FUNDAÇÃO EDUCACIONAL EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

Raul de Abreu Medeiros Júnior

Uma Ontologia para Engenharia de Requisitos de Software

Fortaleza

2006

FUNDAÇÃO EDUCACIONAL EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

Raul de Abreu Medeiros Júnior

Uma Ontologia para Engenharia de Requisitos de Software

Dissertação apresentada ao Curso de Mestrado em Informática Aplicada da Universidade de Fortaleza como parte dos requisitos necessários para a obtenção do Título de Mestre em Informática.

Orientador: Arnaldo Dias Belchior

Fortaleza

2006

Uma Ontologia para Engenharia de Requisitos de Software

Banca Examinadora:

_____

Prof. Orientador Arnaldo Dias Belchior, D.Sc.
(Presidente da Banca)

_____

Prof. Co-orientador Pedro Porfírio Muniz Farias, D.Sc.

_____

Prof[a]. Vera Maria Benjamim Werneck, D.Sc.

_____

Prof. José Bezerra da Silva Filho, D.Sc.

FICHA DE CATALOGAÇÃO

MEDEIROS Jr., Raul de Abreu. **Uma Ontologia para Engenharia de Requisitos de Software.** 2006. 105f. (cento e cinco folhas). Dissertação (Mestrado em Informática Aplicada) – Universidade de Fortaleza (UNIFOR). Fortaleza, 2006.

Perfil do Autor: Graduado em Bacharelado em Ciências da Computação pela Universidade Estadual do Ceará (UECE). Analista de requisitos na empresa Serviço Federal de Processamento de Dados.

RESUMO:
Um dos principais objetivos da engenharia de requisitos é construir requisitos de software de fácil entendimento e que sejam consistentes tanto para os clientes, quanto para os desenvolvedores. Para auxiliar no entendimento desses requisitos entre as partes envolvidas, este trabalho propõe uma ontologia para a engenharia de requisitos de software, com o objetivo de facilitar a compreensão de seus conceitos, e propiciar a melhoria da comunicação entre desenvolvedores e clientes, diminuindo assim problemas posteriores ao longo do ciclo de vida do software. A ontologia proposta neste trabalho foi aplicada a três sistemas, dois de domínio de aplicação financeira e um da área comercial.

PALAVRAS-CHAVE:
1) Engenharia de Requisitos; 2) Ontologias; 3) OWL.

DEDICATÓRIA

Dedico este trabalho aos meus queridos pais que sempre me apoiaram e deram provas de carinho, respeito e dedicação.

AGRADECIMENTOS

A Deus que me deu vida, a inteligência, e me supri, de energia para a contínua caminhada em busca das minhas metas.

Ao meu orientador Dr. Arnaldo Dias Belchior, agradeço pela grande paciência e atenção que sempre teve com a minha pessoa.

Ao meu co-orientador Doutor Pedro Porfírio Muniz Farias, que também teve paciência e compreensão para a elaboração desta dissertação.

Agradeço aos meus amados pais, Raul de Abreu Medeiros e Maria Erileide Façanha Barreto Medeiros, pelo enorme carinho e compreensão durante todo este tempo em que estive escrevendo a minha dissertação.

A minha irmã, Cybelle Façanha Barreto Medeiros Linnard, e ao seu esposo Roberto Sérgio Sobreira Linnard, por toda a ajuda que dispensaram que foi muito importante para a conclusão deste trabalho.

Aos meus irmãos, Rômulo Façanha Barreto Medeiros e Régis Façanha Barreto Medeiros, muito obrigado pela força que foram de fundamental importância para a finalização deste trabalho.

Ao SERPRO, por ter me disponibilizado tempo e verba para o financiamento deste curso.

Aos meus amigos, Joaquim, Welton e Elvira, que estiveram me apoiando e incentivando ao longo da realização desta dissertação.

Aos amigos do mestrado, pelas ajudas nos momentos mais difíceis e pelo aprendizado que me proporcionaram.

Aos demais professores do mestrado, pela constante presença e contribuições indiretas.

À secretaria do MIA, pela atenção e presteza sempre imediatas.

E a todos que direta ou indiretamente contribuíram na elaboração desta dissertação.

RESUMO

Um dos principais objetivos da engenharia de requisitos é construir requisitos de software de fácil entendimento e que sejam consistentes tanto para os clientes, quanto para os desenvolvedores. Para auxiliar no entendimento desses requisitos entre as partes envolvidas, este trabalho propõe uma ontologia para a engenharia de requisitos de software, com o objetivo de facilitar a compreensão de seus conceitos, e propiciar a melhoria da comunicação entre desenvolvedores e clientes, diminuindo assim problemas posteriores ao longo do ciclo de vida do software. A ontologia proposta neste trabalho foi aplicada a três sistemas, dois de domínio de aplicação financeira e um da área comercial.

PALAVRAS-CHAVE: Engenharia de Requisitos; Ontologia; OWL.

ABSTRACT

One of the main objectives of the requirements engineering is to construct software requirement as easy to understand as possible and that they are consistent to both customers and to the developments. To assist in the agreement of these requirements between the involved parts, this work propose an ontology for the software requirement engineering, that aims to facilitate the understanding of its concepts, and to improve the communication between developers and customers. This will prevent future problems in the software life cycle. The ontology proposed in this work has been applied to three different systems, two in the financial application systems domain and one on commercial area.

KEYWORDS: Requirements Engineering; Ontology; OWL .

LISTAS DE ILUSTRAÇÕES

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Este capítulo apresenta as principais questões que motivaram a realização deste trabalho, como também os objetivos a alcançar e sua organização.

Atualmente se desenvolvem muitos produtos de software, que estão cada vez mais sofisticados. No entanto, alguns problemas surgidos durante o processo de desenvolvimento ainda continuam trazendo enormes prejuízos para os envolvidos. Uma questão central, comum a alguns problemas decorrentes de falhas no processo de produção ou de manutenção de software, está na precariedade dos processos de produção empregados, principalmente devido à não-observância de princípios da engenharia de requisitos (LEITE, 2003).

1.1 Motivação

A engenharia de requisitos é um processo que envolve todas as atividades exigidas para criar e manter o documento de requisitos de sistema (conjunto de artefatos relacionados aos requisitos), que é a declaração oficial do que é exigido da equipe de desenvolvimento do sistema (SOMMERVILLE, 2003). Um produto de software deverá ser construído de acordo com seus requisitos.

Experiências reais e pesquisas comprovam que um precário entendimento dos requisitos tem levado a inúmeros problemas no processo de desenvolvimento de software (LEITE, 2003). Naturalmente, um dos objetivos principais da engenharia de requisitos é a construção de requisitos de fácil entendimento, tanto para quem os solicitou, quanto para os seus desenvolvedores.

A engenharia de software associa a palavra “requisitos” ao início da produção de software. Quanto mais cedo for descoberta uma inconsistência, menor será o custo para a corrigir. Boehm e Papaccion (1998) relataram que se para achar e corrigir um erro na fase de requisitos custar 1, esse mesmo erro poderá vir a custar 5 para ser corrigido durante a fase de análise / *design*, 10 para ser corrigido durante a fase de codificação, 20 para ser corrigido na fase de teste unitário, e mais de 200 para ser corrigido depois da entrega do sistema. Portanto, constitui uma boa prática a realização de investimentos em engenharia de requisitos, como forma de evitar problemas futuros ou mudanças ao longo do processo de desenvolvimento.

1.2 Objetivo

Este trabalho objetiva poder auxiliar na melhoria do entendimento dos requisitos entre as principais partes envolvidas (clientes / usuários e desenvolvedores). A melhoria do entendimento deverá ser feita por meio da utilização de uma ontologia para o processo de engenharia de requisitos.

Segundo Gruber (1993) "uma ontologia é uma especificação formal e explicita de uma conceituação compartilhada". Nesta definição, entende-se por "conceituação" um modelo abstrato. A palavra "explicita" indica que os elementos da ontologia são claramente definidos e, finalmente, a palavra "formal" exprime que a ontologia deve ser processável por uma máquina. (FENSEL, 2001).

Para Gruninger (1995), uma ontologia é uma especificação de uma conceituação: uma descrição de conceitos e relações que existem em um domínio de interesse. Basicamente, uma ontologia consiste desses conceitos e relações, e suas definições, propriedades e restrições que estão descritas em forma de axiomas.

1.3 Organização do trabalho

Este trabalho está organizado em seis capítulos, incluindo esta introdução, e dois apêndices, descritos resumidamente a seguir:

No Capítulo 2, **Engenharia de Requisitos**, discorrem-se sobre os principais conceitos e fundamentos da engenharia de requisitos.

No Capítulo 3, **Enfoques sobre Ontologia**, apresentam-se definições de ontologia, utilizações da ontologia na ciência da computação, metodologias de construção de ontologia e a liguagem OWL (*Ontology Web Language*) para definição de uma ontologia.

No Capítulo 4, **Uma Ontologia para Engenharia de Requisitos**, apresenta-se uma ontologia que dá suporte à engenharia de requisitos, no que se refere ao levantamento, à documentação, às métricas e à qualidade dos requisitos.

No Capítulo 5, **Aplicação da Ontologia**, exibe-se a aplicação da ontologia proposta no capítulo 4 a um sistema de treinamento.

No Capítulo 6, **Conclusão**, são apresentadas as conclusões e as contribuições deste trabalho, bem como a indicação de futuros trabalhos.

No Apêndice A, Neste apêndice, **Códigos OWL da Ontologia**, são apresentados os códigos da OWL, que foram gerados na construção da ontologia de engenharia de requisitos proposta.

No Apêndice B, Neste apêndice, **Questionário sobre Qualidade de Requisitos de Software**, mostra o questionário aplicado a especialistas em requisitos de software para seleção dos atributos de qualidade de requisitos a serem incluídos na ontologia proposta neste trabalho.

Capítulo 2

ENGENHARIA DE REQUISITOS

Este capítulo aborda a e engenharia de requisitos, que será utilizada como base para a ontologia proposta neste trabalho.

2.1 Requisitos

Os requisitos descrevem o comportamento de um sistema e possuem três propósitos (PFLEGER, 2001):

- Os requisitos permitem aos desenvolvedores explicar seus entendimentos de como o cliente gostaria que o sistema funcionasse.
- Os requisitos informam aos projetistas de software que funções e características o sistema deverá ter.
- Os requisitos orientam a equipe de teste a demonstrar e convencer o cliente que o sistema que está sendo entregue é de fato o que foi pedido.

Segundo Sommerville (2003), requisitos do sistema são descrições de funções e restrições. A engenharia de requisitos é o processo de descobrir, analisar, documentar e verificar essas funções e restrições.

A Figura 2.1 apresenta a disciplina de *Requisitos*, proposta pelo RUP (2003), que permeia todas as etapas do ciclo de vida do software. Essa disciplina tem com finalidades:

- Estabelecer e manter a concordância com clientes e outros envolvidos sobre o que o sistema deve fazer.
- Oferecer aos desenvolvedores do sistema uma compreensão mais adequada dos requisitos do sistema.
- Definir as fronteiras do sistema (ou delimitar o sistema).
- Fornecer uma base para planejar o conteúdo técnico de cada iteração do ciclo de vida.
- Fornecer uma base para estimar o custo e o tempo de desenvolvimento do sistema.
- Definir uma interface para o sistema, focando em necessidades e metas de usuários.

A engenharia de requisitos é um processo que envolve todas as atividades exigidas para criar e manter o documento de requisitos de sistema (conjunto de artefatos relacionados aos

requisitos), que é a declaração oficial do que é exigido da equipe de desenvolvimento do sistema. Deve incluir os requisitos de usuário para um sistema e uma especificação detalhada dos requisitos de sistema.

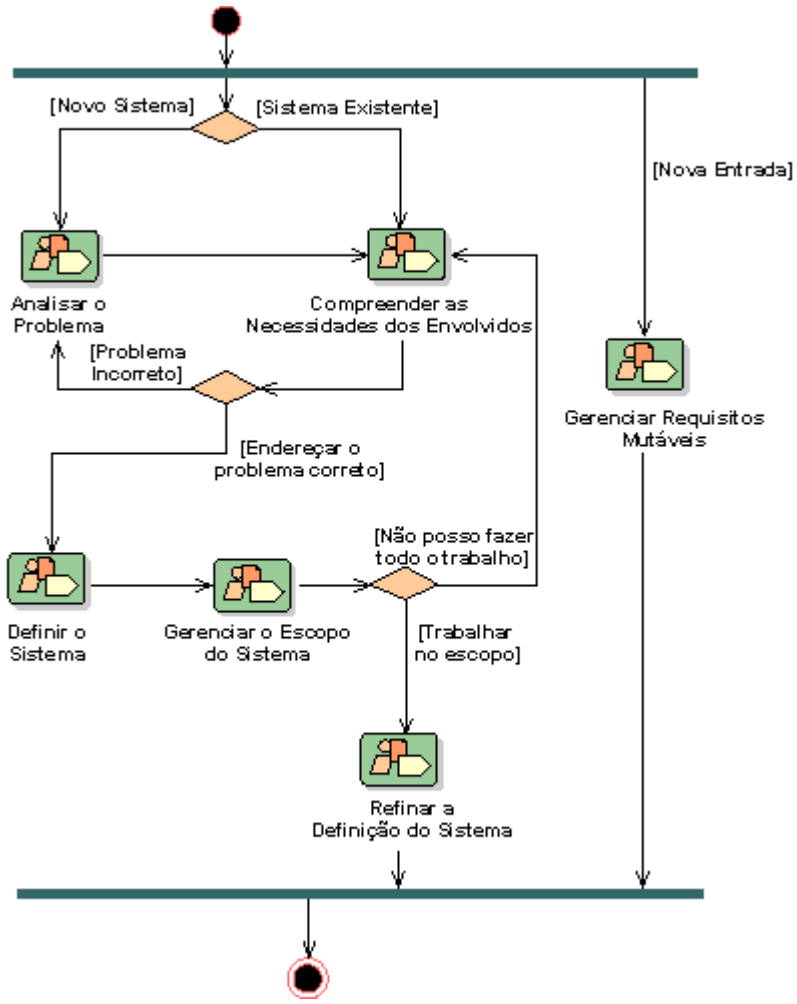

Figura 2.1: Disciplina de Requisitos (RUP, 2003)

Em alguns casos, os requisitos de usuário e de sistema podem ser integrados em uma única descrição. Em outros casos, os requisitos de usuário são definidos em uma introdução à especificação dos requisitos de sistema. Se houver um grande número de requisitos, os requisitos detalhados de sistema poderão ser apresentados como documentos separados. O termo “requisitos do usuário” é usado para designar os requisitos abstratos de alto nível e o termo “requisitos de sistema”, para indicar a descrição mais detalhada do que o sistema deverá fazer (SOMMERVILLE, 2003).

Além destes dois níveis de detalhes, uma descrição mais detalhada (uma especificação de projeto de software) pode ser produzida para associa a engenharia de requisitos e as atividades

de projeto. Os requisitos do usuário, os requisitos de sistema e a especificação de projeto de software podem ser definidos como se segue:

- **Requisitos do usuário**: são declarações, em linguagem natural como também em diagramas, sobre as funções que o sistema deve ter e as restrições sob as quais deve operar. Devem descrever os requisitos funcionais e os requisitos não funcionais de modo que sejam compreensíveis para os usuários. Devem ser legíveis e inteligíveis tanto por usuários que possuem um conhecimento técnico detalhado, como pelos que não possuem esse conhecimento. Devem especificar somente o comportamento externo do sistema, evitando tanto quanto possível as características do projeto de sistema. Conseqüentemente, não devem ser definidos utilizando-se um modelo de implementação. Eles podem ser escritos como o uso de linguagem natural, formulários e diagramas intuitivos simples.
- **Requisitos de sistema**: estabelecem precisa e detalhadamente as funções e as restrições de sistema, através de descrições mais detalhadas dos requisitos do usuário. Podem servir como um contrato entre o comprador do sistema e o desenvolvedor, isto é, um contrato destinado à implementação do sistema e, portanto, deve ser uma especificação completa e consistente de todo o sistema. São utilizados por engenheiros de software como ponto de partida para o projeto de sistema. Em princípio, os requisitos de sistema deveriam definir o que o sistema deveria fazer, e não como ele teria de ser implementado. Contudo, no que se refere aos detalhes exigidos para especificar o sistema completamente, é quase impossível excluir todas as informações de projeto. Algumas razões para se definir algo da implementação:
  - Uma arquitetura inicial do sistema pode ser definida para ajudar a estruturar a especificação de requisitos;
  - Os requisitos de sistema são organizados de acordo com os diferentes subsistemas que constituem o sistema.

O documento de requisitos tem um conjunto diversificado de usuários. Abrange desde a alta gerência da organização, que patrocina o sistema, até as pessoas responsáveis pelo desenvolvimento de software.

Na Figura 2.2, são apresentados alguns *stakeholders* do documento de requisitos e seus papéis.

