência de Conhecimento, como mostra a Figura 5.22.

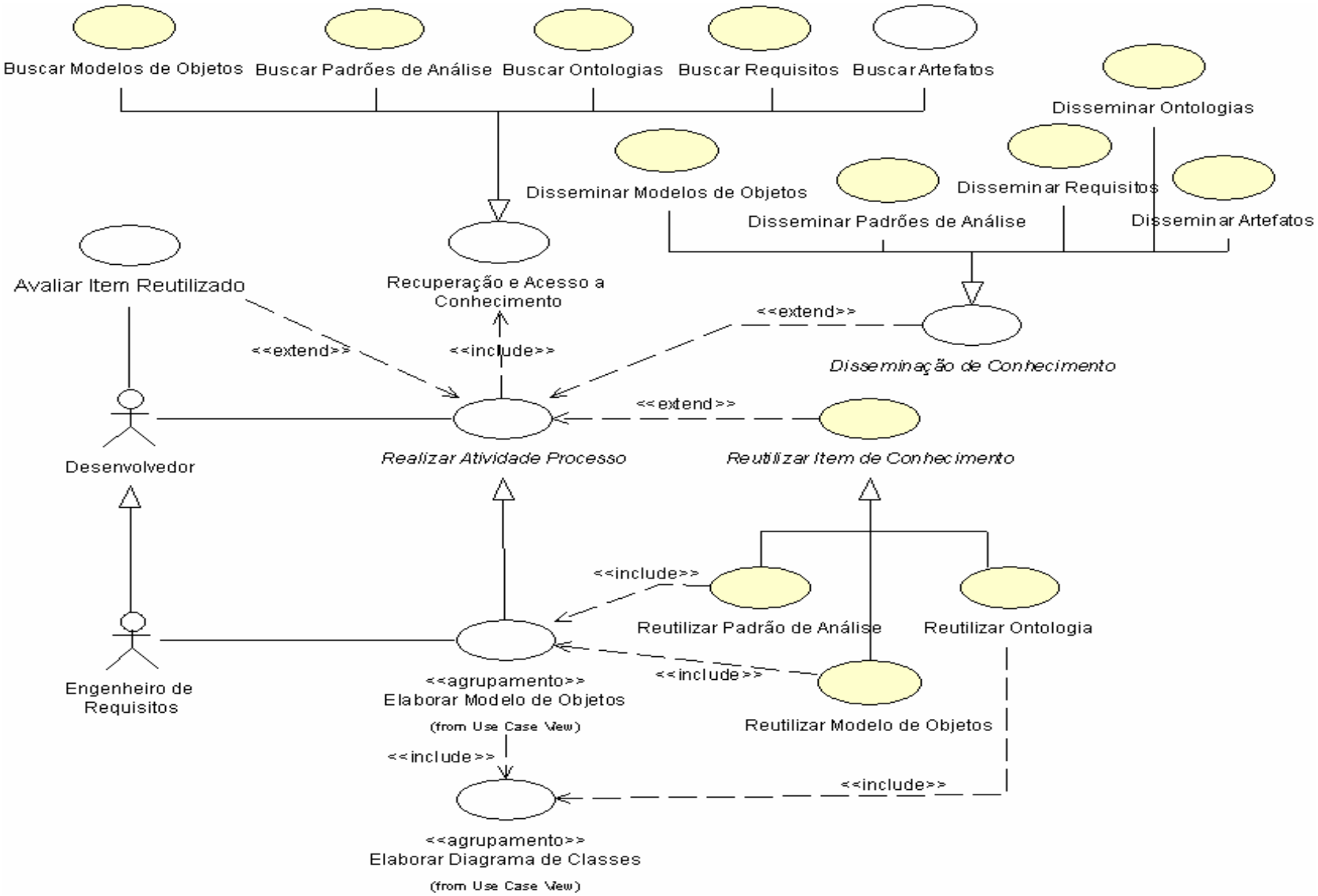


Figura 5.22 – Novos Casos Uso para Reutilização de Itens de Conhecimento Formal sob a ótica do Engenheiro de Requisitos

Os casos de uso *Buscar Modelos de Objetos*, *Buscar Ontologias*, *Buscar Padrões de Análise* e *Buscar Requisitos*, como a especialização do diagrama sugere, modelam funcionalidades referentes ao serviço de Recuperação e Acesso a Conhecimento.

Os casos de uso *Disseminar Modelos de Objetos*, *Disseminar Ontologias*, *Disseminar Padrões de Análise*, *Disseminar Requisitos* e *Disseminar Artefatos* modelam funcionalidades que objetivam implementar o serviço de Disseminação de Conhecimento, fornecendo conhecimento de maneira pró-ativa, podendo-se, portanto, utilizar a tecnologia de agentes, conforme sugerido em (FALBO et al., 2005).

O caso de uso *Reutilizar Itens de Conhecimento* é especializado nos casos de uso *Reutilizar Padrão de Análise, Reutilizar Modelo de Objetos* e *Reutilizar Ontologias*, que visam a fornecer meios para a reutilização desses novos itens formais de conhecimento.

O diagrama da Figura 5.22 mostra, ainda, que um Engenheiro de Requisitos pode, ao *Elaborar um Modelo de Objetos*, reutilizar itens formais de conhecimento como ontologias, padrões de análise ou modelos de objetos de projetos similares. No entanto, esses itens, antes de serem reutilizados, devem ser buscados utilizando-se dos serviços de Recuperação e Acesso a Conhecimento ou fornecidos pelos serviços de Disseminação de Conhecimento. Além disso, ao efetuar a reutilização de um item de conhecimento, o Engenheiro de Requisitos pode *Avaliar o Item Reutilizado* como forma de fornecer um *feedback* útil para a execução dos serviços de Manutenção de Conhecimento.

Uma vez que a função do Gerente de Conhecimento é fornecer e manter conhecimento organizacional, ele passou a dispor de funcionalidades que objetivam dar apoio à captura e ao reúso de ontologias e padrões de análise. A Figura 5.23 exibe o diagrama de casos de uso correspondente à visão desse ator. Como mostra essa figura, é dada ao Gerente de Conhecimento a possibilidade de construir ontologias a serem disponibilizadas para o restante da organização. Além disso, o Gerente de Conhecimento pode, a partir de observações de modelos de objetos, encontrar soluções padronizadas para problemas recorrentes e reutilizar tais modelos de objetos na elaboração de padrões de análise, que são, então, disponibilizados no repositório de conhecimento para posterior reúso. Para tanto, o Gerente de Conhecimento conta também com serviços de Gerência de Conhecimento que apóiam o reúso, a busca, a disseminação e a captura e criação de itens de conhecimento.