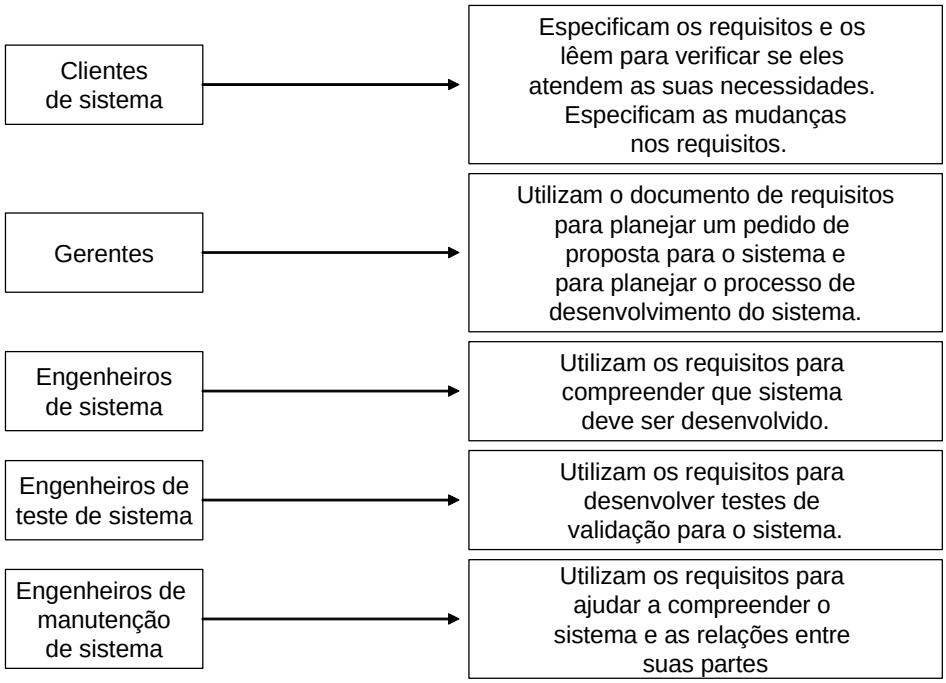

Figura 2.2: *Stakeholders* de um documento de requisitos (SOMMERVILLE, 2003)

Segundo Sommerville (2003) "o termo *stakeholder* é utilizado para se referir a qualquer pessoa que tenha alguma influência direta ou indireta sobre os requisitos do sistema". Dentre os *stakeholder* destacam-se:

- usuários finais que interagirão com o sistema e todo o pessoal, em uma organização, que venha a ser por ele afetado;
- engenheiros que estão desenvolvendo o sistema ou fazendo a manutenção de outros sistemas relacionados;
- gerentes de negócio;
- especialistas no domínio de aplicação;
- a alta gerencia da empresa, que está pagando o sistema;
- analista de requisitos;
- operador do sistema;
- pessoas que irão preparar as entradas para o sistema;
- pessoas que irão utilizar as saídas ou os produtos do sistema;
- gerente das pessoas que irão operar, preparar a entrada, utilizar a saída do sistema.

2.2 Dificuldades nas Atividades de Requisitos

Muitos problemas de engenharia de software se originam da imprecisão na especificação de requisitos (SOMMERVILLE, 2003). Em princípio, a especificação de requisitos de um sistema deve ser completa e consistente. A completitude significa que todas as funções requeridas pelo usuário devem estar definidas. A consistência denota que os requisitos não devem ter definições contraditórias. Na prática, para um sistema complexo e de grande porte, é quase impossível atingir a consistência e a completitude dos requisitos.

A razão disso é, em parte, por causa da complexidade inerente ao sistema e, em parte, porque diferentes *stakeholders* apresentam necessidades inconsistentes. Essas inconsistências podem não serem óbvias, quando da especificação inicial dos requisitos. Os problemas somente emergem depois de uma análise mais aprofundada. À medida que os problemas sejam descobertos durante as revisões ou em fases posteriores do ciclo de vida, esses problemas devem ser corrigidos no documento de requisitos.

Quanto mais cedo for descoberta uma inconsistência, menor será o custo para a corrigir. Boehm e Papaccion (1998) relataram que se para achar e corrigir um erro na fase de requisitos custar 1, esse mesmo erro poderá vir a custar 5 para ser corrigido durante a fase de análise / *design*, 10 para ser corrigido durante a fase de codificação, 20 para ser corrigido na fase de teste unitário, e mais de 200 para ser corrigido depois da entrega do sistema.

O levantamento e a análise de requisitos compõem um processo difícil, por diversas razões (SOMMERVILLE, 2003):

- Os *stakeholders* em geral não sabem na realidade o que querem do sistema computacional, a não ser em termos muito gerais. Eles podem achar difícil articular o que desejam do sistema, podem fazer solicitações não realistas, por não terem noção do custo dessas solicitações.

- Os *stakeholders* em um sistema expressam naturalmente os requisitos em seus próprios termos e com o conhecimento implícito de suas área de atuação. Os engenheiros de requisitos, que não têm experiência no domínio de aplicação do cliente, necessitam compreender esses requisitos.

- Diferentes *stakeholders* têm em mente diferentes requisitos e podem expressá-los de maneira distintas. Cada *stakeholder* tem uma visão particular do sistema e como ele deva trabalhar. Algumas vezes as visões de diferentes *stakeholders* podem gerar

conflitos. Os engenheiros de requisitos precisam descobrir as possíveis fontes de requisitos, encontrar pontos comuns e possíveis conflitos.

- Fatores políticos podem influenciar os requisitos do sistema. Eles podem provir de gerentes que definem requisitos específicos de sistema para aumentar sua influência na organização.
- O ambiente econômico e de negócios, no qual a análise de requisitos ocorre, é dinâmico. Inevitavelmente, ele se modifica durante o processo de análise. Como conseqüência, a importância dos requisitos específicos pode mudar. Novos requisitos podem surgir por parte dos novos *stakeholders*, que não haviam sido consultados inicialmente.

O uso da linguagem natural pode trazer algumas dificuldades para a documentação dos requisitos (SOMMERVILLE, 2003):

- *Ambigüidade*: às vezes, é difícil utilizar a linguagem de maneira precisa e sem ambigüidades, sem produzir um documento de difícil leitura.
- *Falta de clareza*: os requisitos funcionais e os não funcionam, os objetivos do sistema e as informações sobre o projeto podem não estar claramente definidos.
- *Fusão de requisitos*: vários requisitos diferentes podem ser expressos juntos como um único requisito.

Outros problemas com a utilização da linguagem natural podem surgir, quando é utilizada para uma especificação mais detalhada:

- A compreensão da linguagem natural por leitores e por quem escreve as especificações depende muitas vezes do uso das mesmas palavras para vários conceitos. Isso pode conduzir a divergências, devido à ambigüidade da linguagem natural.
- Uma especificação de requisitos em linguagem natural é muito flexível. Pode-se dizer a mesma coisa de modos completamente diferente. Fica por conta de o leitor descobrir quando os requisitos são os mesmos e quando são diferentes.
- Não existe um meio de fácil padronização para requisitos de linguagem natural. Pode ser difícil encontrar todos os requisitos relacionados. Para descobrir as conseqüências de uma mudança, talvez seja preciso examinar cada requisito, em vez de simplesmente um grupo de requisitos relacionados.

Sommerville (2003) descreve a limitação dos usuários em imaginar o sistema em operação e como o sistema se adequaria ao trabalho do usuário. Não é fácil para profissionais habilitados de computação conseguirem realizar este tipo de análise abstrata. Isto é ainda mais difícil para os usuários de sistema. Como resultado, a validação de requisitos tem dificuldades de descobre todos os problemas com os requisitos, e as modificações para corrigir omissões e falhas de compreensão, depois de o documento de requisitos ter sido aceito.

2.3 Classificação de Requisitos

A classificação dos requisitos auxilia na identificação e na descrição dos requisitos. Os requisitos de sistema de software são, freqüentemente, classificados como funcionais ou não funcionais ou como requisitos de domínio:

- *Requisitos Funcionais*: "são declarações de funções que o sistema deve fornecer como o sistema deve reagir a entradas específicas e como se deve comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer" (SOMERVILLE, 2003). "Um requisito funcional descreve uma interação entre o sistema e o ambiente. Os requisitos funcionais descrevem como o sistema deve-se comportar a partir de certo estímulo" (PFLEEGER, 2001).
- *Requisitos não funcionais*: "São restrições sobre os serviços ou as funções oferecidas pelo sistema. Entre eles destacam-se restrições de tempo, restrições sobre o processo de desenvolvimento, padrões, dentre outros" (SOMMERVILLE, 2003).
- *Requisitos de domínio*: "São requisitos que se originam do domínio de aplicação do sistema e que refletem características desse domínio. Podem ser requisitos funcionais ou não funcionais" (SOMMERVILLE, 2003).

Uma outra classificação importante a se fazer é quanto à importância de cada requisito. Pfleeger (2001) sugere classificá-los em três categorias:

- requisitos imprescindíveis;
- requisitos desejáveis, mas não necessários;
- requisitos possíveis, mas que podem ser descartados.

A partir de uma perspectiva de evolução, Sommerville (2003) classifica os requisitos como:

- *Requisitos permanentes*: requisitos relativamente estáveis, que derivam da atividade principal da organização e que se relacionam diretamente com o domínio do sistema.
- *Requisitos voláteis*: requisitos que provavelmente vão-se modificar durante o desenvolvimento do sistema ou depois que o sistema estiver em operação.
- *Requisitos mutáveis*: requisitos que se modificam por causa das mudanças do ambiente no qual a organização está operando.
- *Requisitos emergentes*: requisitos que surgem à medida que a compreensão do cliente do sistema se desenvolve durante o desenvolvimento do sistema. O processo de projeto pode revelar novos requisitos emergentes.
- *Requisitos conseqüentes*: requisitos que resultam da utilização do sistema de computação. Esse sistema pode modificar os processos da organização e criar novos meios de trabalho, que podem gerar novos requisitos de sistema.
- *Requisitos de compatibilidade*: requisitos que dependem de sistemas ou processos de negócios específicos dentro da organização. À medida que estes se modificam, os requisitos de compatibilidade podem também evoluir.

2.3.1 Requisitos não funcionais

É importante dar uma atenção especial aos requisitos não funcionais. Segundo Pfleeger (2001) "um requisito não funcional descreve uma restrição no sistema que limita as escolhas na construção de uma solução para o problema".

Para Sommerville (2003) "os requisitos não funcionais são aqueles que não dizem respeito diretamente às funções específicas fornecidas pelo sistema. Eles podem estar relacionados a propriedades de sistema emergentes, como confiabilidade, tempo de resposta e espaço em disco. Como alternativa, podem definir restrições para o sistema, como a capacidade dos dispositivos de entrada / saída e as representações de dados utilizadas nas interfaces de sistema".

Muitos requisitos não funcionais dizem respeito ao sistema como um todo, e não a características individuais do sistema. Isso significa que eles são, freqüentemente, mais importantes do que os requisitos funcionais individuais. Enquanto a falha em cumprir um requisito funcional individual pode degradar o sistema, a falha em cumprir um requisito não funcional de sistema pode tornar o sistema inútil. Por exemplo, se um sistema de aviação não atender a seus requisitos de confiabilidade, ele não será atestado como seguro para operação.

Se um sistema de controle em tempo real deixar de cumprir seus requisitos de desempenho, as funções de controle não operarão corretamente.

Sommerville (2003) sugere uma taxonomia de requisitos não funcionais como descrito na Figura 2.3.

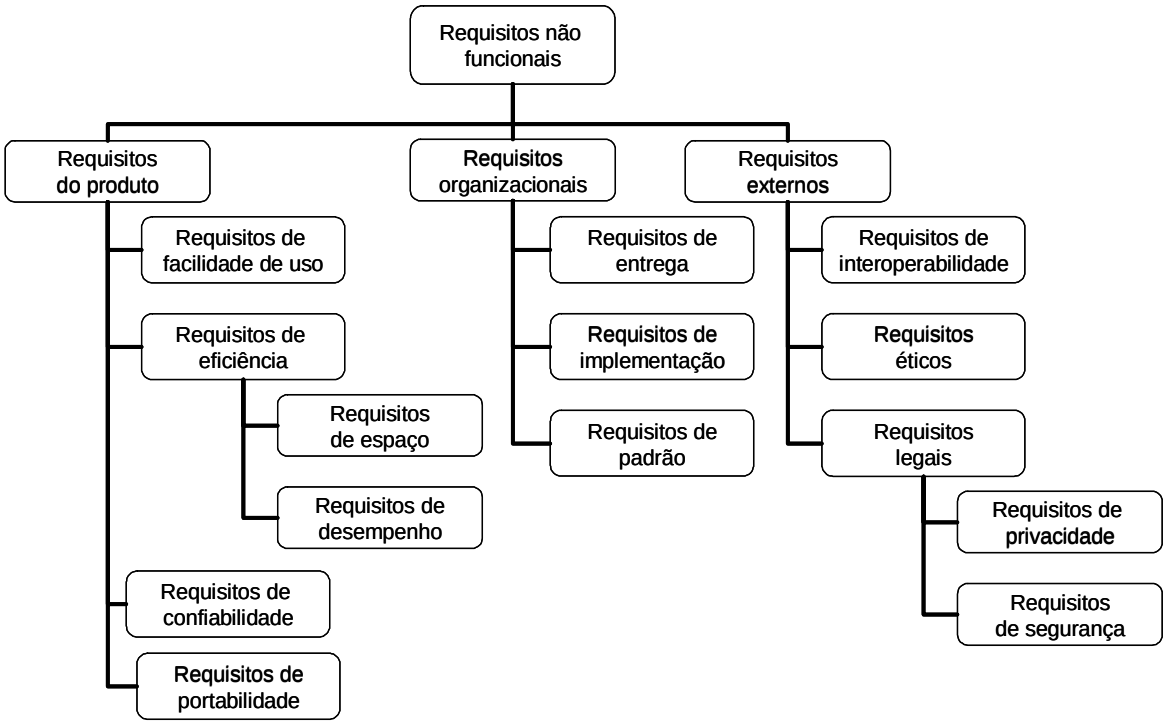

Figura 2.3: Classificação dos requisitos não funcionais (SOMMERVILLE, 2003)

Um problema comum com os requisitos não funcionais é que eles são, às vezes, de difícil verificação; eles podem ser escritos para refletir os objetivos gerais do cliente, como a facilidade de uso, a habilidade de o sistema se recuperar de uma falha ou a rapidez de resposta ao usuário. Esses requisitos causam problemas para os desenvolvedores de sistema, à medida que eles deixam o enfoque aberto à interpretação e à conseqüente discussão, quando o sistema é entregue. Como ideal, os requisitos não funcionais devem ser expressos quantitativamente, utilizando-se métrica que possam ser objetivamente testadas.

Os requisitos não funcionais freqüentemente entram em conflito e interagem com outros requisitos funcionais do sistema. Por exemplo, pode ser requerido que o máximo de espaço ocupado por um sistema seja de 4 Mbytes, porque todo o sistema de ser adequado à memória do tipo *read-only* (memória "somente leitura") e instalado em uma espaçonave. Um requisito adicional pode ser que o sistema seja escrito utilizando-se Ada, uma linguagem de programação projetada para o desenvolvimento de software crítico e em tempo real. Contudo, pode não ser possível compilar um programa Ada com a funcionalidade exigida, com menos

de 4 Mbytes. É preciso encontrar certa "compensação" entre esses requisitos. Uma linguagem alternativa de desenvolvimento pode ser utilizada, ou também é possível acrescentar memória ao sistema.

Em princípio, os requisitos funcionais e não funcionais devem ser diferenciados em um documento de requisitos; na prática, isso é muito difícil. Se os requisitos não funcionais forem definidos separadamente dos requisitos funcionais, algumas vezes será difícil ver o relacionamento entre eles. Se eles forem definidos com os requisitos funcionais, poderá ser difícil separar considerações funcionais e não funcionais e identificar os requisitos que correspondem ao sistema como um todo. É preciso encontrar um equilíbrio adequado e isso depende do tipo de sistema que está sendo especificado.

2.4 Técnicas de Especificação de Requisitos

Existem diversas técnicas de especificação de requisitos, cabendo ao analista de requisitos escolher a técnica que melhor tenha adequação ao sistema que está sendo especificado. Por exemplo: se um sistema tiver requisitos de tempo real, devem-se procurar técnicas que permitam incluir regras de tempo em sua especificação. Pfleeger (2001) afirma que "nenhuma técnica de especificação de requisitos é completa; o que pode ser adequado para a equipe de projeto tratar pode ser difícil para a equipe de teste usar". Apesar existirem diversas técnicas a UML (*Unified Modeling Language*) tem-se tornado um padrão para a modelagem orientada a objetos. Assim, os casos de uso e a obtenção de requisitos com base em caso de uso são cada vez mais utilizados para a captura de requisitos (SOMMERVILLE, 2003).

Os casos de uso são técnicas baseadas em cenários para a obtenção de requisitos, que foram introduzidos pela primeira vez no método *Objectory* (Jacobson *et al*., 1993). Os casos de uso tornaram-se, atualmente, uma característica fundamental da notação UML para descrever modelos de sistemas orientados a objetos. Em sua forma mais simples, um caso de uso identifica os agentes envolvidos em uma interação e especifica o tipo de interação.

A Figura 2.4 e a Figura 2.5 ilustram os pontos essenciais da notação de caso de uso. Os agentes (atores) no processo são representados por bonecos e cada classe de interação é representada por uma elipse com um nome. O conjunto de casos de uso representa todas as possíveis interações que serão representadas nos requisitos de sistemas.

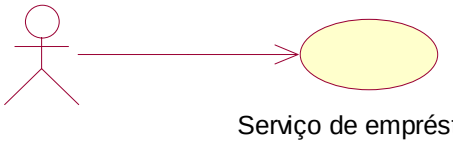

Figura 2.4: Caso de uso de empréstimo (SOMMERVILLE, 2003)

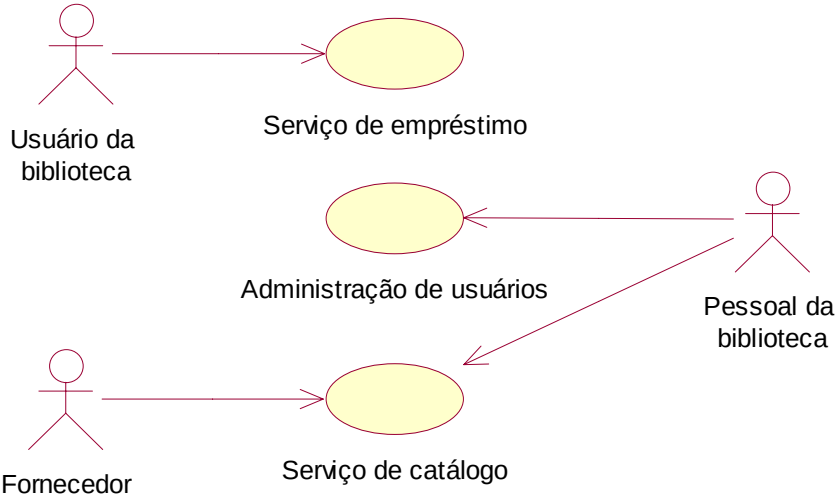

Figura 2.5: Diagrama de casos de uso de biblioteca (SOMMERVILLE, 2003)

Robertson e Robertson (1999) citam algumas vantagens de o sistema ser modelado através de casos de uso:

- Devido às poucas conexões entre um caso de uso e outro, pode-se examinar cada caso de uso separadamente e entendê-lo sem se precisar conhecer todos os detalhes do sistema. Em particular, um de usuário pode entender a funcionalidade pedida por ele sem que haja necessidade de entender as demais funcionalidades do sistema.
- Podem-se usar os casos de uso como base para uma estimativa de tempo e esforço necessários para desenvolver o sistema.
- O desenvolvimento de sistema pode ser orientado em termos de caso de uso. Assim sendo, o gerenciamento do desenvolvimento pode acompanhar o progresso de cada caso de uso nas etapas do processo de desenvolvimento. Pode-se usar um atributo que indique o estado do caso de uso nas etapas do processo de desenvolvimento, como por exemplo: “analisado”, “codificado”, “testado”, “implantado”, etc.