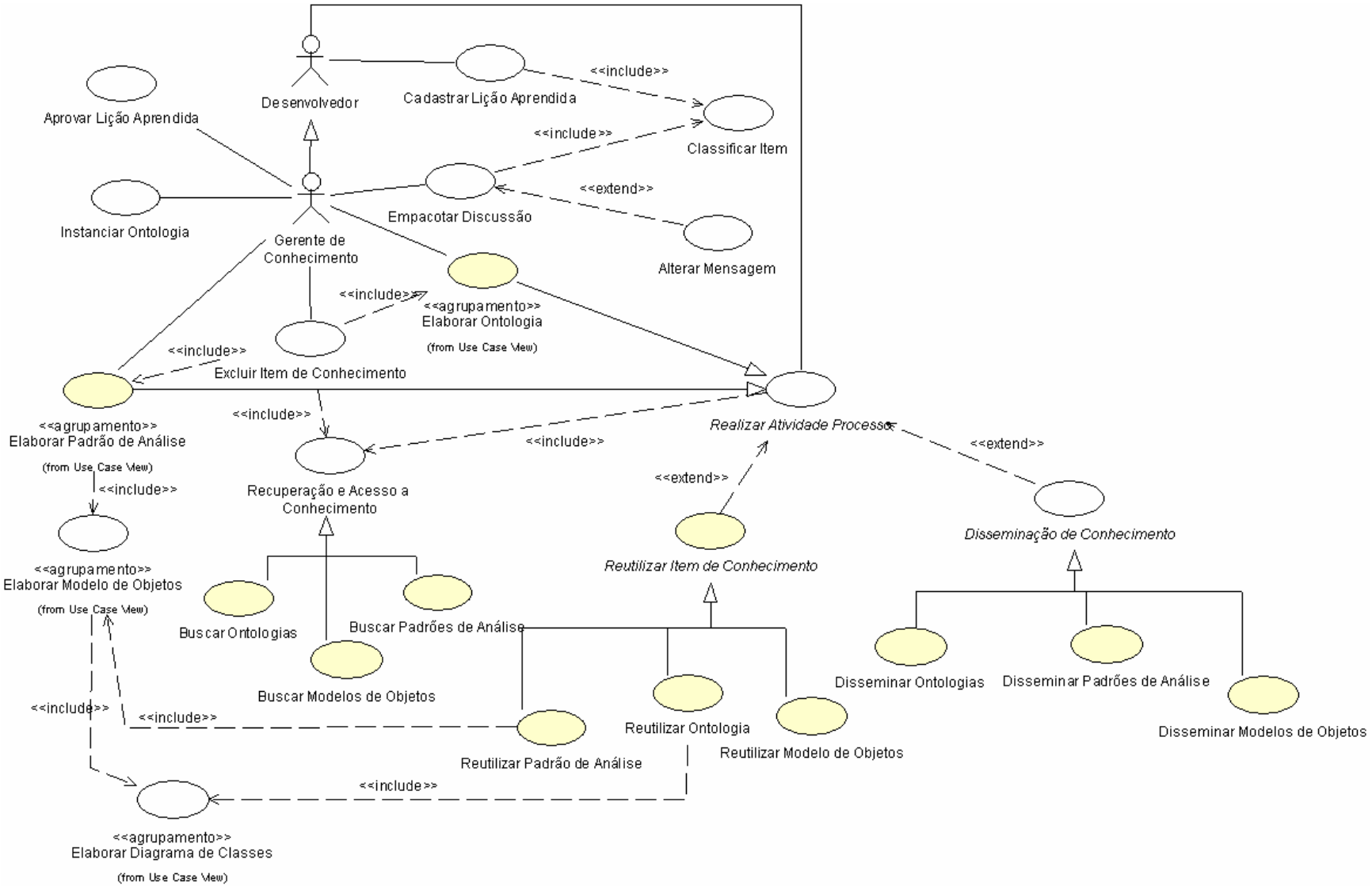


Figura 5.23 - Novos Casos Uso para o Trato de Itens de Conhecimento Formal sob a ótica do Gerente de Conhecimento

Uma vez que foram apresentados os diagramas de casos de uso propostos neste trabalho e descritas as situações em que se aplicam, a seguir é feita uma sucinta descrição de cada caso de uso, a fim de melhor apresentar os objetivos de cada um. É importante reforçar que os casos de uso referentes à busca por itens de conhecimento estão diretamente relacionados ao propósito dos serviços de recuperação e acesso a conhecimento, ou seja, a iniciativa de busca parte do usuário. Já os casos de uso referentes à disseminação de itens de conhecimento estão diretamente relacionados aos serviços de disseminação pró-ativa de conhecimento, ou seja, nesse último caso, a iniciativa de disponibilizar itens de conhecimento para o usuário parte do sistema de Gerência de Conhecimento.

- **Buscar Ontologias**

Este caso de uso permite que um Engenheiro de Requisitos ou um Gerente de Conhecimento busque ontologias no repositório de conhecimento. Para tanto, podem-se utilizar buscas baseadas em palavras-chave dos textos que descrevem as ontologias, baseadas nos conceitos, relações ou propriedades da ontologia, baseadas no domínio de aplicação da ontologia etc.

- **Buscar Padrões de Análise**

Este caso de uso permite que um Engenheiro de Requisitos ou um Gerente de Conhecimento busque padrões de análise no repositório de conhecimento. Para tanto, podem-se utilizar buscas baseadas em palavras-chave dos textos que caracterizam os padrões como (Nome, Solução, Intenção, Motivação etc.), baseadas nos elementos de modelo (classes, casos de uso, estados, atividades etc) do padrão, baseadas no domínio de aplicação do padrão, baseadas nos padrões relacionados etc.

- **Buscar Modelos de Objetos**

Este caso de uso permite que um Engenheiro de Requisitos ou um Gerente de Conhecimento busque modelos de objetos de projetos similares. Assim, dado um conjunto de projetos similares, podem-se realizar buscas baseadas em palavras-chave dos textos que descrevem os modelos de objetos, baseadas nos elementos de modelo (classes, casos de uso, estados, atividades etc) do modelo, baseadas no domínio de aplicação do modelo de objetos etc.

- **Buscar Requisitos**

Este caso de uso permite que o Engenheiro de Requisitos busque requisitos em projetos similares. Assim, pode-se, a partir de um conjunto de projetos similares, buscar requisitos por similaridade de texto, por tipos de requisitos etc.

- **Disseminar Ontologias**

Este caso de uso objetiva fornecer a um Engenheiro de Requisitos ou Gerente de Conhecimento, de forma pró-ativa, ontologias existentes no repositório de conhecimento.

- **Disseminar Padrões de Análise**

Este caso de uso objetiva fornecer a um Engenheiro de Requisitos ou Gerente de Conhecimento, de forma pró-ativa, padrões de análise existentes no repositório de conhecimento.

- **Disseminar Modelos de Objetos**

Este caso de uso objetiva fornecer a um Engenheiro de Requisitos ou Gerente de Conhecimento, de forma pró-ativa, modelos de objetos de projetos similares.

- **Disseminar Requisitos**

Este caso de uso objetiva fornecer ao Engenheiro de Requisitos, de forma pró-ativa, requisitos existentes em projetos similares.

- **Disseminar Artefatos**

Este caso de uso objetiva fornecer ao Engenheiro de Requisitos, de forma pró-ativa, artefatos de projetos similares, tais como documentos de especificação de requisitos e planos de riscos.

- **Reutilizar Item de Conhecimento**

Dados os itens de conhecimento para reúso, este caso de uso permite que o desenvolvedor reutilize esses itens de formas variadas de acordo com as características de cada um. Por ter um caráter genérico, tal caso de uso foi modelado como abstrato e foi especializado nos casos de uso de reutilização descritos a seguir.

- **Reutilizar Ontologia**

Este caso de uso permite que um Engenheiro de Requisitos ou um Gerente de Conhecimento, de posse de ontologias, reutilize suas conceituações na execução de atividades. Assim, um Engenheiro de Requisitos pode consultar ou importar a conceituação de uma ontologia para um modelo de objetos, quando são mapeados conceitos em classes, relações em associações e propriedades em atributos. De modo similar, um Gerente de Conhecimento pode consultar ou importar ontologias na construção de padrões de análise ou mesmo na modelagem de novas ontologias.

- **Reutilizar Padrão de Análise**

Este caso de uso permite que um Engenheiro de Requisitos ou um Gerente de Conhecimento, de posse de padrões de análise, reutilize seus modelos na execução de atividades. Assim, um Engenheiro de Requisitos pode consultar ou importar a solução documentada em um padrão para um modelo de objetos sendo, portanto, copiados os elementos de modelo do padrão de análise para o modelo de objetos. Um Gerente de Conhecimento, por sua vez, pode consultar ou importar soluções documentadas em padrões de análise na geração de outros padrões de análise.

- **Reutilizar Modelo de Objeto**

Este caso de uso permite que um Engenheiro de Requisitos ou Gerente de Conhecimento, de posse de modelos de objetos, reutilize seus diagramas e elementos de modelo na execução de atividades. Assim, um Engenheiro de Requisitos pode consultar ou importar a solução documentada em um modelo de objetos para um outro modelo de objetos, copiando os elementos de modelo de um modelo para outro. Já o Gerente de Conhecimento pode consultar modelos de objetos a fim de detectar padrões de solução para problemas recorrentes e abstrair tais soluções e documentá-las na forma de padrões de análise.

- **Elaborar Padrão de Análise (Agrupamento)**

Esse caso de uso permite que o Gerente de Conhecimento construa e caracterize padrões de análise a serem disponibilizados no repositório de conhecimento para posterior utilização. Diz-se que este caso de uso é um agrupamento pelo fato dele ser, na verdade, composto de diversos casos de uso menores que permitem, dentre outros, elaborar diagramas de classes, diagramas de estado, diagramas de seqüência etc (SILVA, 2003), (NARDI, 2003), (CARREIRO, 2005).

- **Elaborar Ontologia (Agrupamento)**

Este caso de uso permite que o Gerente de Conhecimento construa ontologias a serem disponibilizadas no repositório de conhecimento para posterior utilização. Tal caso de uso foi implementando inicialmente em (MIAN, 2003) e (SOUZA, 2004) e, agora, foi integrado à ferramenta de modelagem UML de ODE (OODE).

A partir das mudanças na estrutura de itens formais de conhecimento e da criação de novas funcionalidades relativas aos serviços de Gerência de Conhecimento de ODE, é

possível apresentar como se dá a reutilização de conhecimento no contexto da Engenharia de Requisitos em *ReqODE*, tratado a seguir.

5.3.3 Reusando Conhecimento a partir de *ReqODE*

Como dito ao longo deste capítulo, funcionalidades específicas de apoio ao reúso de conhecimento na Engenharia de Requisitos foram desenvolvidas e implementadas no contexto de *ReqODE*.