2.5 Validação de requisitos

Validação de requisitos é o processo de determinar que a especificação seja consistente com a definição de requisitos; isto é, a validação garante que os requisitos satisfaçam as necessidades dos usuários. (SOMMERVILLE, 2003).

A validação de requisitos é importante porque a ocorrência de erros em um documento de requisitos pode levar a grandes custos relacionados ao retrabalho, quando esses erros são descobertos durante o desenvolvimento ou depois que o sistema estiver em operação. O custo de fazer uma modificação no sistema, resultante de um problema de requisitos, é muito maior do que reparar erros de projeto ou de codificação. (SOMMERVILLE, 2003)

Pfleeger (2001) define que a validação deve garantir a rastreabilidade entre os requisitos. É importante lembrar que a validação é mais que uma simples checagem de rastreabilidade. A validação deve garantir que o sistema irá atender às expectativas de clientes e usuários. Deve-se confirmar que objetivos e intenções dos clientes e usuários sejam satisfeitos.

Quando qualquer problema for identificado, a equipe de revisão dos requisitos determinará sua causa e as ações a serem tomadas para corrigir o problema, antes que se comecem as fases de construção do sistema. O processo de revisão estará completo, quando clientes e desenvolvedores estiverem-se sentindo confortáveis com as especificações de requisitos.

Durante a revisão, são realizadas as seguintes atividades:

- Será revisado o estado dos objetivos e propósitos do sistema.
- Serão comparados os requisitos com os objetivos e propósitos do sistema para verificar que todos os requisitos são necessários.
- Será apresentado o ambiente onde o sistema irá operar. Serão examinadas as interfaces entre o sistema com os sistemas externos, serão verificados se as interfaces estão corretas e completas.
- Se algum risco está envolvido no desenvolvimento ou em alguma funcionalidade do sistema, o risco será avaliado e documentado. Serão apresentadas, discutidas e comparadas possíveis soluções a serem adotadas. As decisões de que soluções serão utilizadas serão tomadas em conjunto com o cliente.
- Serão discutidos os testes do sistema:

- *Que requisitos continuarão sendo verificados e validados no desenvolvimento do sistema, quando houver uma mudança de requisito*?
- *O que a equipe de teste deverá verificar para garantir que cada requisito foi implementado adequadamente*?
- *Quem irá fornecer os dados de teste*?
- *Como os requisitos serão testados durante a fase de implementação*?

2.5.1 Tipos de verificação

Sommerville (2003) e Pfleeger (2001) destacam alguns tipos de verificação que devem ser realizados sobre os requisitos:

- *Verificação de corretitude*: os requisitos não devem conter erros.
- *Verificação de validade*: será verificada a validade das funções propostas pelos usuários e por funções adicionais encontradas em uma análise mais detalhada dos requisitos. Os sistemas têm diversos usuários com necessidades diferentes e qualquer conjunto de requisitos é inevitavelmente uma solução conciliatória da comunidade de usuários. Também se deve atentar para requisitos que restringem o desenvolvimento desnecessariamente ou incluem funcionalidades, que não estejam diretamente relacionadas com o problema. Por exemplo, um general pode decidir que um novo software de tanque deve permitir aos soldados enviar e receber e-mails, ainda que o principal propósito do tanque seja a travessia de terrenos irregulares. Devem-se manter somente os requisitos que tratam diretamente com a solução do problema.
- *Verificação de consistência*: os requisitos em um documento não devem ser conflitantes nem ambíguos. Não devem existir restrições contraditórias ou descrições diferentes para uma mesma função do sistema. Em outras palavras, dois requisitos são considerados inconsistentes se for impossível satisfazer aos dois simultaneamente.
- *verificação de completitude*: o documento de requisitos deve incluir requisitos que definam todas as funções e restrições exigidas pelo usuário do sistema. Todos os estados possíveis, mudança de estado, entradas, produtos e restrições devem ser descritos para um requisito. Para Pfleeger (2001), existem dois tipos de completitude para um sistema:
  - *completitude externa*: a descrição de requisitos deve conter todos os elos próprios do ambiente estabelecido.

- *completitude interna*: a descrição de requisitos não deve conter indefinições entre os requisitos referenciados.

- *verificação de realismo*: utilizando o conhecimento da tecnologia existente, os requisitos devem ser verificados, a fim de assegurar que eles realmente podem ser implementados. Essas verificações devem também levar em conta o orçamento e os prazos para o desenvolvimento do sistema.

- *facilidade de verificação*: para reduzir o potencial de divergências entre cliente e fornecedor, os requisitos do sistema devem sempre ser escritos de modo que possam ser verificados. Isso significa que um conjunto de verificações pode ser projetado para mostrar que o sistema cumpre com esses requisitos.

- *verificação de rastreabilidade*: todas as funcionalidades do sistema devem ser rastreadas para os requisitos que as originaram. Deve ser fácil achar o conjunto de requisitos que trata com um aspecto específico do sistema.

2.5.2 Técnicas de Validação

Sommerville (2003) destaca algumas técnicas de validação que podem ser utilizadas em conjunto ou individualmente:

- *Revisão de requisitos*: os requisitos são analisados sistematicamente por uma equipe de revisores.
- *Prototipação*: um modelo executável do sistema é mostrado aos usuários finais e clientes, que podem experimentar o modelo para verificar se atende às suas necessidades.
- *Geração de casos de testes*: criação de casos de testes para validação dos requisitos. Se um caso de teste é difícil ou impossível de ser projetado, isto pode significar que os requisitos serão de difícil implementação e devem ser revistos.
- *Análise automatizada da consistência*: se requisitos são expressos em uma notação estruturada ou formal, então ferramentas CASE podem ser utilizadas para verificar sua consistência (Figura 2.6). Uma análise desses requisitos poderá gerar um relatório das inconsistências descobertas.

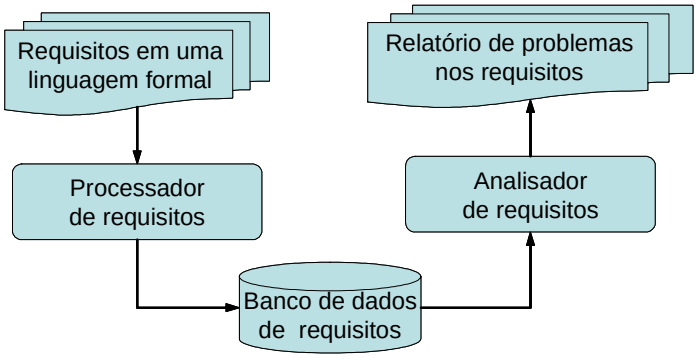

Figura 2.6: Verificação da consistência de requisitos (SOMMERVILLE, 2003).

As dificuldades da validação de requisitos não devem ser subestimadas. É difícil demonstrar que um conjunto de requisitos atende às necessidades de um usuário. A validação de requisitos raramente revela todos os problemas com os requisitos. É inevitável a correção de omissões e falhas de compreensão na documentação de requisitos depois que ela é aceita.

2.6 Gerenciamento de Requisitos

A gerência de requisitos é uma atividade da engenharia de requisitos, que se dedica a gerenciar as modificações nos requisitos (SOMMERVILLE 2003). Praticamente em todos os sistemas, os requisitos se modificam. Uma das razões para isso é que esses sistemas são geralmente desenvolvidos para lidar com problemas "intrincados". Como o problema não pode ser inteiramente definido, os requisitos de sistema são necessariamente incompletos. Durante o processo de software, a compreensão dos desenvolvedores sobre o problema está constantemente se modificando, e essas mudanças se refletem nos requisitos.

Depois de os usuários finais se familiarizarem com o sistema, novos requisitos surgem, pelas seguintes razões:

- Grandes sistemas geralmente têm uma comunidade de usuários diversificada. Diferentes usuários têm diferentes requisitos e propriedades, que podem ser conflitantes ou contraditórios. Os requisitos finais do sistema são inevitavelmente uma conciliação entre eles; e, com a experiência, muitas vezes é constatado que o equilíbrio do apoio dado a diferentes usuários precisa ser mudado.
- Os clientes e os usuários finais de um sistema em geral não são as mesmas pessoas. Os clientes do sistema impõem requisitos em razão de restrições organizacionais e orçamentárias, e esses requisitos podem ser conflitantes com os requisitos dos usuários finais.

- A empresa e o ambiente técnico do sistema modificam-se, e isto deve ser refletido no próprio sistema. Um novo hardware pode ser implementado, pode ser necessário fazer a interface do sistema com outros sistemas, as prioridades da empresa podem-se modificar, acarretando conseqüentes mudanças no suporte necessário ao sistema, e novas legislações e regulamentos, que afetam o sistema, podem ser criados. Os requisitos não funcionais são, particularmente, afetados por mudanças na tecnologia de hardware.
- Os *stakeholders* passam a compreender melhor o que querem que o software faça.

2.6.1 A Área de Processo de Gerenciamento de Requisitos

A área de processo do CMMI (2006), nível de maturidade 2, *Gerenciamento de Requisitos*, objetiva gerenciar requisitos dos produtos e componentes de produtos do projeto, identificando inconsistências entre esses requisitos e planos e artefatos do projeto. Também documenta as mudanças nos requisitos e suas justificativas, para manter a rastreabilidade bidirecional entre os requisitos fonte e todos os requisitos de produtos e componentes de produtos.

Esta área de processo possui apenas a meta específica Gerenciar Requisitos, que é composta por cinco práticas específicas (CMMI, 2006):

- *Obter um entendimento dos requisitos*: desenvolve um entendimento com fornecedores dos requisitos sobre o significado dos requisitos.
- *Obter compromisso sobre os requisitos*: obtém compromissos com os requisitos dos participantes do projeto.
- *Gerenciar mudanças nos requisitos*: gerencia as mudanças nos requisitos, durante sua evolução no projeto.
- *Manter a rastreabilidade bidirecional dos requisitos*: mantém a rastreabilidade bidirecional entre requisitos e planos e artefatos do projeto.
- *Identificar inconsistências entre o trabalho do projeto e os requisitos*: identifica inconsistências entre requisitos e planos e artefatos de projeto.

No CMMI (2006), há ainda a área de processo *Desenvolvimento de Requisitos* do nível 3, que objetiva produzir e analisar requisitos de clientes, produtos e componentes de produtos. Esses requisitos tratam das necessidades dos *stakeholders* relevantes, incluindo as pertinentes às diversas fases do ciclo de vida do produto e atributos do produto, como também tratam das restrições causadas pela seleção de soluções de *design*.

Ao se deparar com as mudanças no produto de software é necessário avaliar o impacto dessas mudanças no sistema, isto é, que requisitos estão direta ou indiretamente relacionados com elas. Para isto, é necessário que os requisitos do sistema estejam rastreados.

2.7 Rastreabilidade de Requisitos

A rastreabilidade é essencial para a engenharia de requisitos e é necessária quando se procura informações no sistema. Para sistemas de porte médio ou complexo, deve ter-se um modelo de rastreabilidade que poderá ser de grande utilidade. É comum existir vários requisitos vindo de uma mesma fonte e vice-versa. É também comum que um requisito se derive de vários outros requisitos ou que vários requisitos tenham convergência para um só requisito.

O conceito de rastreabilidade de requisitos é muito simples: seguir relações ou ligações. A rastreabilidade é essencial no desenvolvimento de sistema, pois muitas informações são usadas e produzidas e elas devem permanecer relacionadas (PINHEIRO, 2003).

Segundo o IEEE Std 830 (1984), "uma especificação de software é rastreável se: (*i*) a origem de cada requisito é clara, e (*ii*) se ele facilita o referenciamento de cada requisito no desenvolvimento futuro ou na otimização da documentação".

Segundo Gotel e Finkelstein (1994), "o rastreamento de requisito refere-se à habilidade de descrever e seguir a vida de um requisito, em ambas as direções: para frente ou pra trás. Ou seja, desde a origem passando por etapas de refinamento e desenvolvimento do requisito até sua aplicação".

2.7.1 Classificação de Rastreabilidade

A rastreabilidade pode ser classificada de várias maneiras, conforme a Figura 2.7 (PINHEIRO, 2003):

- *Rastreabilidade na ordem direta*: é a habilidade de ligar um requisito a componentes de análise e implementação.
- *Rastreabilidade na ordem inversa*: é a habilidade de ligar um requisito a sua fonte, ou seja, a uma pessoa, instituição, lei, argumento, etc.
- *Rastreabilidade de pré-requisitos*: refere-se a todos os aspectos da vida do requisito antes de sua inclusão na especificação de requisitos.
- *Rastreabilidade pós-requisitos*: refere-se a todos os aspectos da vida do requisito após sua inclusão na especificação de requisitos. É útil para se saber os modelos de

análise que foram utilizados naquele requisito ou os procedimentos de teste criados para verificá-lo.

- *Rastreabilidade interna aos requisitos*: refere-se ao relacionamento entre requisitos. No desenvolvimento, um requisito é refeito e refinado várias vezes; muitos são gerados e muitos são abandonados nesse processo.
- *Rastreabilidade externa aos requisitos*: refere-se ao relacionamento dos requisitos com outros artefatos.
- *Rastreabilidade funcional*: está relacionada com aspectos funcionais do desenvolvimento de software, que devem ser descritos em termos de transformações sobre os estados definidos do desenvolvimento de software, seus modelos e seus artefatos.
- *Rastreabilidade não funcional*: está freqüentemente relacionada com intenção, propósito, objetivos, responsabilidades, aspectos de qualidade e resultado de relacionamento de conceitos intangíveis. Não é responsável por referencias diretas, isto é, a maioria dos requisitos não funcionais necessita ser transformado em seus requisitos funcionais correspondentes, para que possam ser verificados.

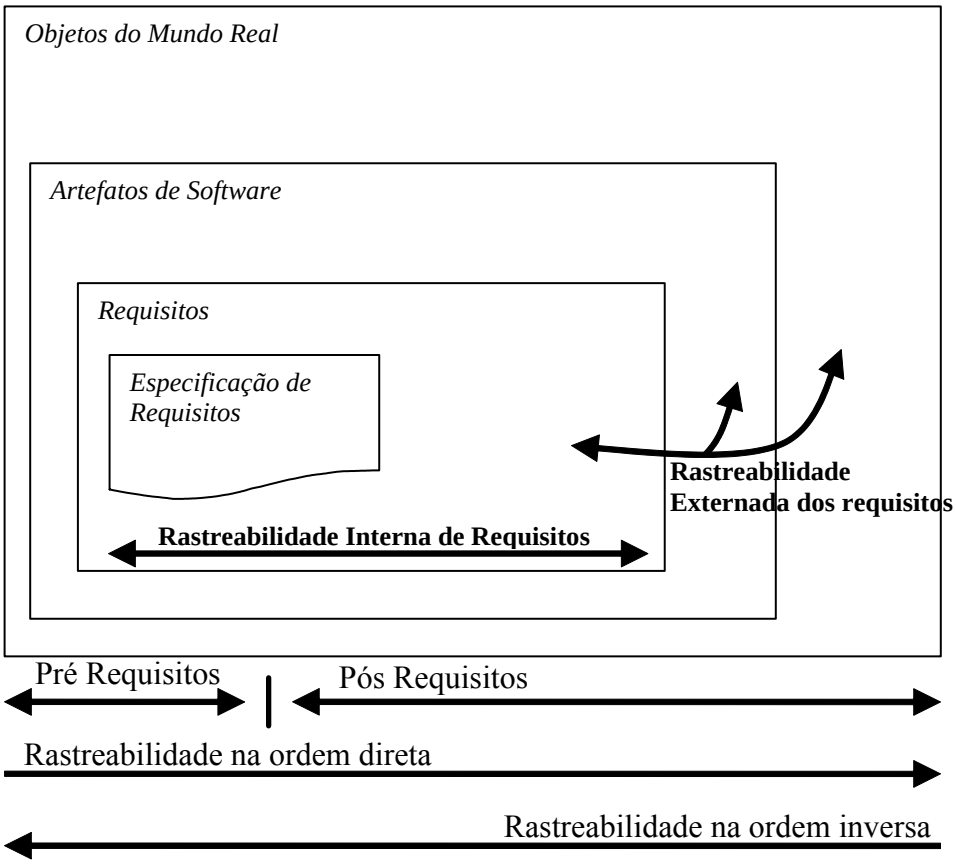

Figura 2.7: Rastreabilidade (PINHEIRO, 2003).

2.7.2 Uso da Rastreabilidade

Rastreabilidade é uma atividade auxiliar para atingir alguns objetivos e para reduzir a burocracia para se procurar uma informação. A rastreabilidade auxilia tanto a pessoas técnicas quanto a gerência. Em ambos os casos é uma atividade muito dinâmica, guiada pela necessidade e não por uma estrutura pré-definida.

Um exemplo de objetivos técnicos é o rastreamento para investigar as causas de um erro de teste. Um exemplo de objetos de gerenciamento é a rastreabilidade para avaliar os impactos de uma mudança.

Ramesh (1998) identifica dois tipos usuários da rastreabilidade: os usuários de baixo nível e os usuários de alto nível. Os usuários de baixo nível estão envolvidos com questões técnicas. Assim, suas necessidades são para resolver problemas técnicos e tomar decisões técnicas. Um exemplo seria saber se todos os requisitos derivados de um dado requisito estão de acordo. Os usuários de alto nível estão mais interessados em questões gerenciais. Por exemplo, saber que informação deve ser capturada, por quem, e como deverá ser usada.

A maioria das necessidades dos usuários de alto nível é de natureza não-funcional e são as mais difíceis de serem automatizadas. Diferentemente, as necessidades dos usuários de baixo nível são comumente satisfeitas com rastreabilidade funcional.

2.7.3 Necessidades de Rastreabilidade

Quando se trata de grande quantidade de informações de sistemas complexos, é essencial ter-se alguma forma de ajuda automatizada para a rastreabilidade. Um ambiente de rastreabilidade, composto de procedimentos, métodos, técnicas e ferramentas, deve fazer parte para auxiliar o processo de rastreabilidade. Usualmente, esse ambiente é construído sob um modelo que dê uma visão uniforme para a rastreabilidade e para os objetos rastreados.

O uso de modelos formais de rastreabilidade auxilia a automação do processo. Não somente a automação do modelo, mas também a generalização de rastreabilidade e a definição de procedimento para verificar consistência e corretitude das rastreabilidade. O uso de formalidade aumenta a eficiência da rastreabilidade.

No entanto, a informação é, por si só, em muitos casos, vista como desestruturada. Por esta razão, o que deve ser rastreado é, em muitos casos, inerentemente informal. Por exemplo, afirmações em linguagem natural, transcrição de entrevistas, e imagens.

Os aspectos informais e não estruturados da informação são próprios de rastreabilidade não-funcional. Devem-se usar modelos para capturar alguns dos aspectos não-funcionais do desenvolvimento de software. O uso de modelos é importante, mas serve apenas como um ponto inicial para guiar o processo de rastreabilidade não-funcional.

2.8 Métrica de Requisitos

As métricas de requisitos fornecem informações sobre o processo de requisitos e sobre a qualidade dos requisitos. Segundo Fenton e Pfleeger (1997), as métricas comumente focam-se em três áreas:

- produto da fase de requisitos (documentação de requisitos);
- processo;
- código fonte.

Em as várias métricas de requisitos pode-se citar:

- *O número de mudanças de requisitos*: um grande número de mudanças indica alguma instabilidade ou incerteza no entendimento do que o sistema deverá fazer ou de seu comportamento. Devem-se tomar medidas para que o número de mudanças de requisitos seja tão pequeno quanto possível na prática.
- *O tamanho dos requisitos*: permite apreender se uma mudança ou incerteza em um requisito é uma característica do software, ou de certos tipos de requisitos, como interface com usuário ou requisitos de banco de dados. Estas informações permitem tomar ações para aumentar o entendimento e reduzir as incertezas em alguns tipos de requisitos em particular.
- *A evolução do requisito*: pode-se tomar nota, para cada requisitos, quando ele for revisado, projetado, codificado e testado. Esta métrica indica o progresso para a entrega do requisito.

2.9 Conclusão

Este capítulo abordou vários aspectos da engenharia de requisitos. O capítulo seguinte enfocará sobre ontologias, que servirá de base para a proposta de elaboração de ontologias para engenharia de requisitos.

Capítulo 3

ENFOQUE SOBRE ONTOLOGIAS

Este capítulo fornece uma visão geral sobre ontologias, que servirá para fundamentar a proposta de construção de uma ontologia para engenharia de requisitos de software.

3.1 Definições

O vocábulo ontologia vem do grego *ontos* (ser) + *logos* (palavra). Foi introduzida na filosofia no século XIX, por filósofos alemães, para distinguir o estudo do ser do estudo dos vários tipos de seres na ciência natural (BRAITMAN, 2003). Antigamente, o termo ontologia estava associado à esfera da filosofia. Como uma disciplina da filosofia, pode-se referir a uma ontologia como um sistema de categorias modelando certa visão do mundo (GUARINO, 98).

Atualmente, este termo está sendo utilizado em várias áreas da ciência da computação: inteligência artificial, lingüística computacional e em teorias de banco de dados. Pode-se dizer ainda, que sua importância é reconhecida em vários "campos" de pesquisa como engenharia do conhecimento, representação do conhecimento, modelagem qualitativa, processamento de linguagem natural, integração de informação, análise orientada a objetos, recuperação e extração de informação, gerência e organização do conhecimento (GUARINO, 98).

Na concepção de Gruber (1993), "uma ontologia é uma especificação formal e explícita de uma conceitualização compartilhada". Nesta definição, entende-se por "conceitualização" um modelo abstrato. A palavra "explícita" indica que os elementos da ontologia são claramente definidos e, finalmente, a palavra "formal" significa que a ontologia deve ser processável por uma máquina (FENSEL, 2001).

Para Gruninger (1995), uma ontologia é uma especificação de uma conceitualização: uma descrição de conceitos e relações que existem em um domínio de interesse. Basicamente, uma ontologia consiste desses conceitos e relações, e suas definições, propriedades e restrições que estão descritas na forma de axiomas. No entanto, não há uma definição universal para ontologias. Uma razão dessas inúmeras definições é a grande quantidade de formas de suas aplicações.

Uma ontologia é essencialmente um acordo, e esse acordo não necessariamente precisa abranger toda a conceitualização de determinado domínio. O acordo pode abranger apenas parte dele, ou seja, a ontologia pode oferecer uma visão para o domínio. Assim, uma ontologia atua

como um contrato entre parceiros, permitindo que se comunique com segurança dentro do domínio de informação (DAUM e MERTEN, 2002).

3.2 Utilizações de Ontologias em Computação

Na ciência da computação, ontologias têm sido desenvolvidas para facilitar o compartilhamento de conhecimento e seu reuso. Atualmente, têm-se expandido em outras áreas como a integração inteligente de informação, sistemas cooperativos, produtos de software baseados em agentes e no comércio eletrônico (BREITMAN, 2003). Gruninger (2002) relata, entre outras questões, a possibilidade do uso de ontologias para comunicação entre homens e sistemas computacionais, para inferência computacional e para o reuso e a organização de conhecimento.

Noy e McGuinness (2001) listam as finalidades abaixo para o uso de ontologia em computação:

- Compartilhar um entendimento comum de uma estrutura de informação entre pessoas e agentes de software. Esta é uma das finalidades mais freqüentes para se desenvolver uma ontologia.
- Possibilitar o reuso do conhecimento de um domínio. Uma vez construída uma ontologia representando um domínio, esta poderia ser reutilizada ou estendida quando conveniente.
- Tornar explícitas as premissas utilizadas. Freqüentemente, boa parte do conhecimento a cerca do domínio da aplicação está embutido dentro do código dos sistemas. Esta codificação torna muito difícil identificar as premissas utilizadas quando é necessária alguma manutenção. Explicitar premissas por meio de uma ontologia facilita sua manutenção quando o conhecimento acerca do domínio mudar. Também, especificar explicitamente o conhecimento do domínio é útil para novos usuários, os quais necessitam aprender seus termos e significados.
- Separar o conhecimento do domínio do conhecimento da operacionalização. O conhecimento operacional, em geral, é mais facilmente representado através de algoritmos, já o conhecimento do domínio pode ser mais apropriadamente representado numa ontologia.
- Analisar o domínio do conhecimento é possível uma vez que a especificação declarativa dos termos esteja disponível. A análise formal dos termos é de grande valia para reuso das ontologias existentes e para as estender.

3.3 Metodologias para Construção de Ontologias

Atualmente, existem grupos de pesquisa que estão propondo um conjunto de passos e metodologias para a construção de ontologias. Porém, a padronização de um processo para construção de ontologias é ainda incipiente. O que acontece em muitos dos casos é a adoção de uma metodologia própria desenvolvida em função do domínio e da aplicação desejada. (GAVA e MENEZES, 2003).

Noy e McGuinness (2001) apresentam três pressupostos no projeto de ontologias:

- Não existe uma maneira correta de modelar um domínio – sempre existem alternativas viáveis. A melhor solução quase sempre depende da aplicação.
- O desenvolvimento de uma ontologia é necessariamente um processo iterativo.
- Conceitos em uma ontologia deveriam ser próximos dos objetos (físicos ou lógicos) e relacionamentos do domínio de interesse. Os conceitos, então, deveriam ser provavelmente substantivos (objetos) ou verbos (relacionamentos) em sentenças que descrevem o domínio.

A seguir, apresentamos algumas metodologias utilizadas para a modelagem de ontologias. Inicialmente, apresentamos a metodologia TOVE, proposta por Gruninger e Fox (1995), que utiliza lógica de primeira ordem com o objetivo de construir modelos com a capacidade de responder a questões simples acerca de domínios no contexto das empresas. A segunda metodologia é a proposta por Noy e McGuiness (2001), que é mais flexível e coloca o conceito de iteratividade na construção da ontologia. Por último, comentamos a metodologia proposta por Falbo (1998), que apresenta a utilização de modelos gráficos para facilitar a comunicação com especialistas do domínio, utilizando também um modelo iterativo.

3.3.1 TOVE

A metodologia TOVE (*TOronto Virtual Enterprise*) foi proposta por Gruninger e Fox (1995) com base em suas experiências do *Entreprise Integration Laboratory* da Universidade de Toronto. Essa metodologia é composta de seis fases, conforme a Figura 3.1.

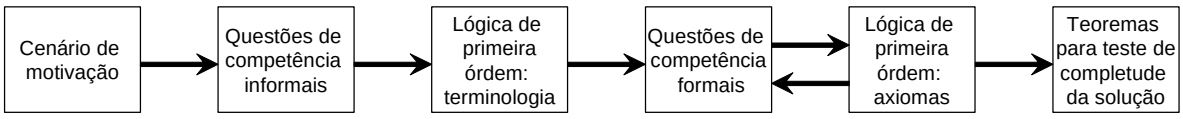

Figura 3.1: Processo de Modelagem e Evolução da Ontologia pela TOVE

a) Cenário de Motivação

O desenvolvimento de ontologias é motivado por cenários originados na aplicação. Os cenários de motivação freqüentemente têm a forma de história (*story*) de um problema, que não são respondidas nas ontologias existentes, fornecendo um conjunto intuitivo de possíveis soluções para o problema. Essas soluções dão uma idéia inicial da *semântica informal pretendida* de classes e relações que depois serão incluídas na ontologia.

b) Questões de Competência Informais

Dado um cenário de motivação, um conjunto de perguntas irá originar-se de lugares ainda não explorados na ontologia. Estas questões são consideradas como requisitos que as questões formais devem ser capazes de responder. Estas são consideradas questões de competência informal até que sejam expressas em uma linguagem formal da ontologia. É aconselhado que essas questões sejam estratificadas em questões de alto nível e questões de baixo nível.

c) Lógica de Primeira Ordem: Terminologia

Uma vez propostas as questões informais de competência, a terminologia da ontologia deve ser expressa em lógica de primeira ordem. Uma ontologia é formada de classes, propriedades de classes e relações entre essas classes. Isto estabelece a linguagem que será usada para expressar as definições e as restrições em axiomas. Essa linguagem deve fornecer a terminologia necessária para declarar as questões informais de competência.

O primeiro passo na especificação da terminologia é a identificação das classes no domínio de discussão. Elas serão representadas por constantes e variáveis na linguagem. Os atributos das classes serão definidos como predicados unários e as relações entre as classes como predicados *n*-ários.

d) Questões de Competência Formais

Após as questões de competência terem sido propostas informalmente e a terminologia da ontologia esteja definida, as questões de competência são definidas formalmente. As respostas a tais questões devem poder ser derivadas formalmente como conseqüências lógicas dos axiomas. Assim, se $T_{\text{Ontologia}}$ é o conjunto de axiomas propostos na ontologia, $T_{\text{Instância}}$ é o conjunto de

instâncias, e Q é uma sentença de primeira ordem usando somente predicados na linguagem $T_{\text{Ontologia}}$. Então, $T_{\text{Ontologia}} \cup T_{\text{Instância}} \models Q$.

Temos também que: $T_{\text{Ontologia}} \cup T_{\text{Instância}} \not\models \neg Q$, ou seja, Q é consistente com $T_{\text{Ontologia}} \cup T_{\text{Instância}}$.

É importante perceber que, neste estágio, ainda não se tem nenhum axioma em $T_{\text{Ontologia}}$; contudo, as questões de competência colocadas restringem que axiomas devem ser incluídos. É importante salientar que todos os termos presentes nas sentenças das questões formais de competência devem estar devidamente incluídos na terminologia da ontologia.

e) Lógica de Primeira Ordem: Axiomas

Os axiomas na ontologia, expressos em lógica de primeira ordem, especificam definições de termos da ontologia e restrições nas interpretações desses termos. A simples identificação de um conjunto de termos em lógica de primeira ordem, não constitui uma ontologia. Os axiomas devem fornecer a definição semântica e de significado desses termos.

O processo de definição dos axiomas é talvez o aspecto mais difícil na definição da ontologia. Contudo, este processo é guiado pelas questões de competência. Os axiomas na ontologia devem ser necessários e suficientes para expressar as questões de competência e caracterizar suas soluções.

f) Teoremas para teste de completude da solução

Uma vez que as questões de competência e os axiomas foram formalizados, devem-se definir um conjunto Φ de condições a serem testadas que garantam a completude das soluções oferecidas para as questões de competência. Esse conjunto de condições deve ser especificado sob a forma de teoremas. Ao conseguir provar esses teoremas a partir da Ontologia, estamos validando as condições consideradas necessárias para garantir a completude da solução empregada. Assim, se $T_{\text{Ontologia}}$ é o conjunto de axioma da ontologia, $T_{\text{Instância}}$ é o conjunto de instâncias, Q é uma sentença em lógica de primeira ordem que especifica perguntas nas questões de competência, e ainda Φ é o conjunto de sentenças em lógica de primeira ordem que define o conjunto de condições a serem atendidas pelas soluções apresentadas. Então, temos que os teoremas do conjunto Φ têm uma das seguintes formas:

- $T_{\text{Ontologia}} \cup T_{\text{Instância}} \models \Phi$, se e somente se $T_{\text{Ontologia}} \cup T_{\text{Instância}} \models Q$.
- $T_{\text{Ontologia}} \cup T_{\text{Instância}} \models \Phi$, se e somente se $T_{\text{Ontologia}} \cup T_{\text{Instância}} \cup Q$ é consistente.
- $T_{\text{Ontologia}} \cup T_{\text{Instância}} \cup \Phi \models Q$ ou $T_{\text{Ontologia}} \cup T_{\text{Instância}} \cup \Phi \models \neg Q$.
- Todo modelo de $T_{\text{Ontologia}} \cup T_{\text{Instância}}$ está de acordo com a extensão de algum predicado P.

Qualquer extensão da ontologia deve ser capaz de atender aos teoremas utilizados para o teste da completude.

3.3.2 Metodologia de NOY e McGuiness

Noy e McGuiness (2001) propuseram um processo interativo para o desenvolvimento de ontologias, a partir de um guia elaborado com suas experiências através da ferramenta Protegé-2000. Inicialmente, a ontologia é desenvolvida de forma "grosseira", sendo revisada e refinada progressivamente em seus detalhes. Ao longo da definição da ontologia, são discutidas decisões de modelagem que o autor precisará tomar, onde são apresentados os prós e os contras e as implicações de possíveis soluções. Esse processo é composto por sete etapas a seguir.

a) Determinar o Domínio e o Escopo da Ontologia

Nesta fase, é sugerido o uso de questões de competência (seção 3.3.1). Também é sugerido responder as seguintes questões para ajudar a limitar o escopo:

- Qual o domínio que a ontologia deve cobrir?
- Para que alguém usará a ontologia?
- Para que tipos de questões as informações contidas na ontologia devem responder?
- Quem usará e manterá a ontologia?

b) Procurar e Reusar Ontologias Existentes

É de grande valor procurar trabalhos relacionados e avaliar se eles podem ser refinados e estendidos para o domínio de interesse da ontologia a ser construída. O reuso de ontologias existentes pode ser um requisito da sua ontologia, caso ela deva interagir com aplicações que já desenvolveram uma ontologia ou um vocabulário controlado. Muitas ontologias estão em formatos eletrônicos e podem ser importadas para um ambiente de desenvolvimento da ontologia.

c) Listar Termos Importantes da Ontologia

É muito útil listar todos os termos relativos a uma ontologia. No início, é importante conseguir uma lista abrangente desses termos e não se deve preocupar se os conceitos que eles representam estão sobrepostos ou se esses conceitos são classes, propriedades ou relações. Nas duas próximas etapas (definir classes e hierarquia, e definir as propriedades das classes), os conceitos serão relacionados. Algumas questões auxiliam na listagem desses termos:

- Sobre quais termos se quer falar?
- Que propriedade esses termos têm?
- O que se quer falar sobre esses termos?

d) Definir Classes e Hierarquia

Nesta etapa, as classes devem ser definidas e hierarquizadas entre si. Para esta tarefa sugere-se usar uma das seguintes abordagens:

- *Top-down*: o processo de desenvolvimento começa com a definição das classes de conceitos mais gerais no domínio. Depois, essas classes devem ser especializadas.
- *Bottom-up*: o processo de desenvolvimento começa com a definição das classes mais específicas. A hierarquia será construída com o agrupamento consecutivo dessas classes em conceitos mais genéricos.
- *Combinação*: o processo é uma junção das duas abordagens anteriores (*top-down* e *bottom-up*). Inicialmente, os conceitos mais evidentes são definidos, depois são feitas generalizações e especificações apropriadas.