A reutilização de requisitos de projetos similares é mostrada na Figura 5.24. A partir de *ReqODE*, podem-se recuperar projetos similares, utilizando a estrutura de caracterização de projetos existente em ODE (CARVALHO et al., 2006) e, a partir de um desses projetos, solicita-se a listagem de seus requisitos. Caso haja algum requisito a ser reutilizado, o desenvolvedor pode importá-lo para seu projeto.

Para prover ferramentas integradas de apoio à Engenharia de Requisitos, neste trabalho, possibilitou-se o acesso, a partir de *ReqODE*, à ferramenta de apoio à modelagem usando a UML, OODE (SILVA, 2003) (NARDI, 2003) (CARREIRO, 2005), dado que a modelagem de requisitos também é parte do processo de ER. Assim, a Figura 5.25 destaca essa integração e procura dar uma visão geral da interface de OODE. Como fruto deste trabalho, OODE passou a ser uma ferramenta de modelagem de propósito mais geral, permitindo, além da elaboração de modelos de objetos, a construção de ontologias e padrões de análise.

Reforçando essa idéia de propósito geral, a Figura 5.26 mostra o diagrama de uma ontologia construída em OODE. Assim, o Engenheiro de Conhecimento pode modelar e disponibilizar ontologias no repositório de conhecimento do ambiente, as quais podem, a partir desse momento, serem reutilizadas. Outro ponto a ser destacado é a uniformidade da representação dos diagramas de OODE. Tanto os diagramas de modelos de objetos e padrões de análise quanto de ontologias utilizam a mesma representação. Isso facilita a comunicação entre os desenvolvedores, mesmo em diferentes níveis de abstração.

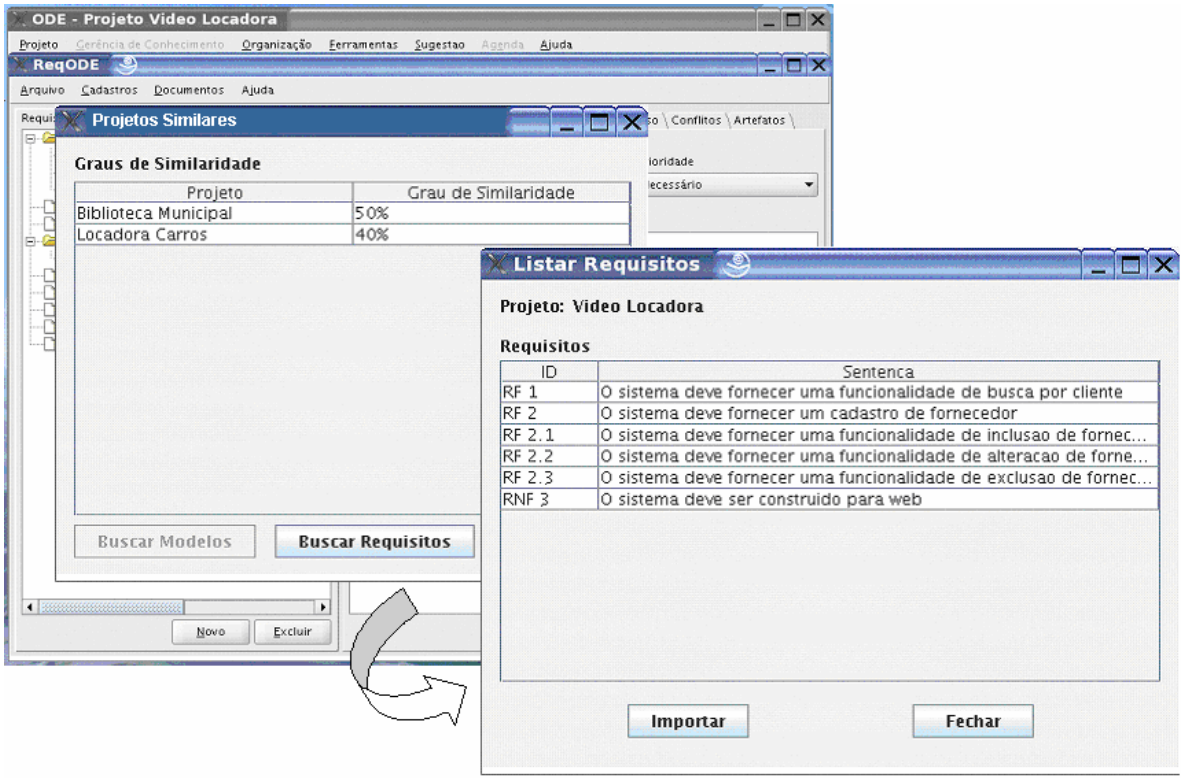


Figura 5.24 – Reúso de Requisitos de Projetos Similares

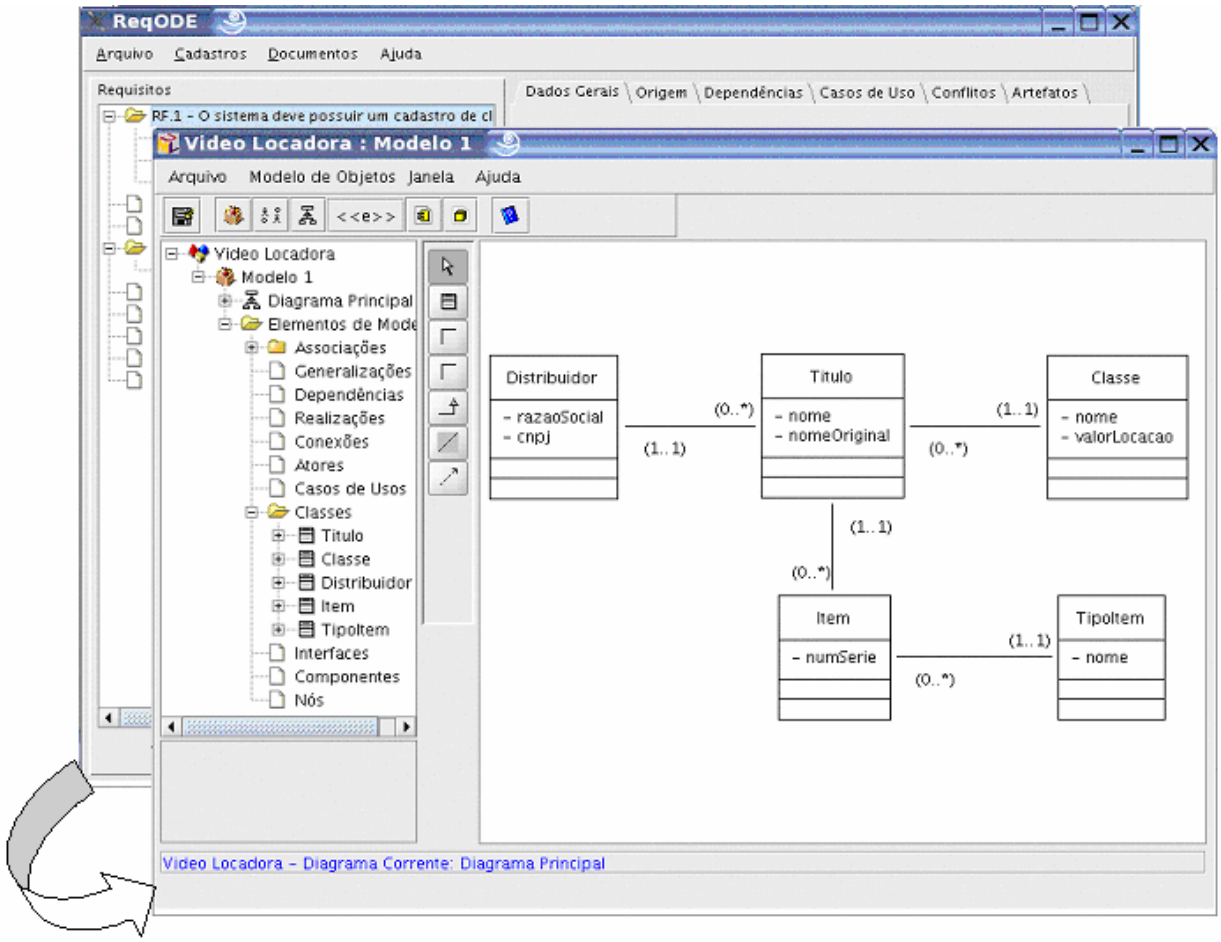


Figura 5.25 – Integração entre *ReqODE* e *OODE*

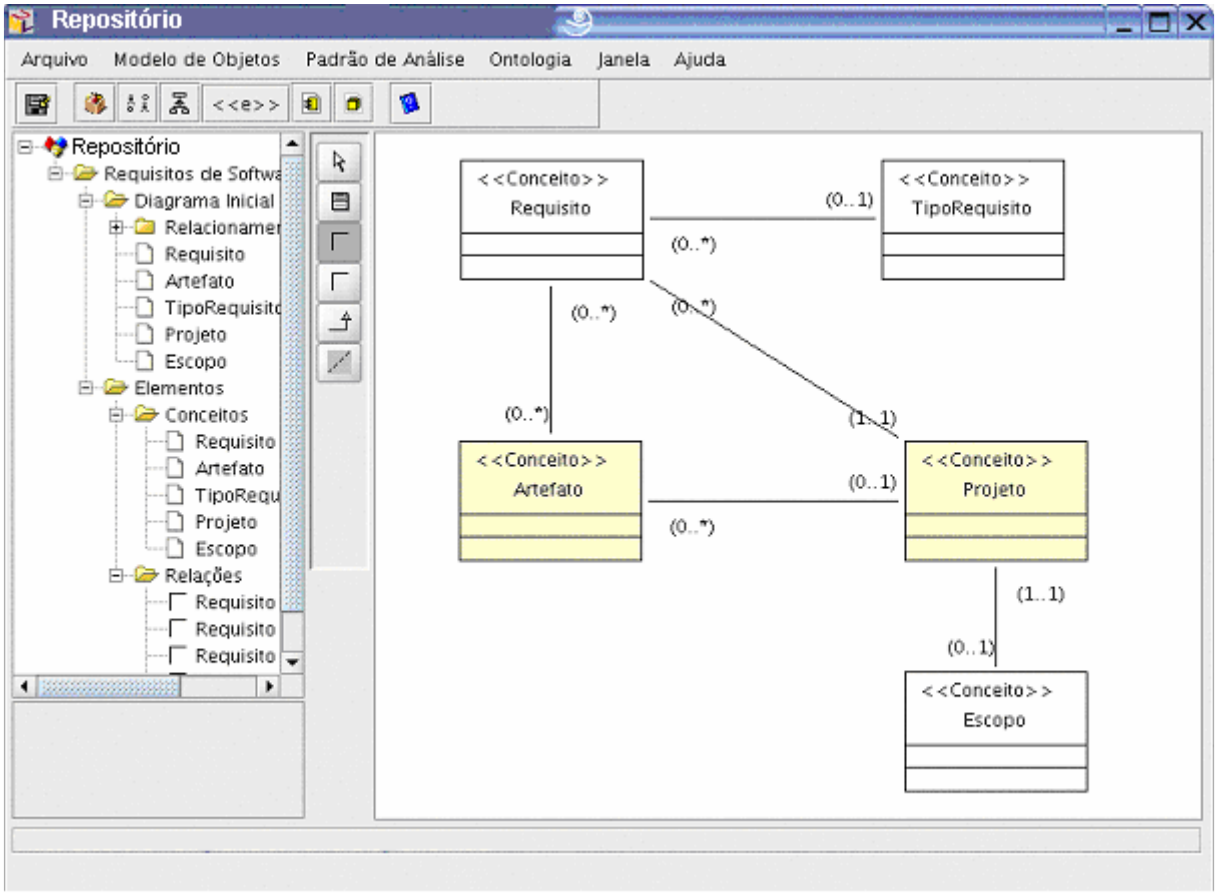


Figura 5.26 – Modelagem de Ontologias em OODE

A Figura 5.27 ilustra o reúso de modelos de objetos de projetos similares. De forma semelhante ao reúso de requisitos, inicialmente recuperam-se os projetos similares e, a partir da escolha de um desses projetos, são listados os modelos de objetos correspondentes. De posse desses modelos, o desenvolvedor pode, por exemplo, solicitar a importação de um deles para o projeto que está sendo trabalhado. Assim todos os diagramas e elementos do modelo importado podem ser reutilizados.

De modo análogo, podem-se buscar ontologias existentes no repositório de conhecimento para posterior reúso, como mostra a Figura 5.28. Assim, ontologias podem ser recuperadas, por exemplo, pelo domínio de aplicação e ter toda sua conceituação importada para a construção de modelos de objetos, ou podem ser apenas abertas para uma consulta em seus diagramas.

Padrões de análise podem ser reutilizados de modo semelhante. Eles podem ser recuperados pelos seus domínios de aplicação e podem ter todos os diagramas e elementos de modelo importados para a construção de um modelo de objetos, ou apenas para serem abertos para que as soluções que eles documentam sejam consultadas e, se possível, adotadas.