Nenhuma destas abordagens é melhor que a outra. Depende muito da visão do domínio em particular. Por exemplo, se um desenvolvedor tem uma visão *top-down* sistemática do domínio, então, para ele, é mais fácil usar a abordagem *top-down*. A abordagem 'combinação' é geralmente a mais fácil para os desenvolvedores, desde que os conceitos 'intermediários' tendam a ser mais descritivos no domínio.

e) Definir as Propriedades das Classes

As classes por si só não proverão informação suficiente para responder as questões de competência da primeira etapa (determinar o domínio e o escopo da ontologia). Uma vez definidas as classes, deve-se descrever sua estrutura interna. Já foram selecionadas as classes entre

os termos listados na terceira etapa. A maioria dos termos que restaram são propriedades dessas classes. Em geral, existem alguns tipos de propriedades:

- *Intrínseca*: característica interna das classes. Por exemplo, o aroma de um vinho.
- *Extrínseca*: característica externa as classes. Por exemplo, o nome de um vinho.
- *Partes*: caso a classe seja estruturada, ela pode ter partes físicas e abstratas. Por exemplo, as rodas de um carro.
- *Relação*: relação com outras classes. Por exemplo, um carro possui um proprietário.

f) Definir as Restrições das Propriedades

Nesta etapa, definimos as restrições de cada propriedade, que podem ser: tipo de dados, valores permitidos, cardinalidade e outras restrições que a propriedade possa ter.

g) Criar Instâncias das Classes

Nesta última etapa, as instâncias das classes são criadas. A definição da instância envolve os seguintes passos:

- Escolher uma classe.
- Criar uma instância.
- Preencher suas propriedades.

3.3.3 Metodologia de Falbo

Falbo (1998) propôs uma metodologia para o desenvolvimento de ontologia, que vê a construção da ontologia como um processo iterativo e compõe-se das seguintes etapas, conforme Figura 3.2:

- Identificação de propósito e especificação de requisitos;
- Captura da ontologia;
- Formalização da ontologia;
- Integração com ontologia existente;
- Avaliação e documentação.

As linhas tracejadas indicam que há uma interação constante, embora mais fraca, entre as etapas associadas. As linhas cheias mostram o fluxo principal de trabalho no processo de construção de uma ontologia. A linha envolvendo as etapas de captura e formalização da ontologia realça a forte interação e, por conseguinte, iteração que ocorre entre suas etapas.

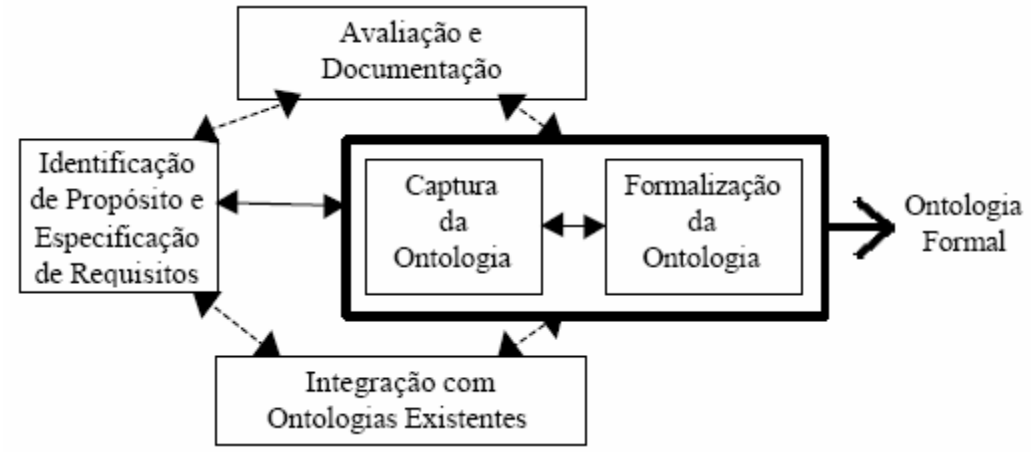

Figura 3.2: Etapas do Desenvolvimento de uma Ontologia e suas Interdependências (FALBO, 1998)

a) Identificação de Propósito e Especificação de Requisitos

Nesta etapa, identifica-se o propósito e o uso esperado pela ontologia. Identificam-se, também, os potenciais usuários e os cenários que motivam o desenvolvimento da ontologia. A especificação de requisitos da ontologia é expressa em questões de competência (GRÜNINGER *et al*., 1995). Ao se especificar um relacionamento entre as questões de competência e os cenários de motivação, estamos dando uma justificativa para a ontologia e, o mais importante, estamos provendo um mecanismo para sua avaliação.

b) Captura da Ontologia

Nesta etapa, devem ser definidos os conceitos e suas relações. Os conceitos primitivos, aqueles que não são passiveis de definição em termos de outros conceitos da ontologia, devem ser definidos utilizando linguagem natural e exemplos. Os conceitos, que são passíveis de descrição em termos de outros conceitos, devem ser definidos com referências claras a estes, com objetivo de facilitar a formalização.

É sugerido o uso de um modelo gráfico acompanhado de um dicionário de termos para facilitar a comunicação com especialistas do domínio, como também a utilização de taxonomias para organizar categorias e subcategorias interconectadas do conhecimento do domínio de interesse.

Outra importante atividade desta fase é a descrição dos axiomas. Os axiomas aqui descritos devem ser em linguagem natural e refletir as restrições existentes sobre o universo de discurso. Usa-se uma classificação quanto à natureza do axioma como guia de definição dos axiomas de uma ontologia.

O processo de definição de axiomas deve ser guiado pelas questões de competência. Os axiomas devem ser necessários e suficientes para expressar as questões de competência. Além disso, qualquer solução para uma questão de competência deve ser descrita pelos axiomas da ontologia e deve ser consistente com eles. Se os axiomas propostos não forem suficientes para este propósito, então, conceitos, relações ou axiomas adicionais devem ser introduzidos na ontologia. Neste sentido, a captura de uma ontologia é um processo iterativo e fortemente ligado à avaliação. Podem existir diferentes formas de se gerar axiomas para uma ontologia, sendo que as questões de competência devem ser usadas para avaliar a completude do conjunto de axiomas em um caso em particular (GRÜNINGER *et al*., 1994).

c) Formalização da Ontologia

Nesta etapa, deve-se representar explicitamente a conceituação capturada na etapa anterior (captura da ontologia) em uma linguagem formal. É aconselhado o uso da lógica de primeira ordem. A formalização se inicia pela especificação de símbolos não lógicos da linguagem: as *constantes*, denotando indivíduos específicos do universo de discurso; os *símbolos funcionais*, denotando funções; e os *predicados*, denotando propriedades e relações declaradas dos indivíduos. Após isto, é possível formar sentenças sobre os indivíduos do universo de discurso, os axiomas formais.

d) Integração com Ontologias Existentes

Durante os processos de captura e/ou formalização, pode surgir a necessidade de integrar a ontologia em questão com outras já existentes, visando aproveitar conceituações previamente estabelecidas. De fato, é uma boa prática desenvolver ontologias funcionais modulares, que sejam gerais e mais amplamente reutilizáveis e, quando necessário, integrá-las, obtendo o resultado desejado.

e) Avaliação e Documentação

Por último, a ontologia deve ser avaliada para verificar se satisfaz os requisitos estabelecidos na especificação. Adicionalmente, ela deve ser avaliada em relação a critérios de qualidade para o projeto de ontologias descritas por Gruber (1995):

- *Clareza*: uma ontologia deve comunicar efetivamente o significado projetado dos termos definidos e, assim, suas definições devem ser objetivas. Onde for possível, uma

definição completa é preferida em relação a uma definição parcial e todas as definições devem ser documentadas em linguagem natural, de modo a reforçar a clareza.

- *Coerência*: uma ontologia deve ser coerente, isto é, deve comportar apenas inferências consistentes com as definições. Coerência deve ser observada, também, em relação a conceitos definidos informalmente. Se uma sentença passível de ser inferida a partir dos axiomas da ontologia contradiz uma definição ou exemplo dado informalmente, então a ontologia é incoerente.
- *Extensibilidade*: uma ontologia deve ser projetada para antecipar usos do vocabulário compartilhado e, portanto, sua representação deve poder ser estendida e especializada. Em outras palavras, deve ser possível definir novos termos para usos especiais, com base no vocabulário existente, sem haver necessidade de rever definições existentes.
- *Compromissos de codificação mínimos*: a conceituação deve ser especificada no nível de conhecimento sem depender de uma tecnologia em particular de representação de conhecimento. Uma tendência de codificação surge, quando escolhas de representação são feitas puramente para a conveniência de notação ou implementação. Assim, essa tendência deve ser minimizada, já que agentes compartilhando conhecimento podem ser implementados em diferentes sistemas e paradigmas de representação.
- *Compromissos ontológicos mínimos*: o conjunto de compromissos ontológicos de uma ontologia deve ser minimizado, sendo capaz de suportar as atividades planejadas de compartilhamento de conhecimento. Uma ontologia deve fazer tão poucas imposições sobre o domínio quanto possível, permitindo especializações instanciações da ontologia sempre que necessário. Os compromissos ontológicos são baseados no uso adequado de um vocabulário e, portanto, podem ser minimizados através da especificação de uma teoria mais fraca (que admita um maior número de modelos), tendo definições restritas apenas para os termos essenciais à comunicação consistente do conhecimento da teoria.

É importante notar que a etapa de avaliação e documentação deve ser realizada em paralelo com as etapas de captura e formalização, em um processo iterativo. Para avaliar a completude da ontologia, especialmente no que se refere a seus axiomas, as questões de competência têm um papel fundamental.

Em um domínio particular, é possível escrever um número bastante grande de axiomas. Entretanto, surge uma questão: Como não escrever mais axiomas do que o necessário? O conjunto de axiomas deve ser necessário e suficiente para expressar as questões de competência e caracterizar suas soluções, e apenas isto (GRÜNINGER *et al*., 1995; FOX *et al*., 1993).

Todo o desenvolvimento da ontologia deve ser documentado, incluindo propósitos, requisitos e cenários de motivação, as descrições textuais da conceituação, a ontologia formal e os critérios de projeto adotados.

Os termos capturados na conceituação do universo de discurso devem ser descritos em um Dicionário de Termos, considerando dois princípios importantes: o princípio do vocabulário mínimo e o princípio da auto-referência. O princípio do vocabulário mínimo diz respeito ao vocabulário utilizado na definição dos termos da ontologia ser o menor possível e não apresentar ambigüidades. O princípio da auto-referência indica que a definição de um termo no Dicionário deve preferencialmente ser feita utilizando outros termos do Dicionário.

3.4 Linguagens para Representação de Ontologias

Linguagens de Ontologias permitem formalizar conceitos explícitos de modelos de domínio. Segundo Antoniou e van Harmelen (2004), uma linguagem para representação de ontologias deve atender aos seguintes requisitos:

- Possuir uma sintaxe bem definida;
- Possuir uma semântica formal;
- Possuir um suporte eficiente de raciocínio;
- Ser poderosa e suficientemente expressiva;
- Ter facilidade de expressão.

É evidente a importância de **uma sintaxe bem definida**. Esta é uma condição necessária para que uma ontologia possa ser processada adequadamente.

Uma **semântica formal** descreve os significados do conhecimento de forma precisa. Isto significa que as semânticas não se referem às interpretações subjetivas, ou seja, não são permitidas interpretações deferentes por diferentes pessoas (ou máquinas). A alternativa mais bem estabelecida e estudada é utilizar a lógica matemática para estabelecer uma semântica formal.

O uso de uma semântica formal permite que se façam inferências utilizando o conhecimento representado na ontologia. A seguir, serão exemplificadas algumas destas possíveis inferências:

- *Inferência sobre membros de uma classe*: se *x* é uma instância de uma classe *C*, e *C* é uma subclasse de *D*, então se pode concluir que *x* é uma instância de *D*.

- *Inferência sobre equivalência de classes*: se a classe *A* é equivalente a uma classe *B*, e a classe *B* é equivalente à classe *C*, então *A* é também equivalente a *C*.
- *Inferência sobre consistência*: suponha que tenhamos declarado que *x* é uma instância de *A* e que *A* é uma subclasse de $B \cap C$, A é subclasse de *D*, e *B* e *D* são disjuntas. Então temos uma inconsistência porque *A* deve ser vazio, mas ele tem uma instância *x*. Isto é uma indicação de erro na ontologia.
- *Inferência sobre classificação*: se declararmos que a ocorrência de um conjunto de valores de propriedades é uma condição suficiente para um objeto ser membro da classe *A*, então, se uma instância *x* satisfizer essas condições, pode-se concluir que *x* deve ser uma instância de *A*.

A semântica é um pré-requisito para **suporte eficiente de raciocínio**. Derivações como as anteriores podem ser automatizadas, uma vez que podem ser resolvidas por procedimentos finitos. Suporte ao raciocínio é importante porque isto permite:

- Verificar a consistência da ontologia e do conhecimento;
- Verificar relações intencionais entre classes;
- Classificar automaticamente instâncias nas classes.

Suporte automatizado ao raciocínio permite a verificação de muito mais casos que poderiam ser verificados manualmente. Verificação como as citadas anteriormente são valiosas para desenvolver ontologias de qualquer porte, onde muitos autores estão envolvidos, e para integração e compartilhamento de diversas ontologias.

Uma semântica formal e um suporte ao raciocínio são dados pelo mapeamento da uma linguagem de ontologia para um formalismo lógico conhecido, e pelo uso de ferramentas automatizadas de raciocínio que já existem nesse formalismo. A OWL é uma linguagem para representar ontologias, que é (particularmente) mapeada para a lógica descritiva, que é um subconjunto de lógica de predicados com suporte eficiente ao raciocínio.

3.4.1 OWL

A *World Wide Web Consortium* (W3C, 2004) padronizou a OWL (*Ontology Web Language*) como sendo uma linguagem para definição e instanciação de ontologias Web. A OWL pode ser usada para representar explicitamente significados de termos de um vocabulário e as relações entre eles (OWL, 2004).

Uma ontologia escrita em OWL deve incluir descrições de classes, propriedades e suas instâncias. A semântica formal da OWL especifica como derivar fatos não literalmente presentes na ontologia, mas que são produto de uma inferência lógica. Essa inferência pode ser baseada em um simples documento ou em múltiplos documentos distribuídos.

Na pilha de Linguagens propostas para Web Semântica (Figura 3.3), a linguagem OWL situa-se acima da camada de Metadados (expressos em RDF e RDF *Schema*).

RDF (*Resource Description Framework*) é uma linguagem para representar informações acerca de recursos na Web. RDF *Schema* é uma linguagem que permite construir modelos onde estão definidas, por exemplo, classes, subclasses, propriedades e "subpropriedades". As primitivas de modelagem presentes no RDF *Schema* são utilizadas na OWL.

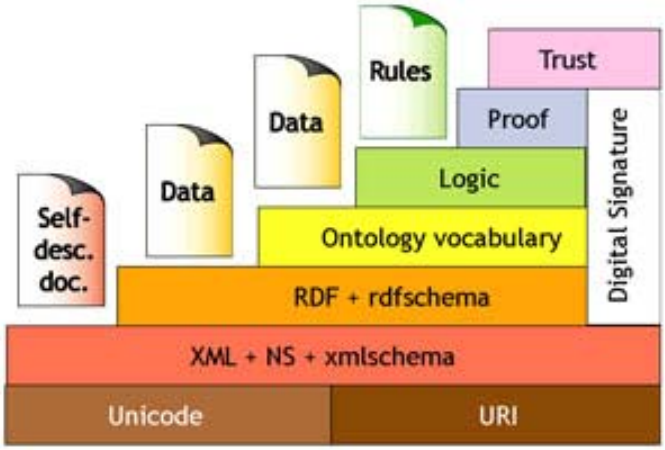

Figura 3.3: Representação da Web Semântica em camadas (ANTONIOU e van HARMELEN, 2004)

A OWL é constituída de três linguagens: *OWL Lite*; *OWL DL*; *OWL Full*. Esta subdivisão visa atender a diferentes necessidades dos usuários. Cada uma dessas três linguagens é uma extensão de sua predecessora mais simples. Podemos então concluir as seguintes relações, entre as linguagens, todavia não podemos concluir a inversa delas.

- Toda ontologia *OWL Lite* válida é uma ontologia *OWL DL* válida.
- Toda ontologia *OWL DL* válida é uma ontologia *OWL Full* válida.
- Toda conclusão *OWL Lite* válida é uma conclusão *OWL DL* válida.
- Toda conclusão *OWL DL* válida é uma conclusão *OWL Full* válida.

a) OWL Lite

A *OWL Lite* suporta todas as necessidades primárias de classificação e restrições simples. Por exemplo, suporta restrições de cardinalidade, contudo só permite valores de cardinalidade 0 ou 1. É mais simples fazer ferramentas para a *OWL Lite* do que para uma linguagem com mais restrições. A *OWL Lite* provê um caminho rápido para migração de enciclopédias e outras taxonomias. A *OWL Lite* também tem uma complexidade formal menor que a *OWL DL*.

A vantagem *OWL Lite* é de ser uma linguagem que é mais fácil de compreender (para usuários) e mais fácil de implementar (para os desenvolvedores). A desvantagem é a restrição de sua expressividade.

b) OWL DL

A *OWL DL* permite que sejam construídas expressões com base na lógica descritiva. A lógica descritiva é um subconjunto da lógica de primeira ordem, para a qual é possível garantir a completude computacional (garante que todas as conclusões válidas são computáveis) e a decidibilidade (é possível concluir, em um tempo finito, para cada sentença bem formada em OWL se ela é verdadeira ou falsa).