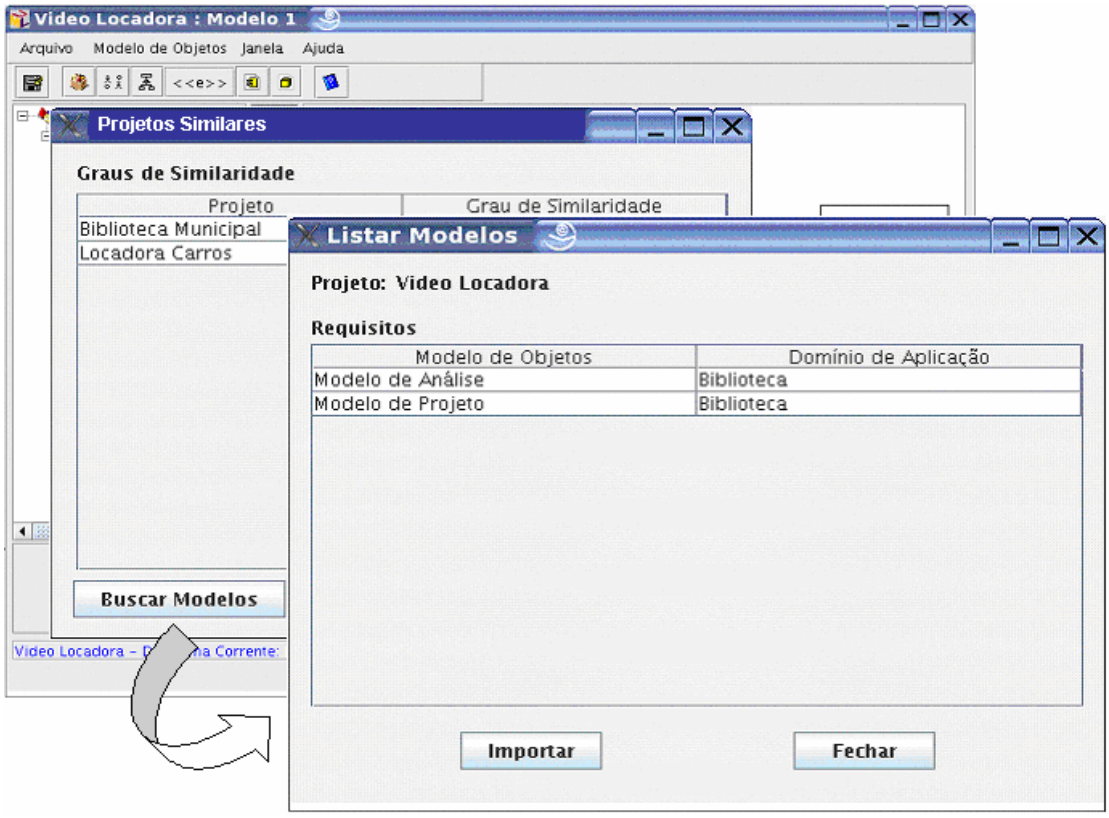


Figura 5.27 – Reúso de Modelos de Objetos

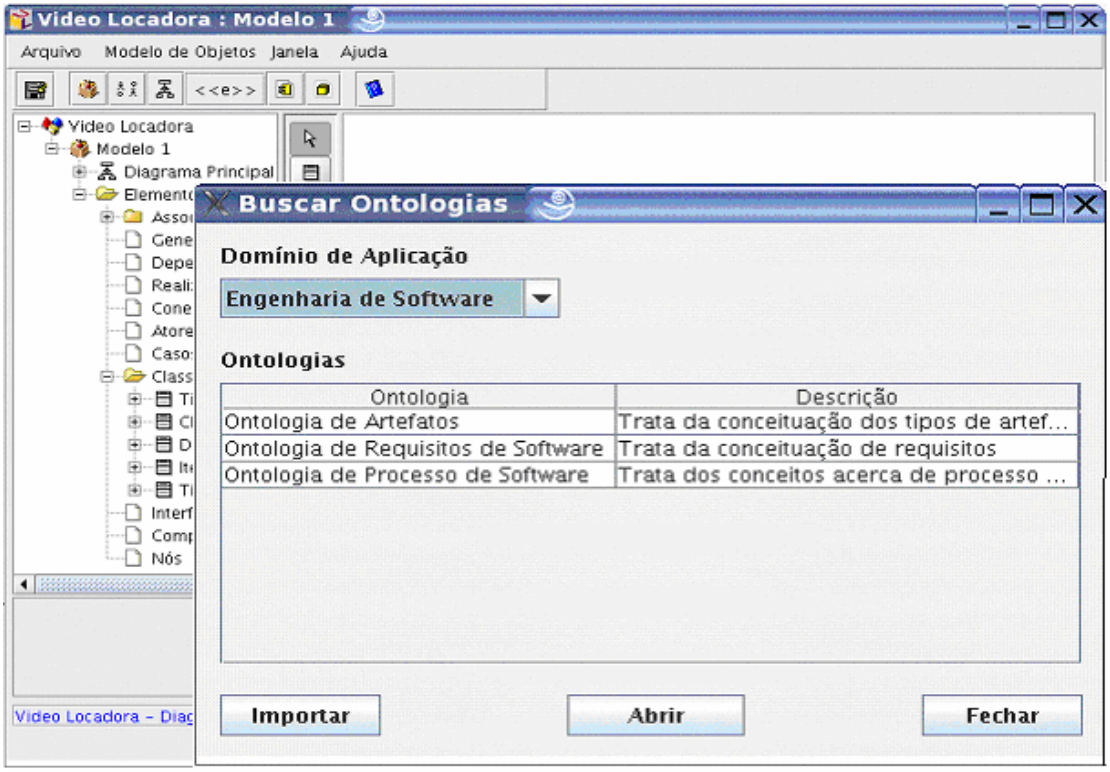


Figura 5.28 – Reúso de Ontologias

Uma vez que modelos de objetos e padrões de análise têm muitas características em comum - tanto que esses foram implementados como especializações daqueles - decidiu-se ilustrar apenas os aspectos que os diferenciavam. Dessa maneira, a Figura 5.29 ilustra a caracterização de um padrão de análise, na qual são dadas informações relevantes para quem for reutilizar a solução documentada no padrão.

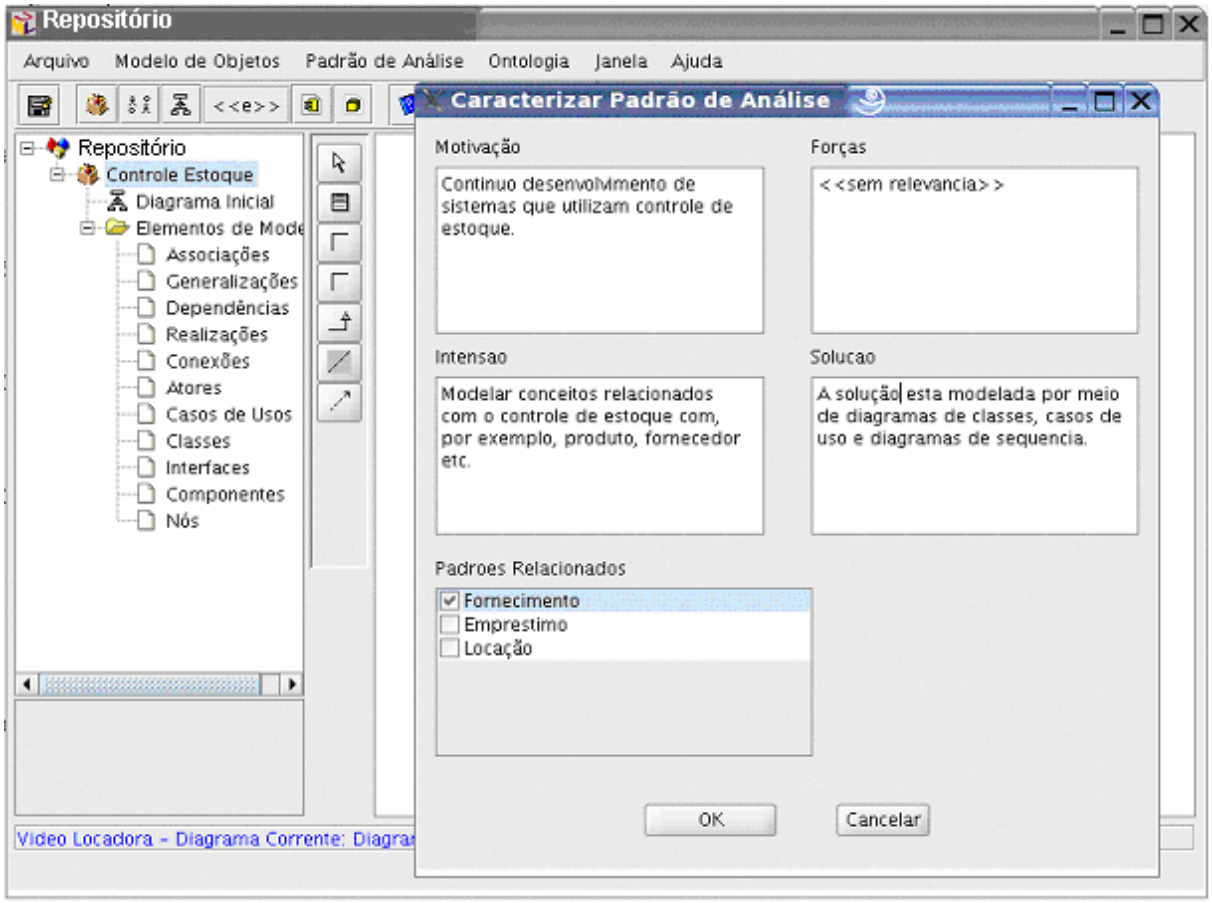


Figura 5.29 – Caracterização de Padrão de Análise

Capítulo 6

Considerações Finais

O objetivo deste capítulo é tecer comentários finais acerca do trabalho realizado, sobretudo no que se refere às conclusões e perspectivas futuras de pesquisas, levantadas a partir dos resultados e idéias que surgiram ao longo da realização deste trabalho.

6.1 Conclusões

Ao longo de todo o período da pesquisa que resultou neste trabalho, muito se dedicou ao estudo das áreas de Engenharia de Requisitos (ER), Gerência de Conhecimento (GC), Reutilização de Software, Ontologias, Padrões de Análise e Ambientes de Desenvolvimento de Software (ADSs).

Cada uma dessas áreas, em seus progressos e limitações, tem sido estudada, pesquisada e combinada com outras, de maneira a se conseguir novas possibilidades ou novos arranjos de solução para diversos problemas, como, por exemplo, integração de sistemas e de bases de dados, comunicação entre pessoas e produção de software de maior qualidade.

Este trabalho partiu dessas várias pesquisas, que foram conduzidas nos mais diversos contextos e sob diferentes pontos de vista, e seguiu seu rumo no sentido de unir alguns de seus elementos no sentido de prover apoio baseado em gerência de conhecimento às atividades do processo de Engenharia de Requisitos no contexto do ambiente de desenvolvimento de software ODE (*Ontology-based Software Development Engineering*).

Assim, no contexto de ODE, foi possível explorar a questão da integração de ferramentas com base em ontologias e a utilização de uma infra-estrutura de gerência de conhecimento constituída e integrada ao ambiente. Neste sentido, são contribuições deste trabalho:

1) Desenvolvimento de uma Ontologia de Requisitos de Software (NARDI et al., 2006), a qual foi construída de forma integrada às outras ontologias do ambiente visando à reutilização de conceituações e à integração de ferramentas;

2) Construção de *ReqODE*, uma ferramenta de apoio à Engenharia de Requisitos, que foi construída com base na Ontologia de Requisitos de Software e de maneira

integrada às outras ferramentas de ODE. Tal ferramenta fornece funcionalidades para o cadastro e controle de requisitos, focando, principalmente, em questões relacionadas à rastreabilidade;

3) Integração entre academia e iniciativa privada no sentido de definir requisitos base para a construção de *ReqODE*, de modo a atender às empresas que buscam ferramentas de apoio à ER;

4) Definição de uma abordagem de apoio ao reúso de itens de conhecimento no contexto da Engenharia de Requisitos, o que levou à especialização da infra-estrutura de Gerência de Conhecimento de ODE no sentido de se criarem funcionalidades integradas às ferramentas de modelagem (*OODE*) e de requisitos (*ReqODE*), e de dar suporte à reutilização de ontologias, padrões de análise e modelos de objetos como itens formais de conhecimento. O uso dessa infra-estrutura em *ReqODE* incrementa a ferramenta, fornecendo um apoio mais sofisticado à ER, baseado no reúso de conhecimento previamente criado por desenvolvedores que utilizam *ReqODE* em uma organização;

5) Alteração da ferramenta de modelagem OODE para permitir, além da elaboração de modelos de objetos, a construção de padrões de análise e ontologias sob uma mesma linguagem gráfica de representação, a UML.

De maneira geral, a maior contribuição deste trabalho se deu no sentido de que ele foi inédito no contexto de ODE, por tratar questões de Engenharia de Requisitos, até então não trabalhadas em ODE, a saber: trato com requisitos e o reúso de conhecimento no contexto da ER. Sendo assim, a condução desta pesquisa abriu espaço para uma série de novas possibilidades de trabalhos, que podem explorar outras facetas da Engenharia de Requisitos.

Além disso, analogamente às pesquisas precedentes no ambiente ODE, este trabalho baseia-se no reúso de resultados anteriores, contribuindo para a construção de um ambiente integrado, no qual as funcionalidades complementam umas as outras.

6.2 Perspectivas Futuras

Ao cabo deste trabalho, estabeleceu-se tanto um apoio ao trato com requisitos quanto ao reúso de itens de conhecimento no contexto da Engenharia de Requisitos. No entanto, uma

vez que esta pesquisa é inicial no contexto de ODE, ainda que tal apoio tenha sido definido, algumas arestas restaram e podem ser aparadas por trabalhos futuros.

Neste sentido, algumas melhorias podem ser trabalhadas em OODE no sentido de tornar a elaboração de ontologias mais abrangente. Para tanto, pode-se integrar a essa ferramenta as outras funcionalidades de ODEd (MIAN, 2003) (SOUZA, 2004), como, por exemplo, o apoio à axiomatização e o uso de uma máquina de inferência para apoiar a validação de ontologias (SOUZA, 2004). Além disso, pode-se fazer com que OODE passe a dar suporte ao uso de OWL (*Web Ontology Language*) (W3C, 2006) para a descrição de ontologias, a fim de facilitar sua integração com outras ferramentas de modelagem de ontologias, como o Protégé, por exemplo.

Tomando-se o processo de Engenharia de Requisitos, ou seja, as atividades de Levantamento de Requisitos, Análise e Negociação, Documentação, Verificação e Validação e Gerência de Requisitos, novos trabalhos podem ser conduzidos no âmbito de *ReqODE* no sentido de ampliar o apoio à ER. Assim, pode-se citar, por exemplo, que tal ferramenta não fornece apoio à negociação de requisitos. Para tanto, poder-se-ia fornecer um apoio integrado por meio de ferramentas de *groupware* como *chat* e fórum que possibilitasse a comunicação entre os envolvidos em um projeto de modo a resolverem questões acerca de requisitos. Tais discussões poderiam ser armazenadas para registrar evidências, ou recuperadas a fim de reutilizar conhecimento organizacional.

Ainda que este trabalho tenha, na Ontologia de Requisitos de Software, tratado da conceituação acerca de características de qualidade de requisitos, esse tópico não foi abordado em termos de funcionalidades em *ReqODE*. Assim, surge mais uma possibilidade de trabalho no sentido de apoiar as atividades de verificação e validação de requisitos. Dessa maneira, poderiam ser trabalhadas, por exemplo, funcionalidades que disponibilizassem listas de verificação e validação a serem preenchidas pelos desenvolvedores com base nas características de qualidade abordadas pela organização de software.

No que tange à atividade de gerência de requisitos, o apoio fornecido por *ReqODE* consiste, basicamente, da definição de ligações de rastreabilidade e geração de um relatório geral exibindo tais ligações. Assim, pode-se explorar essa base constituída no sentido de fornecer relatórios mais detalhados de rastreabilidade. Além disso, pelo fato da ferramenta de Gerência de Configuração de ODE não estar integrada ao ambiente, neste trabalho não foi possível estabelecer um controle de alterações de requisitos. Dessa maneira, como trabalhos

futuros, pode-se integrar a ferramenta de Gerência de Configuração e adequá-la para tratar requisitos.

Além de melhorias que podem ser trabalhadas para tornar as funcionalidades de reúso de itens de conhecimento mais abrangentes e das possibilidades de expansão do apoio ao processo de Engenharia de Requisitos, ao longo de toda a pesquisa destacaram-se algumas áreas afins que, embora não exploradas, merecem ser citadas a fim de documentar possibilidades de outras pesquisas.

Neste sentido, podem-se citar estudos acerca de integração de ontologias, que podem se dar quando uma nova ontologia é construída reutilizando-se outras disponíveis, quando se constrói uma ontologia pela fusão de várias outras em uma única que unifica todas elas ou quando se constrói uma aplicação que utiliza uma ou mais ontologias (GÓMEZ-PÉREZ et a., 1999). Assim, ainda que este trabalho tenha trabalhado com as três questões acima, elas não foram exploradas em detalhes, pois o objetivo principal era fornecer uma infra-estrutura de construção e reúso de ontologias. Em trabalhos futuros, portanto, pode-se estudar mais a fundo cada uma dessas possibilidades de integração de ontologias.

Ainda considerando a integração de ontologias, durante este trabalho investigou-se a possibilidade de se integrar ontologias de domínio e ontologias de tarefas para a derivação de ontologias de aplicação. No entanto, tal pesquisa não avançou, pois se constatou que ainda há poucos trabalhos sobre a utilização de ontologias de tarefas e que o escopo dessa pesquisa iria além do de uma dissertação de mestrado. Porém, como resultados de discussões e leituras adicionais, foram encontradas algumas pesquisas que parecem promissoras para o avanço na área e que tratam de *workflow patterns* (WORKFLOW, 2006) (AALST, 2006), ou seja, padrões que visam a descrever fluxos comuns a uma série de contextos. Essa poderia, portanto, ser uma possibilidade para descrever ontologias de tarefas. A partir da descrição de ontologias de tarefas, poder-se-ia estudar como combiná-las com ontologias de domínio, no sentido de prover apoio para a elaboração de modelos da Engenharia de Requisitos.

Como muito se falou sobre disseminação pró-ativa de itens de conhecimento, outro trabalho interessante seria trabalhar com a tecnologia de agentes na disseminação de itens de conhecimento na Engenharia de Requisitos. Como em ODE existe uma infra-estrutura de apoio à construção de agentes, denominada AgeODE (PEZZIN et al., 2004), um trabalho interessante seria utilizá-la a fim de criar agentes disseminadores de conhecimento, de modo que, enquanto um desenvolvedor estivesse construindo modelos ou gerenciando requisitos,

agentes atuando em ODE poderiam fornecer itens de conhecimento relevantes para a execução de seu trabalho.

Ainda em relação ao reúso de conhecimento no âmbito de modelos, um trabalho interessante seria definir um esquema de anotações em modelos de modo que um desenvolvedor, ao definir elementos como classes, associações, casos de uso etc, pudesse criar anotações nesses elementos de modo a representar decisões de projeto, dificuldades encontradas, aspectos avaliados, enfim, o *design rationale,* envolvido na construção do modelo. Assim, quando outro desenvolvedor fosse reutilizar o modelo ele poderia ter disponível uma série de informações capturadas durante a sua elaboração.

Assim, ainda que muito trabalho tenha sido feito ao longo desta pesquisa, muitos outros trabalhos podem ser desenvolvidos a partir das idéias abordadas por ela.

Referências Bibliográficas

AALST W. V. D. Home page. Disponível em: http://is.tm.tue.nl/staff/wvdaalst/. Acesso em: 24/05/2006.

ANGELE, J.; DECKER, S.; PERKUHN, R.; STUDER, R. **Modeling Problem-Solving Methods in New KARL.** In: Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop. Ranff, Canada. November 9-14.1996, pp. 1-18.

ARANGO, G. **Domain Analysis Methods.** In: Workshop on Software Architecture, Los Angeles: USC Center for Software Engineering, 1994.

ARANTES, D. O.; FALBO, R.A. **Gerenciando o Conhecimento Proveniente de Interações entre Membros de Organizações de Software.** II Workshop de Tecnologia de Informação e Gerência de Conhecimento, III Simpósio Brasileiro de Qualidade de Software - SBQS'2004, Brasília, Brasil, Junho 2004.

BARROCA, L., GIMENES, I.. M. S.; HUZITA, E. H. M. **Conceitos Básicos.** In: Desenvolvimento Baseado em Componentes: Conceitos e Técnicas. 1ª. Edição. Editora Ciência Moderna LTDA. Rio de Janeiro. 2005. pp 1-26.

BENJAMINS, V. R.; DIETER, F.; PÉREZ, A. G. **Knowledge Management through Ontologies**. In: 2nd International Conference on Practical Aspects of Knowledge Management. Basel, Switzerland, 29-30, Outubro, 1998.

BERTOLLO, G., **Definição de Processos em um Ambiente de Desenvolvimento de Software**, Dissertação de Mestrado, Mestrado em Informática, Universidade Federal do Espírito Santo, Maio de 2006.