Existem restrições nas expressões válidas para *OWL DL*. Por exemplo: uma classe pode ser subclasse de muitas classes, contudo uma classe não pode ser uma instância de outra classe.

c) OWL Full

A *OWL Full* possibilita o máximo de expressividade e liberdade sintática permitidos em RDF, mas, para tanto, sacrifica a decidibilidade. Por exemplo: na *OWL Full*, uma classe pode ser tratada simultaneamente como uma coleção de itens e como um item por si mesmo. Não é possível automatizar completamente a inferência para todas as características da *OWL Full*.

A OWL usa RDF e RDF *Schema* em várias situações:

- As variações da OWL utilizam RDF e sua sintaxe.
- Instâncias são declaradas como na RDF, usando descritores RDF e informações de tipos.
- Construtores OWL como `owl:class`, e `owl:DatatypeProperty`, e `owl:ObjectProperty` são especializações das expressões correspondentes em RDF.

A Figura 3.4 mostra a relação de subclasses entre as primitivas em OWL e RDF / RDFS.

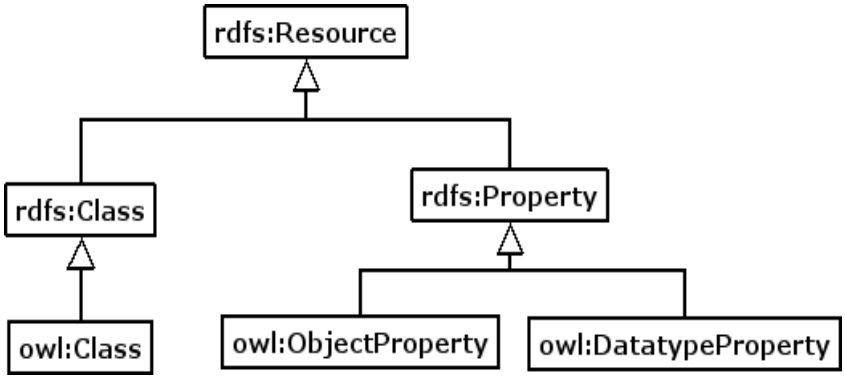

Figura 3.4: Relação de hierarquia entre as classes OWL e RDF / RDFS

A seguir, são apresentados os seguintes construtores da OWL: sintaxe, cabeçalho, classe, propriedade, restrição de propriedade, propriedades especiais, combinação booleana, enueração, instâncias, tipos de dado, e informação de versionamento.

d) Sintaxe

A OWL foi desenvolvida sobre a RDF e a RDF *Schema* e usa sua sintaxe baseada em XML da RDF. Essa sintaxe baseada em RDF/XML não é muito legível. Por causa disso, outras formas sintáticas da OWL foram definidas:

- Uma sintaxe baseada em XML, que não segue as convenções do RDF e é de mais fácil leitura para usuários humanos.
- Uma sintaxe abstrata, usada na especificação de documento (OWL, 2004), que seja mais compacta e legível que a sintaxe XML ou que a sintaxe RDF / XML.
- Uma sintaxe gráfica baseada em convenções da UML (*Unified Modeling Language*).

e) Cabeçalho

Um documento OWL é freqüentemente chamado de uma "ontologia OWL" e é um documento RDF. O elemento raiz de uma ontologia OWL é um elemento `rdf:RDF`, que é especificado num conjunto de namespaces[1]:

[1] *Namespace* é um contexto para identificadores. Um identificador definido em um *namespace* é associado a esse *namespace*. Um identificador pode ser definido em mais de um *namespace*, porém o seu significado é diferente para cada *namespace* (WIKIPEDIA, 2006).

```
<rdf:RDF
     xmlns:owl  ="http://www.w3.org/2002/07/owl#"
     xmlns:rdf  ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl  ="http://www.w3.org/2001/XMLSchmea#">
```

Uma ontologia OWL deve começar com uma coleção de declarações acerca da ontologia. Estas declarações são agrupadas sobre um elemento `owl:Ontology`, que contêm comentários, controle de versão, e inclusões de outras ontologias. Por exemplo:

```
<owl:Ontology rdf:about="">
     <rdfs:comment>Um exemplo de uma ontologia OWL</rdfs:comment>
     <owl:priorVersion rdf:resource="http://www.mydomain.org/uni-ns-old">
     <owl:imports rdf:resource="http://www.mydomain.org/persons">
     <rdf:label>Ontologia de um Jornal</rdfs:label>
</owl:Ontology>
```

A declaração `owl:imports` tem uma conseqüência para o significado lógico da ontologia, listando outras ontologias que têm seu conteúdo assumido como uma parte da ontologia corrente.

Note que, enquanto os *namespaces* são usados para retirar ambigüidades, a importação de ontologia provê definições que podem ser usadas. Freqüentemente, pode ser importado um elemento para cada *namespace* usado, mas é possível importar ontologias adicionais, como, por exemplo, ontologias que dêem definições sem introduzir nenhum novo termo.

É importante lembrar que a `owl:imports` é uma propriedade transitiva: se uma ontologia *A* importa uma ontologia *B*, e a ontologia *B* importa a ontologia *C*, então a ontologia *A* também importa a ontologia *C*.

f) Classe

Classes são definidas usando um elemento `owl:Class`. Por exemplo, a definição de uma classe *Reporter* seria:

```
<owl:Class rdf:ID="Reporter">
     <rdfs:subClassOf rdf:resource="#Empregado"/>
</owl:Class>
```

A classe *Reporter* pode ser representada conforme a Figura 3.5. Essa representação foi obtida através da ferramenta EzOWL (EzOWL, 2004),

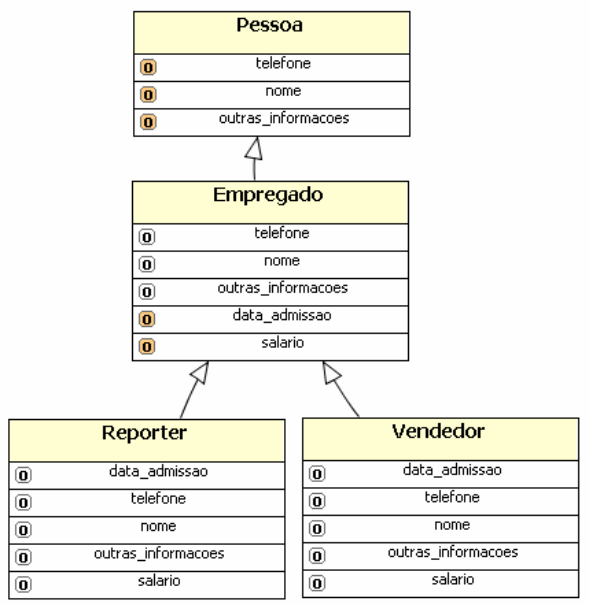

Figura 3.5: Classe Repórter

Pode-se representar que a classe *Reporter* e a classe de *Vendedor* são disjuntas, usando o elemento `owl:disjointWith` (Figura 3.6). Esse elemento pode ser incluído na definição da classe ou pode ser declarado em um outro lugar e referenciar o ID dele usando `rdf:about`. Este mecanismo é herdado do RDF.

```
<owl:Class rdf:about="#Reporter">
     <owl:disjointWith rdf:resource="#Vendedor"/>
</owl:Class>
```

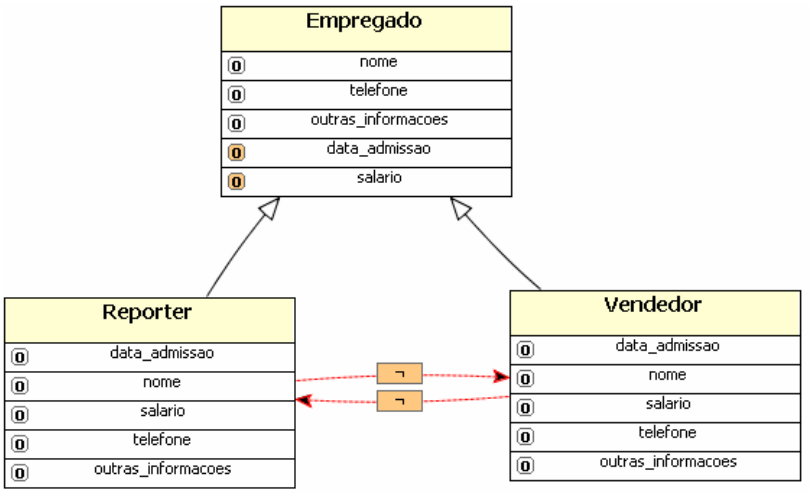

Figura 3.6: Disjunção da classe Repórter e Vendedor

A equivalência (Figura 3.7) de classes pode ser definida usando `owl:equivalentClass`:

```
<owl:Class rdf:ID="Apresentador">
     <owl:equivalentClass rdf:resource="#Reporter"/>
</owl:Class>
```

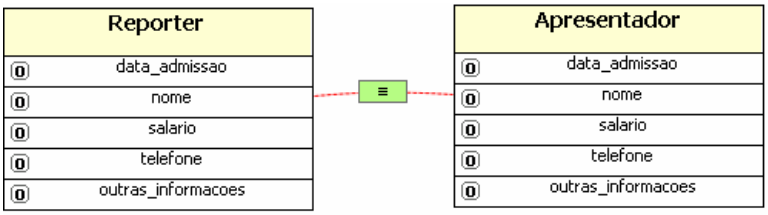

Figura 3.7: Equivalência da classe Repórter e Apresentador

Por fim, existem duas classes predefinidas, `owl:Thing` e `owl:Nothing`. Toda classe é subclasse de `owl:Thing` e superclasse de `owl:Nothing`.

g) Propriedade

A OWL tem as seguintes propriedades (Figura 3.8):

- *Propriedade de tipo de dado*: relaciona objetos com valores de tipo de dados. Por exemplo: tipo_artigo, palavra_chave, texto, etc. OWL não tem nenhum tipo de dado predefinido, nem proporciona facilidades especiais de definição. Contudo, permite o uso dos tipos de dados do XML Schema. Exemplo de uma propriedade de tipo de dado:

```
<owl:DatatypeProperty rdf:ID="nome">
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
     <rdfs:domain rdf:resource="#Pessoa"/>
</owl:DatatypeProperty>
```

- *Propriedades de objeto*: relaciona objetos com outros objetos. Por exemplo: a propriedade autor da classe *Artigo*. Exemplo de uma propriedade de objeto:

```
<owl:ObjectProperty rdf:ID="autor">
    <rdfs:domain rdf:resource="#Artigo"/>
    <rdfs:range rdf:resource="#Reporter"/>
</owl:ObjectProperty>
```

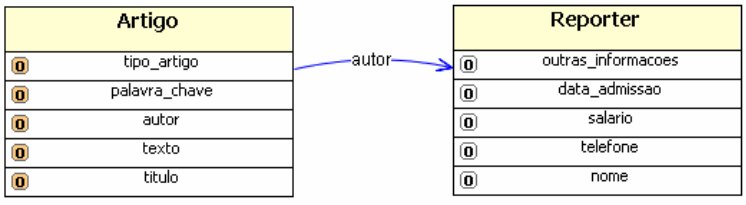

Figura 3.8: Propriedades das classes

Mais de um domínio e mais de um intervalo de valores podem ser declarados. Quando mais de um domínio for declarado deverá ser considerada a interseção entre eles. Da mesma forma, quando mais de um intervalo de valores for declarado, também deverá ser considerada a interseção entre eles.

A OWL permite descrever “propriedades inversas”. Por exemplo: um “Repórter escreve um Artigo” e um “Artigo é possui um Repórter” como autor. Neste caso, as propriedades inversas são “autor” e “escreve”, como se segue:

```
<owl:ObjectProperty rdf:ID="autor">
    <owl:inverseOf rdf:resource="#escreve"/>
    <rdfs:domain rdf:resource="#Artigo"/>
    <rdfs:range rdf:resource="#Reporter"/>
</owl:ObjectProperty>
```

A propriedade de equivalência (`owl:equivalentProperty:`) pode ser definida como no exemplo abaixo:

```
<owl:ObjectProperty rdf:ID="escreve">
     <owl:equivalentProperty rdf:resource="#e_autor"/>
</owl:ObjectProperty>
```

h) Restrição de Propriedade

Com `rdfs:subClassOf` pode-se especificar que uma classe *C* é subclasse de outra classe *C’*; então toda instância de *C* é também uma instância de *C’*.

Suponha que se declarasse ainda que a classe *C* satisfaz certas condições, ou seja, toda instância de *C* satisfaz as condições. É equivalente dizer que *C* é subclasse de *C*’, onde *C*’ tem todos os objetos que satisfazem essas condições. É desta forma que em OWL são descritas as restrições de uma superclasse anônima.

A restrição a seguir fala que um artigo policial só pode ser escrito por um repórter policial (Figura 3.9):

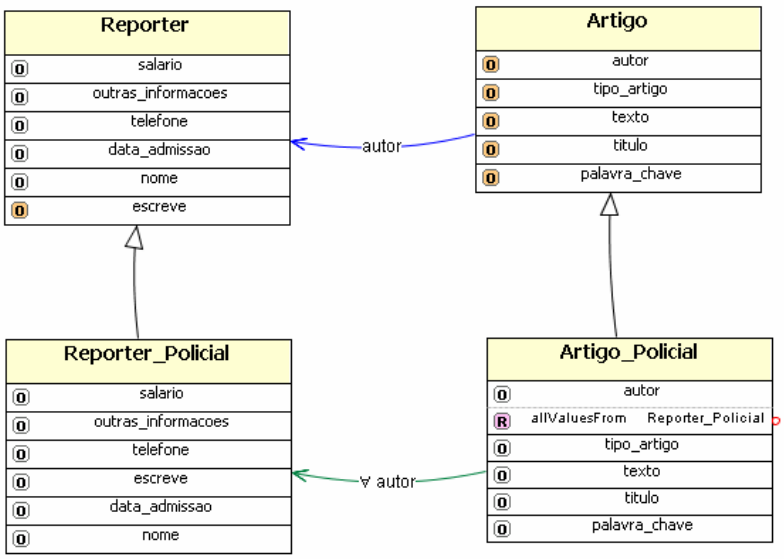

Figura 3.9: Restrição de propriedade: `allValuesFrom`

```
<owl:Class rdf:about="#Artigo_Policial">
     <rdfs:subClassOf>
          <owl:Restriction>
```

```
            <owl:onProperty rdf:resource="#autor"/>
            <owl:allValuesFrom rdf:resource="#Reporter_Policial”/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

A propriedade `owl:allValuesFrom` é usada para especificar a classe de valores possíveis que a propriedade especificada por `owl:onProperty` pode assumir, isto é, todos os valores da propriedade devem vir dessa classe. No exemplo apresentado, somente um repórter policial pode ter valores da propriedade `autor` da classe `Artigo_Policial`.

Pode-se declarar, por exemplo, que a primeira página deve ter como autor o repórter Dorneles Trindade (`rdf:resource="Dorneles_Trindade"`), da seguinte forma (Figura 3.10):

```
<owl:Class rdf:about="#Primeira_Pagina">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#autor"/>
            <owl:hasValue rdf:resource="Dorneles_Trindade"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

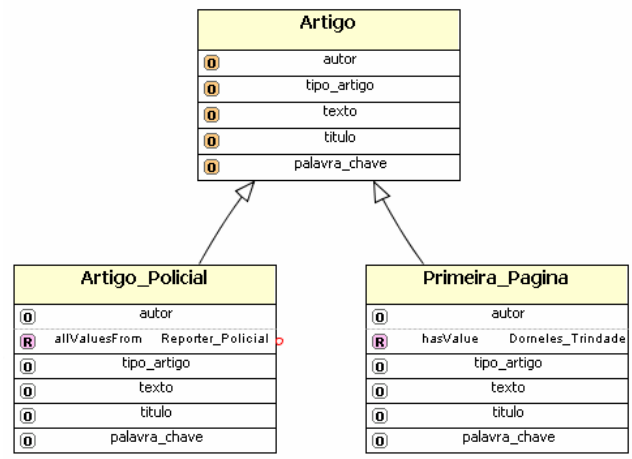

Figura 3.10: Restrição de propriedade: `hasValue`

A restrição `owl:hasValue` declara um valor específico que a propriedade especificada por `owl:onProperty` deve ter.

Pode-se declarar também que todos os repórteres devem escrever pelo menos um artigo (Figura 3.11):

```
<owl:Class rdf:about="#Reporter">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#escreve"/>
            <owl:sameValuesFrom rdf:resource="#Artigo"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

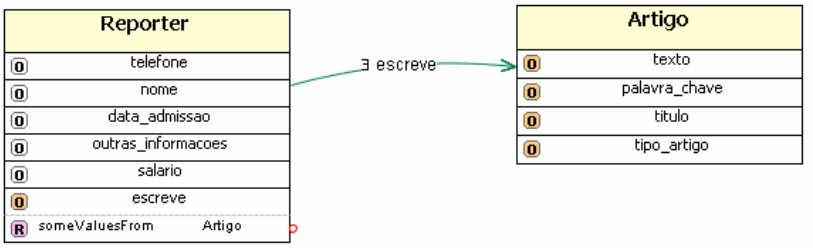

Figura 3.11: Restrição de propriedade: `someValuesFrom`

Um outro tipo de restrição é a de cardinalidade. Por exemplo, quer-se restringir que todo artigo seja escrito por pelo menos um repórter (Figura 3.12):

```
<owl:Class rdf:about="#Artigo">
     <rdfs:subClassOf>
          <owl:Restriction>
               <owl:onProperty rdf:resource="#autor"/>
               <owl:minCardinality
                      rdf:datatype="&xsd;nonNegativeInteger"/>
               1
               </owl:minCardinality>
          </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

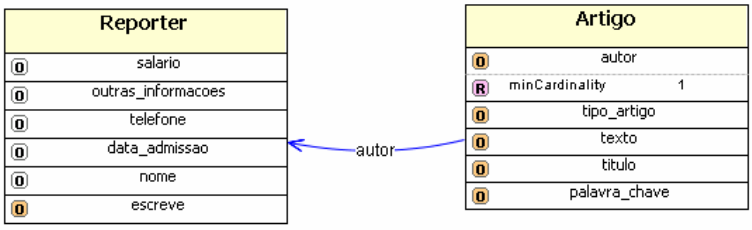

Figura 3.12: Restrição de propriedade: `minCardinality`

Note que o "1" teve que ser especificado para ser interpretado como um Inteiro não negativo (`nonNegativeInteger`), caso contrário ele seria interpretado como um texto. Foi usada também a declaração `xsd namespace`, feita no cabeçalho do documento para se referir a um documento XML *Schema*.