BREUKER, J.; VAN DE VELDE, W. **CommonKADS Library for Expertise Modelling.** IOS Press. 1994.

CARREIRO, O.M., **Modelagem de Interação em OODE Segundo o Meta-modelo da UML**, Projeto de Graduação, Curso de Ciência da Computação, UFES, 2005.

CARVALHO, A. L., **SIDER: Um Ambiente de Desenvolvimento de Software Orientado ao Domínio de Siderurgia.** Dissertação de Mestrado, Mestrado em Informática, UFES, 2002.

CARVALHO, V.A., ARANTES, L.O., FALBO R.A., **EstimaODE: Apoio a Estimativas de Tamanho e Esforço no Ambiente de Desenvolvimento de Software ODE**. Anais do V Simpósio Brasileiro de Qualidade de Software, p. 12-26, Vila Velha, Brasil, Maio 2006.

CHANDRASEKARAN, B.; JOSEPHSON, J. R.; BENJAMINS. V. R. **Ontology of Tasks and Methods.** Baff Knowledge Acquisition Workshop. 1998.

CHRISSIS, M. B., KONTAD M., SHRUM S, S., **CMMI: Guidelines for Process Integration and Product Improvement**, Addison Wesley, 2003.

CIMA, A. M., WERNER, C. M. L. **A Reutilização de Conhecimento Abstrato e a Análise de Domínio.** 1997. Relatório Técnico ES-432/97 – Engenharia da Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro.

COTA, R.I.; MENEZES, C.S.; FALBO, R.A. **Modelagem Organizacional Utilizando Ontologias e Padrões de Análise.** In: VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software - IDEAS'2004, pp. 56-67, Arequipa, Perú, Maio 2004.

COTA, R. I. **Modelagem Computacional para Gestão Empresarial. Dissertação de Mestrado.** Universidade Federal do Espírito Santo (UFES), Departamento de Informática (DI). Vitória-ES, 2002.

CROFTS, M.; SMITH, R.; FRAUNHOLZ, B. **Global Software Development: The Next RE Frontier?** AWRE'04 9th Australian Workshop on Requirements Engineering, 12.1 - 12.12. 2004.

DAVENPORT, T. H.; PRUSAK, L., **Working Knowledge: How Organizations Manage What They Know**, Harvard Business School Press, Boston, MA, 1998.

DEVEDZIC, V. **Ontologies: Borrowing from Software Patterns.** ACM intelligence Magazine, Fall 1999, pp. 14-24.

DUARTE, K. C. **Um *Framework* de Reuso no Domínio de Qualidade de Software**. Dissertação de Mestrado. Departamento de Informática (DI) da Universidade Federal do Espírito Santo (UFES). Vitória-ES. Junho de 2001.

FALBO, R.A., RUY, F.B., MORO, R.D., **Using Ontologies to Add Semantics to a Software Engineering Environment**. 17th International Conference on Software Engineering and Knowledge Engineering, SEKE'2005, p. 151 - 156, Taipei, China, July 2005.

FALBO, R.A., RUY, F.B., BERTOLLO, G., TOGNERI, D.F. **Learning How to Manage Risks Using Organizational Knowledge**. Advances in Learning Software Organizations (Proceedings of the 6th International Workshop on Learning Software Organizations - LSO'2004), Melnik G. and Holz, H. (Eds.): LNCS 3096, pp. 7-18, Springer-Verlag Berlin Heidelberg, Banff, Canada, June 2004.

FALBO, R.A., RUY, F.B., PEZZIN, J., DAL MORO R. **Ontologias e Ambientes de Desenvolvimento de Software Semânticos**. Actas de las IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería Del Conocimiento, JIISIC'2004, Volumen I, pp. 277-292, Madrid, España, Noviembre 2004.

FALBO, R.A., ARANTES, D.O., NATALI, A.C.C. **Integrating Knowledge Management and Groupware in a Software Development Environment** Proceedings of the 5th International Conference on Practical Aspects of Knowledge Management - PAKM'2004, Karagiannis, D. , Reimer, U. (Eds.): LNAI 3336, pp. 94-105, Springer-Verlag Berlin Heidelberg, Vienna, Austria, December 2004.

FALBO, R. A. **Experiences in Using a Method for Building Domain Ontologies**. Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering, SEKE'2004, International Workshop on Ontology In Action, OIA'2004. Banff, Alberta, Canada, June 2004.

FALBO, R.A.; NATALI, A.C.C.; MIAN, P.G.; BERTOLLO, G.; RUY, F.B. **ODE: Ontology-based software Development Environment**, IX Congreso Argentino de Ciencias de la Computación, p. 1124-1135, La Plata, Argentina, outubro 2003.

FALBO, R.A.; GUIZZARDI, G.; DUARTE, K.C. **An Ontological Approach to Domain Engineering**. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE'2002, pp. 351- 358, Ischia, Italy, 2002.

FALBO, R. **A. Integração de Conhecimento em um Ambiente de Desenvolvimento de Software**. Tese de Doutorado, COPPE/UFRJ, RJ, Dezembro, 1998.

FIORINI, S. T., STAA, A. V., BAPTISTA, R. M. **Engenharia de Software com CMM**. Brasport, Rio de Janeiro, 1998.

FOWLER, M. **Analysis Patterns: Reusable Object Models.** Addison-Wesley Professional Computing Series, 1997.

GAMMA, E., HELM R., JOHNSON R., VLISSIDES, J. **Design Patterns - Elements of Reusable Object-oriented Software.** Addison-Wesley Professional Computing Series, 1995.

GEYER-SCHULZ, A.; HAHSLER, M. **Software Engineering with Analysis Patterns**. Working Paper 01/2001, Working Papers on Information Processing and Information Management, Institut für Informationsverarbeitung und -wirtschaft, Wirtschaftsuniversität Wien, Augasse 2-6, 1090 Wien, Austria, November 2001.

GIMENES, I.. M. S.; HUZITA, E. H. M. **Apresentação.** In: Desenvolvimento Baseado em Componentes: Conceitos e Técnicas. 1ª. Edição. Editora Ciência Moderna LTDA. Rio de Janeiro. 2005. pp XI-XVI.

GIMENEZ, I. **Processo de Desenvolvimento de Software.** In: Qualidade de Software: Teoria e Prática. 1ª. Edição. Editora Prentice Hall. São Paulo. 2001. pp 43-47.

GÓMEZ-PÉREZ, A.; MARTINS, J. P. **Some Issues on Ontology Integration**. Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, agosto de 1999.

GRUBER, T. R. **Toward Principles for the Design of Ontologies used for Knowledge Sharing.** Int. J. Human-Computer Studies, v. 43, n. 5/6, 1995.

GUARINO, N. **Formal Ontology in Information System**. In: Proceedings of the First Int. Conference on Formal Ontology in Information Systems, Trento, Italy, June 1998. p.3-15.

GUIZZARDI, G. **Desenvolvimento para e com Reuso: Um Estudo de Caso no Domínio de Vídeo sob Demanda**. Dissertação (Mestrado em Informática), Universidade Federal do Espírito Santo (UFES), 2000.

HARRISON, W., OSSHER, H., TARR, P. **Software Engineering Tools and Environments: A Roadmap**. Proceedings of The Future of Software Engineering (ICSE'2000). Limerick, Ireland, p. 263 – 277, 2000.

HENNINGER, S.; LAPPALA, K.; RAGHAVENDRAN, A. **An Organizational Learning Approach to Domain Analysis**. In: Seventeenth International Conference on Software Engineering (Seattle, WA), 1995 pp. 95-104.

IEEE Standards Boards. IEEE Std 830-1998: **IEEE Recommended Practice For Software Requirements Specifications**, jun. 1998.

ISO/IEC 12207, **Information Technology - Software life cycle processes**, 1995.

ISO/IEC 12207, **Information Technology - Software life cycle processes, Amendment 1**, 2002.

ISO/IEC 12207, Information **Technology - Software life cycle processes, Amendment 2**, 2004.

JARKE, M. **Requirements Tracing**. Communications of the ACM, v. 41, n. 12, dec. 1998.

KOTONYA, G.; SOMMERVILLE, I. **Requirements Engineering:** Process and Techniques. 1º Edição. Editora Wiley. 1998.

LAMSWEERDE, A. V. **Requirements Engineering in the Year 00: A Research Perspective**. Proceedings of the 22nd international conference on Software engineering (ICSE00). Limerick, Ireland, p. 5-19, jun. 2000. [on line]. Disponível: http://www.acm.org/pubs/contents/proceedings/soft/337180/ [capturado em 30 dez. 2000]

LOPES, P. S. N. D. **Uma Taxonomia da Pesquisa na Área de Engenharia de Requisitos**. Dissertação de Mestrado. Instituto de Matemática e Estatística da Universidade de São Paulo. São Paulo. Março de 2002.

MALDONADO, J. C.; FABBRI, S. C. F. **Verificação e validação de software.** In: Qualidade de Software: Teoria e Prática. 1ª. Edição. Editora Prentice Hall. São Paulo. 2001. pp 66-73.

MIAN, P.G. **ODEd: Uma Ferramenta de Apoio ao Desenvolvimento de Ontologias em um Ambiente de Desenvolvimento de Software**. Dissertação de Mestrado, Mestrado em Informática, UFES, Abril/2003.

MIAN, P. G.; FALBO, R. A. **Building Ontologies in a Domain Oriented Software Engineering Environment.** IX Congreso Argentino de Ciencias de la Computación, p. 930-941 La Plata, Argentina, Outubro 2003.

MIZOGUCHI, R.; WELKENHUYSEN, J. van; IKEDA, M. **Task ontology for reuse of problem-solving knowledge**. In N.J.I. Mars, editor, Proceedings of the 2nd International Conference on Knowledge Building and Knowledge Sharing(KB & KS'95), Twente, The Netherlands, pages 46-57, Amsterdam, NL., 1995. IOS Press.

NARDI, J. C.; FALBO, R.A. **Uma Ontologia de Requisitos de Software.** IX Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software, La Plata, Argentina, abril de 2006.

NARDI, J. C., Modelagem de Estados em OODE Segundo o Meta-Modelo da UML**. Projeto Final de Curso, Ciência da Computação, Universidade Federal do Espírito Santo (UFES), 2003.**

NATALI, A.C.C.; FALBO, R.A. **Gerência de Conhecimento em ODE**. Anais do XVII Simpósio Brasileiro de Engenharia de Software (SBES'03), Manaus, Outubro de 2003.

NATALI, A. C. C. **Uma Infra-estrutura para Gerência de Conhecimento em um Ambiente de Desenvolvimento de Software**. Dissertação (Mestrado em Informática), Universidade Federal do Espírito Santo (UFES), 2003.

NONAKA, I., TAKEUCHI, H., **Criação de Conhecimento na Empresa – Como as Empresas Japonesas Geram a Dinâmica da Inovação**, Editora Campus, 1997.

NUNES, B. V. **Integrando Gerência de Configuração de Software, Documentação e Gerência de Conhecimento em um Ambiente de Desenvolvimento de Software**. 2005. Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo.

NUNES, V.B.; SOARES, A.O.; FALBO, R.A. **Apoio à Documentação em um Ambiente de Desenvolvimento de Software**. VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software, IDEAS'2004, pp 50-55, Arequipa, Peru, Maio 2004.

NUSEIBEH, B.; EASTERBROOK S. **Requirements engineering: A roadmap.** The Future of Software Engineering., ICSE-2000,Limerick, Ireland, June 2000, Finkelstein A (ed.). ACM Press: New York, 2000.

O´LEARY, D.E.; STUDER, R. **Knowledge Management: An Interdisciplinary Approach.** IEEE Intelligent Systems, January/February, vol. 16, No. 1, 2001.

PALUDO, M. A., BURNET, R. C. **Desenvolvimento Baseado em Componentes e Padrões.** In: Desenvolvimento Baseado em Componentes: Conceitos e Técnicas. 1ª. Edição. Editora Ciência Moderna LTDA. Rio de Janeiro. 2005. pp 199-231.

PEZZIN, J.; FALBO, R. A. **AgeODE: Uma Infra-Estrutura para Apoiar a Construção de Agentes para Atuarem no Ambiente de Desenvolvimento de Software ODE.** Actas de las IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento, JIISIC'2004, Volumen II, pp. 389-404, Madrid, España, Noviembre 2004.

PFLEEGER, S.L. **Software Engineering: Theory and Practice.** 2nd edition, New Jersey: Prentice Hall, 2001.

PRESSMAN, R. S. **Engenharia de Software**. 5º edição. Rio de Janeiro: McGrawHill, 2002.

PRIETO-DIAZ, R.; **Status Report: Software Reusability**; Software, IEEE Volume 10, Issue 3, May 1993 Page(s):61 – 66 Digital Object Identifier 10.1109/52.210605.

PRIETO-DIAZ, R.; **Domain Analysis: An Introdution.** Software Engineering Notes vol.15 nº 2 pp. 47, april, 1990.

ROBERTSON, S.; ROBERTSON, J. **Mastering the Requirements Process.** 1º Edição. Editora ACM Press, Addison Wesley. 1999.

RUS, I.; LINDVALL, M. **Knowledge Management in Software Engineering**. IEEE Software 19(3) May/Jun. 26-38. 2002.

RUY, F. B., **Semântica em um Ambiente de Desenvolvimento de Software**, Dissertação de Mestrado, Mestrado em Informática, Universidade Federal do Espírito Santo, Maio de 2006.

RUY, F.; BERTOLLO, G.; FALBO, R.A. **Knowledge-based Support to Process Integration in ODE**. Clei Electronic Journal, Volume 7, Number 1, June 2004.

SAYÃO, M.; Von STAA, A. & LEITE, J. C. S. P. **Qualidade em Requisitos – relatório técnico 47/03**, série Monografias em Ciência da Computação, DI/PUC-Rio, 2003.

SILVA, P. B., **Adequação da Ferramenta de Documentação de ODE a uma Ontologia de Artefato**, Projeto de Graduação, Curso de Ciência da Computação, Universidade Federal do Espírito Santo, 2004.

SILVA, B. C. C. **Adequação ao meta-modelo da UML em OODE: Apoio à Elaboração de Diagramas de Classe e Casos de Uso.** Projeto de Graduação, Curso de Ciência da Computação. Universidade Federal do Espírito Santo (UFES), novembro de 2003.

SOFTEX, S. **MPS.BR (Melhoria de Processo do Software Brasileiro): Guia Geral**. Versão 1.0. abril de 2005.

SOUZA, V. E. S. **Estendendo ODEd: Axiomatização e Adequação ao Meta-Modelo da UML.** Projeto de Graduação, Curso de Ciência da Computação. Universidade Federal do Espírito Santo (UFES), abril de 2004.

SOMMERVILLE, I. **Engenharia de Software.** 6º Edição. São Paulo: Pearson – Addilson Wesley. 2004.

SPYNS, P.; MEERSMAN, R.; JARRAR, M. **Data Modelling Versus Ontology Engeneering**. SIGMOD Rec., Vol. 31, No. 4. (December 2002), pp. 12-17.

STAAB, S., STUDER, R., SCHURR, H. P., and SURE, Y. **Knowledge Processes and Ontologies**. IEEE Intelligent Systems, January/February, Vol. 16, No. 1, 2001.

TOGNERI, D. F. **Apoio Automatizado à Engenharia de Requisitos Cooperativa**. Dissertação (Mestrado em Informática), Universidade Federal do Espírito Santo (UFES), Vitória, 2002.

TORANZO, M. A.; CASTRO, J.; MELLO, E. **Uma Proposta para Melhorar o Rastreamento de Requisitos.** In: V Workshop on Requirements Engineering, 2002, Valencia. Proceedings of the V Workshop on Requirements Engineering. Valencia: Universidad Politecnica de Valencia, 2002. v. 1. p. 194-209.

TORRES, A. H. S. **Captura e Disseminação do Conhecimento em Projetos de Software.** Dissertação de Mestrado. Programa de Pós-Graduação em Gestão do Conhecimento e Tecnologia da Informação. Brasília, 2006.

TRAVASSOS, G.H. **O Modelo de Integração de Ferramentas da Estação TABA.** Tese de Doutorado. Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Março/1994.

ZAVE, P. **Classification of research efforts in requirements engineering.** ACM Computing Surveys Journal, New York, v. 29, n. 4, p. 315-321, dez. 1997.

ZLOT, F. **Conhecimento de Tarefa em Ambientes de Desenvolvimento de Software Orientados a Domínio.** Dissertação de Mestrado. Programa de Pós-Graduação de Engenharia da UFRJ. Rio de Janeiro, Brasil, junho de 2002.

W3C, **Web Ontology Language (OWL).** http://www.w3.org/2004/OWL/, capturado em 25/05/2006.

WERNER, C. M. L.; BRAGA, R. M. M. **A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes.** In: Desenvolvimento Baseado em Componentes: Conceitos e Técnicas. 1ª. Edição. Editora Ciência Moderna LTDA. Rio de Janeiro. 2005. pp 57-103.

WIEGERS, K. E. **Software Requirements: Pratical techniques for gathering and managing requirements throughout the product development cycle**. Microsoft Press, Redmond, Washington, 2003. Second Edition.

WORKFLOW Patterns. Disponível em: http://is.tm.tue.nl/research/patterns/. Acesso em: 24/05/2006.