Pode-se querer especificar que, por razões práticas, uma edição do jornal deva ter no mínimo 15 e no máximo 40 artigos (Figura 3.13):

```
<owl:Class rdf:about="#Edicao">
     <rdfs:subClassOf>
          <owl:Restriction>
               <owl:onProperty rdf:resource="#artigos_desta_edicao"/>
               <owl:minCardinality
                      rdf:datatype="&xsd;nonNegativeInteger"/>
               15
               </owl:minCardinality>
          </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
          <owl:Restriction>
```

```
            <owl:onProperty rdf:resource="#artigos_desta_edicao"/>
            <owl:maxCardinality
                  rdf:datatype="&xsd;nonNegativeInteger"/>
            40
            </owl:minCardinality>
         </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

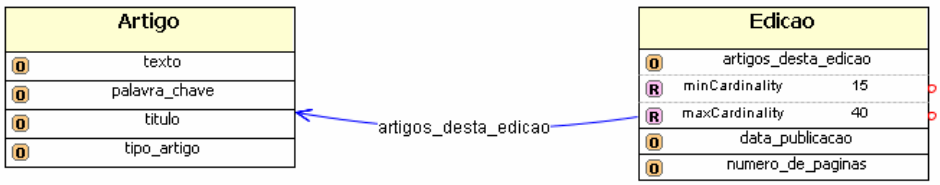

Figura 3.13: Restrição de propriedade: `minCardinality` **e** `maxCardinality`

Também é possível especificar um número preciso, por exemplo, um aluno deve ter exatamente dois supervisores. Isto é feito usando o mesmo número na `owl:minCardinality` e `owl:maxCardinality`. Por conveniência, a OWL também oferece `owl:cardinality` para especificar um número exato.

i) Propriedades Especiais

Algumas propriedades especiais de elementos podem ser definidas diretamente:

- `owl:TransitiveProperty` define uma propriedade transitiva. Por exemplo: "é mais alto que", "tem grade melhor que", ou "é predecessor de".
- `owl:SymmetricProperty` define uma propriedade simétrica. Por exemplo: "tem a mesma grade de", ou "é irmão de".
- `owl:FunctionalProperty` define uma propriedade que tem mais de um valor para cada objeto. Por exemplo: "idade", "altura", ou "supervisorDireto".
- `owl:InverseFunctionalProperty` define uma propriedade tal que dois objetos diferentes não possam ter o mesmo valor. Por exemplo: "CPF" (um CPF deve estar associado a uma pessoa apenas).

Exemplo das formas sintáticas apresentadas acima:

```
<owl:ObjectProperty rdf:ID="#autor">
     <rdf:type rdf:resource="&owl;TransitiveProperty" />
     <rdf:type rdf:resource="&owl;SymmetricProperty" />
     <rdfs:domain rdf:resource="#Artigo" />
     <rdfs:range rdf:resource="#Artigo" />
</owl:ObjectProperty>
```

j) Combinação Booleana

É possível falar sobre combinações booleanas (união, interseção, complemento) das classes (definidas por `owl:Class` ou expressões de classes). Por exemplo, pode-se dizer que artigos publicados e artigos não publicados são disjuntos (uma instância da classe `Artigo_publicado` não pode também ser uma instância da classe `Artigo_nao_publicado`) da seguinte maneira (Figura 3.14):

```
<owl:Class rdf:about="#Artigo_publicado">
<rdfs:subClassOf>
          <owl:Class>
               <owl:complementOf rdf:resource="#Artigo_nao_publicado"/>
          </owl:Class>
</rdfs:subClassOf>
</owl:Class>
```

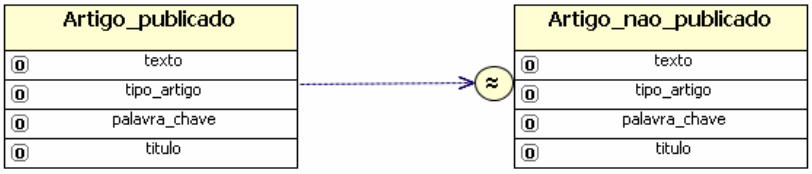

Figura 3.14: Combinação booleana: `complementOf`

Isso diz que toda classe `Artigo_publicado` é uma instância do complemento da classe `Artigos_nao_publicado`, ou seja, nenhum "artigo publicado" é um "artigo não publicado". A mesma coisa poderia ser dita usando `owl:disjointWith`.

A união de classes é construída usando `owl:unionOf` (Figura 3.15):

```
<owl:Class rdf:ID="Artigo">
   <owl:equivalentClass>
     <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Artigo_publicado" />
          <owl:Class rdf:about="#Artigo_nao_publicado" />
     </owl:unionOf>
   </owl:equivalentClass>
</owl:Class>
```

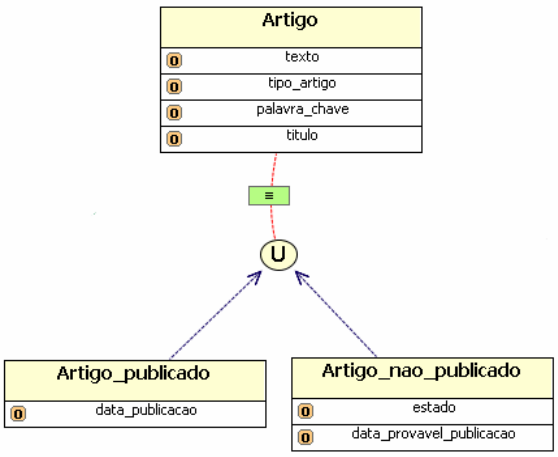

Figura 3.15: Combinação booleana: `unionOf`

Isto não significa dizer que a nova classe é uma subclasse da união, mas diz que a nova classe é "igual" a união. Em outras palavras, tem-se um estado de equivalência de classes. Também, não foi especificado que as duas classes devem ser disjuntas. Neste caso, é possível que um "artigo publicado" também seja um "artigo não publicado".

A interseção é iniciada com `owl:intersectionOf` (Figura 3.16):

```
<owl:Class rdf:ID="Pessoa_Empregada">
   <owl:equivalentClass>
     <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Pessoa" />
          <owl:Class rdf:about="#Empregado" />
     </owl:intersectionOf>
   </owl:equivalentClass>
</owl:Class>
```

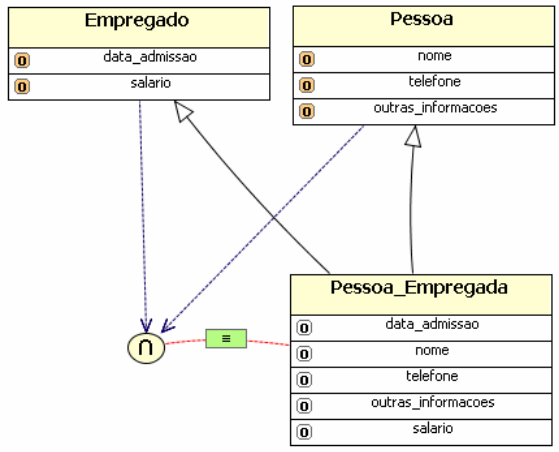

Figura 3.16: Combinação booleana: `intersectionOf`

Neste contexto, foi construída a interseção de duas classes: a classe de empregados e a classe de pessoas. A interseção dessas duas classes é equivalente a classe de pessoas empregadas.

k) Enumeração

Uma enumeração é especificada através de uma expressão `owl:oneOf`, usada para definir uma classe pela relação de todos os seus elementos:

```
<owl:Class rdf:ID="diasDaSemana">
     <owl:oneOf rdf:parseType="Collection">
          <owl:Thing rdf:about="#Domingo" />
          <owl:Thing rdf:about="#SegundaFeira" />
          <owl:Thing rdf:about="#TerçaFeira" />
          <owl:Thing rdf:about="#QuartaFeira" />
          <owl:Thing rdf:about="#QuintaFeira" />
          <owl:Thing rdf:about="#SextaFeira" />
          <owl:Thing rdf:about="#Sabado" />
     </owl:oneOf>
</owl:Class>
```

l) Instâncias

A instância de uma classe é declarada como em RDF:

```
<rdf:Description rdf:ID="949352">
     <rdf:type rdf:resource="#Reporter" />
</rdf:Description>
```

Pode ser declarada também da seguinte forma:

```
<Reporter rdf:ID="949352" />
```

Isto pode ser melhor detalhado como:

```
<Reporter rdf:ID="949352">
    <telefone>3322-1122</telefone>
    <nome>Dorneles Trindade</nome>
</Reporter>
```

Diferentemente dos sistemas de banco de dados, a OWL não adota a hipótese de nomes únicos para indivíduos diferentes, ou seja, não é porque duas instâncias têm nomes ou ID diferentes que elas sejam indivíduos diferentes. Por exemplo, se determinarmos que cada artigo é escrito por no máximo um reporter:

```
<owl:ObjectProperty rdf:ID="autor" />
     <rdf:type redf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>
```

E depois, em outra sentença podemos colocar um curso dado por dois professores:

```
<Artigo rdf:ID="CIT1111" />
     <autor rdf:resource="#949318" />
     <autor rdf:resource="#949352" />
</Artigo>
```

Isto não causa um erro de inconsistência na OWL. No entanto, o sistema irá inferir que os objetos “949318” e “949352” são aparentemente iguais. Para assegurar que diferentes indivíduos sejam de fato reconhecidos como diferentes, deve-se explicitamente declarar esta diferença:

```
<lecturer rdf:ID="949318" />
     <owl:differentFrom rdf:resource="#949352" />
</lecturer>
```

Pelo fato de a declaração de desigualdade ocorrer com freqüência, o número de declarações de desigualdade poderia explodir se fosse necessário declarar um grande número de desigualdades, a OWL estabeleceu uma notação mais resumida para declarar a desigualdade de todos os indivíduos de uma lista:

```
<owl:AllDiffent>
     <owl:distinctMembers rdf:parseType="Collection">
          <lecturer rdf:about="#949318" />
          <lecturer rdf:about="#949352" />
          <lecturer rdf:about="#949111" />
     </owl:distinctMembers>
</owl:AllDiffent>
```

A `owl:distinctMembers` só pode ser usada concomitantemente com `owl:allDifferent`.

m)Tipos de Dado

Apesar do XML *Schema* prover um mecanismo de construção de tipos de dados definidos pelos usuários (por exemplo, um tipo de dado `idadeAdulta` que pode ser composto por todos os inteiros maiores de 18, ou um tipo de dados onde todas as *strings* devam começar com um número), tal tipo de dados derivado não pode ser usado na OWL. Na verdade, alguns dos tipos de dados construídos em XML *Schema* não podem ser usados em OWL. O documento de referência da OWL lista todos os tipos de dados que podem ser usados, que incluam os tipos de dados mais usados tais como: *string*, inteiro, booleano, tempo e data.

n) Informação de Versionamento

Já foi visto no cabeçalho a sentença `owl:priorVersion` que indica logo a versão dessa ontologia. Esta informação não tem nenhuma semântica formal, contudo ela pode ser usada por leitores humanos e por programas para gerenciamento da ontologia.

Além da `owl:priorVersion`, a OWL possui outras sentenças que dão informações sobre o versionamento:

- `owl:versionInfo` geralmente contém um texto informando sobre a versão corrente. Por exemplo, informações de programas que gerenciam o controle de versão.
- `owl:backwardCompatibleWith` contém uma referência a outras ontologias. Ela identifica que a ontologia deste documento é compatível com a versão anterior. Particularmente, isto indica que todos os identificadores da versão anterior têm a mesma interpretação da nova versão.
- `owl:incompatibleWith`, indica que a versão anterior desta ontologia não é compatível com a ontologia atual.

Depois de serem apresentados os construtores da OWL, já se pode especificar com completitude cada característica da linguagem que estarão presentes em cada uma das três linguagens da OWL.

o) OWL Full

Na OWL Full, todos os construtores podem ser usados em qualquer combinação contanto que o resultado seja um RDF válido.

p) OWL DL

A fim de utilizar o suporte formal e a correspondência com a Lógica Descritiva, algumas regras devem ser obedecidas em uma ontologia OWL DL:

- *Partição de Vocabulário*: qualquer recurso somente poderá ser uma classe, um tipo de dado, uma propriedade de um tipo de dado, uma propriedade de um objeto, um indivíduo, um valor de dado ou parte do vocabulário pré-construído. Isto significa que, por exemplo, uma classe não pode ser ao mesmo tempo um individuo, ou ainda, que uma propriedade não pode ter alguns valores que são um tipo de dado e outros valores que são classes (pois uma propriedade não pode ser ao mesmo tempo uma *data type property* e uma *object property*).
- *Tipo Explicito*: não somente todos os recursos devem ser particionados (como descrito no item anterior), mas estas partições devem ser explicitas. Por exemplo, se uma ontologia contém o seguinte:

```
<owl:Class rdf:ID="C1">
     <rdfs:subClassOf rdf:about="#C2" />
</owl:Class>
```

É também deduzido que `C2` é uma classe (em virtude da especificação `rdfs:subClassOf`). Contudo, uma ontologia OWL DL deve conter esta informação explicitada:

```
<owl:Class rdf:ID="C2" />
```

- *Separação de propriedade*: em virtude da primeira regra acima indicada, o conjunto de `object properties` e `data type properties` são disjuntos. Isto implica que nenhum dos seguintes tipos pode ser especificado para `data type properties`:

`owl:inverseOf`,
`owl:FunctionalProperty`,
`owl:InverseFunctionalProperty`, e
`owl:SymmetricProperty`.

- *Não são permitidas restrições de cardinalidade transitiva*: nenhuma restrição de cardinalidade pode ser aplicada em propriedades transitivas (ou suas subpropriedades, que são também transitivas, por implicação).
- *Restrição no uso de classes anônimas* (`anonymous`): classes anônimas só podem ocorrer como domínio ou imagem em `owl:equivalentClass` ou `owl:disjointWith`, e como imagem (mas não domínio) do `rdfs:subClassOf`.

q) OWL Lite

Uma ontologia OWL Lite válida deve ser uma ontologia OWL DL válida e deve também satisfazer as seguintes regras:

- Os construtores `owl:oneOf`, `owl:disjointWith`, `owl:unionOf`, `owl:complementOf` e `owl:hasValue` não são permitidos.
- Instruções de cardinalidade (mínimo, máximo e cardinalidade exata) podem somente ser utilizadas através dos valores 0 e 1 e não mais qualquer valor inteiro não negativo.
- A sentença `owl:equivalentClass` não pode mais ser usada entre classes anônimas, mas somente entre classes identificadas.

3.5 Conclusão

As ontologias estão sendo usadas cada vez mais na computação para compartilhar um entendimento comum de uma estrutura de informação entre pessoas e agentes de software e para possibilitar o reuso do conhecimento de um domínio, dentre outras utilizações. No capítulo seguinte, será proposta uma ontologia para engenharia de requisitos de software.

Capítulo 4

UMA ONTOLOGIA PARA ENGENHARIA DE REQUISITOS

Este capítulo apresenta uma ontologia para engenharia de requisitos, fornecendo suporte a documentação, métricas e qualidade de requisitos.

4.1 Metodologia

Para o desenvolvimento da ontologia proposta neste trabalho, utilizou-se a metodologia proposta por Noy e McGuiness (2001). Essa metodologia utiliza a iteratividade para aprimoramento da própria ontologia. Ao longo da definição da ontologia, são discutidas decisões de modelagem que o autor da ontologia precisará tomar, onde são apresentados prós e contras e implicações de possíveis soluções.

A metodologia de Noy e McGuiness (2001) é composta pelas seguintes etapas:

a) Determinar o domínio e o escopo da ontologia:

A ontologia proposta deve ser capaz de responder as seguintes questões, que podem auxiliar na determinação do escopo da ontologia:

- *O que o domínio da ontologia deve cobrir*?
- *Para que alguém usará essa ontologia*?
- *Para que tipos de questões as informações contidas na ontologia devem responder*?
- *Quem usará e manterá a ontologia*?

Nesta etapa, também foram usadas *questões de competência* propostas por Gruninger e Fox (1995). As questões de competência envolvem um conjunto de perguntas, que delimitam o escopo da ontologia e servem como diretrizes para sua construção.

b) Procurar e reusar ontologias existentes:

Procuram-se trabalhos relacionados com o objetivo de refiná-los ou estendê-los. O reuso de ontologias existentes pode ser um requisito da ontologia proposta, caso ela deva interagir com aplicações que já desenvolveram uma ontologia ou um vocabulário controlado.

c) Listar termos importantes da ontologia:

Devem ser definidos e listados termos e conceitos relevantes utilizados na ontologia. Os termos capturados na conceituação do universo de discurso devem ser descritos, considerando dois princípios importantes: o *princípio do vocabulário mínimo* e o *princípio da auto-referência*. O princípio do vocabulário mínimo diz respeito ao vocabulário utilizado na definição dos termos da ontologia. Esse vocabulário deve ser o menor possível e não deve apresentar ambigüidades. O princípio da auto-referência indica que a definição de um termo deve, sempre que possível, ser feita utilizando outros termos.

d) Definir classes e hierarquia entre elas:

As classes devem ser definidas e hierarquizadas entre si. Para esta tarefa sugere-se usar uma das seguintes abordagens:

- *Top-down*: o processo de desenvolvimento começa com a definição de classes de conceitos mais gerais do domínio. Depois, essas classes devem ser especializadas.
- *Bottom-up*: o processo de desenvolvimento começa com a definição de classes mais específicas. A hierarquia será construída com o agrupamento consecutivo dessas classes em conceitos mais genéricos.
- *Combinação*: o processo de desenvolvimento é uma combinação das duas abordagens (*top-down* e *bottom-up*). Em geral, os conceitos mais evidentes são definidos no início; depois, são geradas as generalizações e especificações apropriadas.

Nenhuma dessas abordagens é melhor que a outra; contudo, para o desenvolvimento da ontologia de engenharia de requisitos proposta foi utilizada a abordagem '*combinação'*, por ser considerada mais flexível, pois os conceitos 'intermediários' tenderam a serem mais descritivos no domínio.

e) Definir propriedades das classes

As classes por si só não proverão informação suficiente para responder as questões de competência da primeira etapa (determinar o domínio e o escopo da ontologia). Uma vez que as classes tenham sido definidas, deve-se descrever a estrutura interna delas. Como as classes foram selecionadas entre os termos listados na terceira etapa, a maioria dos termos que restaram são propriedades dessas classes.

f) Definir as restrições das propriedades

As restrições de cada propriedade das classes devem ser definidas. Essas restrições podem ser: tipo de dados, valores permitidos, cardinalidade, etc.

g) Criar instâncias das classes

Nesta última etapa, as instâncias das classes são criadas. A definição de cada instância envolve os seguintes passos:

- Escolher uma classe;
- Criar uma instância; e
- Preencher suas propriedades.

4.2 Escopo e Domínio da Ontologia

A ontologia proposta tem como objetivo melhorar a comunicação e o entendimento entre os envolvidos no processo de definição dos requisitos a que o software deverá atender. O escopo dessa ontologia é definido por meio de questões de competência (GRÜNINGER *et al*., 1995). Inicialmente, apresentam-se algumas questões de competência relacionadas com o entendimento do que vem a ser um requisito e como o requisito é transformado durante o processo de desenvolvimento de software:

- *O que é um requisito*?
- *Quais os tipos de requisitos*?
- *Como os requisitos são refinados e usados no processo de construção de software*?

Elencam-se, também, algumas questões de competência que se referem à qualidade dos requisitos:

- *Que métricas estão associadas a que requisitos*?
- *Que critérios de qualidade são desejáveis para os requisitos*?

No que se refere ao suporte de levantamento e à documentação de requisitos, foram elaboradas as seguintes questões:

- *Como levantar requisitos adequadamente*?
- *Como rastrear os requisitos capturados*?

4.3 Reuso de ontologia

A "Ontologia de Qualidade de Software", proposta por Duarte (2000), foi reusada parcialmente para a elaboração da ontologia para engenharia de requisitos.

A ontologia para qualidade de software tem como objetivo apoiar a compreensão do domínio de qualidade de software. A seguir, são apresentadas as questões de competência, os termos relevantes e o modelo dessa ontologia. Esse modelo havia sido feito em uma outra linguagem de modelagem e, neste trabalho, foi feito na linguagem OWL.

a) Questões de competência

Duarte (2000) apresentou um conjunto de questões de competência em sua ontologia proposta. São apresentadas abaixo as questões de competência referentes à medição da qualidade e que serão reutilizadas neste trabalho:

- *Como uma característica de qualidade por ser medida*?
- *Que métricas podem ser usadas para quantificar uma dada característica*?

b) Termos relevantes da ontologia

A ontologia para qualidade de software apresenta um conjunto de termos relevantes. A seguir são listados os termos importantes para as questões de competência definida no item anterior.

- *Característica de qualidade*: atributo de um artefato ou processo de software através do qual a qualidade de software pode ser avaliada.
- *Característica de qualidade diretamente mensurável*: pode ser computada diretamente por meio de uma métrica. Exemplo: número de páginas.
- *Característica de qualidade indiretamente mensurável*: somente pode ser computada combinando resultado de subcaracterísticas de qualidade.
- *Cálculo*: indica que uma característica indiretamente mensurável deve ser calculada por meio de outras características de qualidade diretamente ou indiretamente mensurável.
- *Correlação*: relação entre as classes de métricas e de características diretamente mensuráveis. A correlação indica que essas características podem ser computadas por meio de métricas. Exemplo: a métrica "número de alterações sofridas por um

requisito" pode ser usada para computar a característica diretamente mensurável "estabilidade do requisito".

- *Determinação*: relação entre as classes de medição e medida. A determinação indica que uma medição determina uma medida. Exemplo: contar o número de alterações sofridas determina um valor, esse valor é a medida.
- *Métrica*: é a unidade de medição. Exemplo: número de requisitos do sistema.
- *Medida*: valor que uma métrica assume no ato da medição. Exemplo: 12 (doze) requisitos.
- *Medição*: atividade que relaciona uma medida a uma métrica. Exemplo: obter o número de requisitos de um sistema.
- *Quantificação*: indica que uma medida quantifica uma métrica. Exemplo: a medida "12 requisitos" quantifica a métrica "número de requisitos do sistema".
- *Subcaracterísticas*: indica que uma característica indiretamente mensurável é decomposta em outras características.

c) Modelo

A qualidade pode ser entendida como um conjunto de características a serem satisfeitas em um determinado grau, de modo que o produto de software atenda às necessidades explícitas e implícitas de seus usuários. No entanto, não se obtém a qualidade do produto espontaneamente. A qualidade deve ser construída. Portanto, a qualidade do produto depende seguramente da qualidade de seu processo de desenvolvimento.

A avaliação da qualidade é feita por meio de métricas. Para cada característica de qualidade são escolhidas métricas, que quantificam a característica de qualidade. As métricas escolhidas devem ser capazes de quantificar uma característica de qualidade e devem ser feitas medições para determinar essa medida. Uma medida é o resultado da aplicação de uma métrica. Por exemplo, caso se deseje saber o tamanho de um determinado produto de software, poderia ser utilizada a métrica: "Número de linhas de código" (LOC). A medição seria: "contar o número de linhas de código, não considerando comentários, do conjunto de programa do software". E a medida seria o número obtido na medição, uma quantidade, por exemplo, de "5.000 linhas de código".

Algumas características de qualidade não podem ser medidas diretamente através de métricas. As características de qualidade, que não podem ser medidas diretamente, são

chamadas de características indiretamente mensuráveis. As características que podem ser medidas diretamente são chamadas de características diretamente mensuráveis.

Para medir as características de qualidade indiretamente mensuráveis é necessário decompô-las em subcaracterísticas. As subcaracterísticas, resultado da decomposição, podem ser diretamente mensuráveis ou indiretamente mensuráveis. As subcaracterísticas indiretamente mensuráveis devem ser decompostas novamente em outras subcaracterísticas até que se tenham apenas subcaracterísticas diretamente mensuráveis. Uma característica (ou subcaracterística) diretamente mensurável possui uma métrica relacionada a ela que, através de uma medição, se aplica uma medida. As características indiretamente mensuráveis podem ser descritas em termos de outras suas subcaracterísticas, sendo calculada por meio de subcaracterísticas diretamente mensuráveis.

O modelo de medição da qualidade proposto por Duarte (2000) foi remodelado neste trabalho em OWL (Figura 4.1), utilizando a ferramenta ezOWL (EzOWL, 2004).

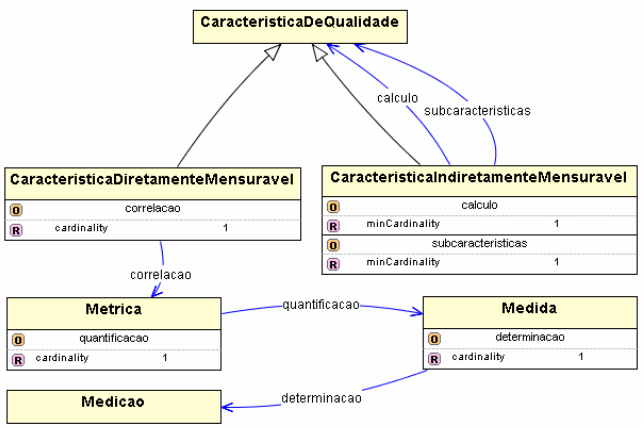

Figura 4.1: Medição da qualidade

4.4 Termos Relevantes na Ontologia de Engenharia de Requisitos

São definidos a seguir os termos usados na ontologia de requisitos de software proposta:

- *Artefato de documentação*: utilizado para documentar os requisitos de sistema.
- *Componente*: parte do produto de software entregue ao contratante (CMMI, 2006).
- *Cliente*: são todos os *stakeholders* que não fazem parte do grupo de desenvolvimento (vide subseção 2.1).
- *Características de qualidade*: avaliam a qualidade de um requisito de software. Essas características foram baseadas em Beaufond (1997) e constituíram um questionário que foi aplicado a especialistas em requisitos (Apêndice B). As seis características melhores avaliadas foram utilizadas neste trabalho:

- *Avaliabilidade*: capacidade de o requisito poder ser avaliado com relação à forma e ao conteúdo.
- *Completitude*: o requisito deve garantir que os aspectos abordados foram totalmente cobertos.
- *Consistência*: o requisito deve estar isento de contradições entre os aspectos especificados para ele.
- *Não-ambigüidade*: o requisito deve abordar apenas aspectos considerados imprescindíveis a seu entendimento, evitando-se, assim, descrições desnecessárias e irrelevantes.
- *Não-redundância*: o requisito não deve abordar aspectos tratados em outro requisito, evitando-se, assim, que uma mesma questão seja tratada em mais de um lugar.
- *Rastreabilidade*: o requisito deve ser rastreável, identificando-se a agregação de detalhes de dados específicos, desde sua visão mais global, até a mais detalhada, e vice-versa.

- *Necessidade*: indica a contribuição para o problema (oportunidade de negócios), que deve ser resolvido para justificar a aquisição ou o uso do software.
- *Regras de negócio*: declaração de políticas ou condições que devem ser satisfeita dentro do negócio. Podem ser classificadas em ambientais e organizacionais.
  - *Regras de negócio ambientais*: representam as regras de negócio relacionadas com o contexto político, econômico e padrões que podem afetar a organização (TORANZO *et al.*, 2002).
  - *Regras de negócio organizacionais*: constituem as regras da organização. As regras de aquisição e o uso dos sistemas computacionais originam-se destas informações (TORANZO *et all*, 2002).
- *Requisito*: são descrições de condição ou capacidade com a qual um sistema deve estar de acordo, seja ela diretamente derivada de necessidades dos usuários ou declarada em um contrato, um padrão, uma especificação ou outro documento afim (RUP, 2003).
  - *Requisito funcional*: "são declarações de funções que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como se deve comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer" (SOMMERVILLE, 2003).

- *Requisitos não funcionais*: "são restrições sobre serviços ou funções oferecidas pelo sistema. Entre eles destacam-se restrições de tempo, restrições sobre o processo de desenvolvimento, padrões, entre outros" (SOMMERVILLE, 2003).

A classificação dos requisitos não funcionais utilizados neste trabalho foi baseada na taxonomia proposta por Sommerville (2003) (vide Figura 2.3). Segue essa classificação:

- *Restrições de projeto*: representam decisões de projeto que foram impostas e devam ser obedecidas. Por exemplo: linguagens de software, requisitos de processo de software, uso prescrito de ferramentas de desenvolvimento, restrições de *design* e de arquitetura, componentes comprados, bibliotecas de classes, etc. Estes requisitos enquadram-se nas taxonomias "Requisitos de Implementação" e "Requisitos de Padrão" de Sommerville (2003).
- *Suportabilidade*: representam requisitos que aprimorarão a suportabilidade ou manutenibilidade do sistema que está sendo criado, incluindo padrões de codificação, convenções de nomeação, bibliotecas de classes, acesso à manutenção e utilitários de manutenção. Estes se enquadram nas taxonomias de "Requisitos de Confiabilidade" e "Requisitos de Portabilidade" de Sommerville (2003).
- *Eficiência*: representam requisitos que levam o produto de software realizar suas funções sem desperdício de recursos, sob condições especificadas. Enquadra-se nas taxonomias "Requisitos de Espaço" e "Requisitos de Desempenho" de Sommerville (2003). Por exemplo:
  - Tempo de resposta de uma transação (médio, máximo).
  - Taxa de transferência (ex.: transações por segundo).
  - Capacidade (ex.: o número de clientes ou de transações que podem ser acomodados pelo sistema).
  - Modos de degradação (o modo aceitável de operação quando o sistema tiver sido degradado de alguma maneira).
  - Utilização de recursos (ex.: memória, disco, comunicações, etc.).
- *Confiabilidade*: correspondem aos requisitos que levam o produto de software manter-se em um determinado nível de desempenho, quando usado sob condições especificadas. Enquadra-se na taxonomia "Requisitos de Facilidade de Uso" de Sommerville (2003). Por exemplo:
  - Disponibilidade: tempo que o sistema deve estar disponível para o uso dos usuários, para o acesso à manutenção, para as operações de modo degradado, etc.

- Tempo Médio entre Falhas (MTBF): normalmente especificado em horas, mas também poderá ser especificado em termos de dias, meses ou anos.
- Tempo Médio para Reparo (MTTR): tempo que o sistema poderá ficar sem funcionar após uma falha.
- Exatidão: especificação da precisão (resolução) e exatidão (através de algum padrão conhecido) necessários na saída dos sistemas. Ex.: quantidade de casas decimais em operações matemáticas.
- Taxa máxima de erros ou defeitos: geralmente expressa em termos de erros/KLOC ou de erros/ponto de função.
- Taxa de erros ou defeitos: categorizados em termos de erros pouco importantes, importantes e críticos: o(s) requisito(s) deve(m) definir o que se entende por um erro "crítico" (ex.: perda total de dados ou total incapacidade de usar determinadas partes da funcionalidade do sistema).

- *Usabilidade*: correspondem aos requisitos que levam o produto de software ser entendido, aprendido, usado e atrativo para o usuário, quando usado sob determinadas condições Por exemplo:
  - Tempo de treinamento necessário para que usuários normais e usuários com conhecimentos avançados tornem-se produtivos em operações específicas.
  - Período de tempo mensurável para tarefas típicas.
  - Requisitos que estejam em conformidade com os padrões comuns de usabilidade. Ex.: padrões CUA da IBM ou padrões GUI da Microsoft.

4.5 O Modelo Proposto

A ontologia para engenharia de requisitos proposta foi elaborada a partir da OWL (W3C, 2006), utilizando-se conceitos de classe e propriedade. As fases “Definir classes e hierarquia”, “Definir propriedades das classes”, “Definir as restrições das propriedades”, que compõem a metodologia utilizada (seção 4.1), serão apresentadas em conjunto. Também serão descritas algumas decisões de modelagem tomadas.

Inicialmente, construiu-se a classe `Cliente`. Essa classe está relacionada com a classe `Necessidade`, através da propriedade `temNecessidade`, cuja cardinalidade é maior que um, já que não faz sentido que um sistema seja construído sem necessidades. Na classe `Necessidade`, documenta-se todas as necessidades (propriedade `descricao`) do cliente. A propriedade `atendidaPeloRequisito` da classe `Necessidade` informa como essas necessidades estão

sendo atendidas nos requisitos levantados. A classe `Requisito` é composta pela união da classe `RequisitoFuncional` e da classe `RequisitoNaoFuncional` (Figura 4.2).

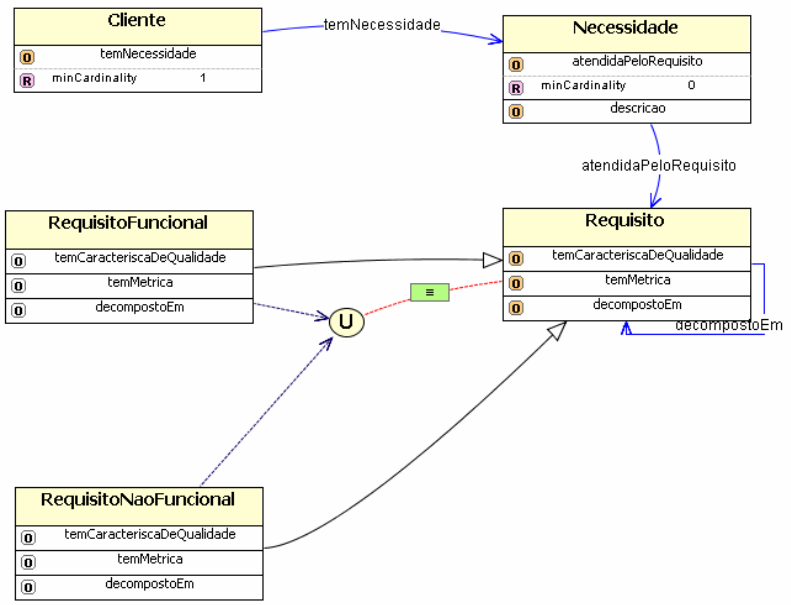

Figura 4.2: Modelo para requisitos do cliente

Na classe `Requisito`, têm-se requisitos documentados e controlados. Um requisito pode ser decomposto em outros requisitos (propriedade `decompostoEm`). Os requisitos podem ser divididos em três níveis (Capítulo 2): *requisitos do usuário*, *requisitos de sistema* e *requisito de componente*. Os requisitos de usuário são requisitos abstratos de alto nível. Os requisitos de sistema descrevem de forma mais detalhada o que o sistema deverá fazer. Os requisitos de componente são mais detalhados ainda, associando a engenharia de requisitos com as atividades de projeto.

Na classe `Requisito`, tem-se a propriedade `temMetrica`, que relaciona um requisito a uma métrica. Como exemplo disto, pode-se citar a quantidade de pontos por função que este requisito possui.

A propriedade `temCaracteristicaDeQualidade` relaciona um requisito a uma característica que deve ser atendida para atender a qualidade dos requisitos (Figura 4.3). Por meio da propriedade `temCaracteristicaDeQualidade` acoplamos a ontologia de qualidade de software proposta por Duarte (2000). Podem-se citar como exemplo as características citadas por Beaufond (1997): avaliabilidade, completitude, consistência, não-ambiguidade, não redundância, rastreabilidade (seção 4.3).

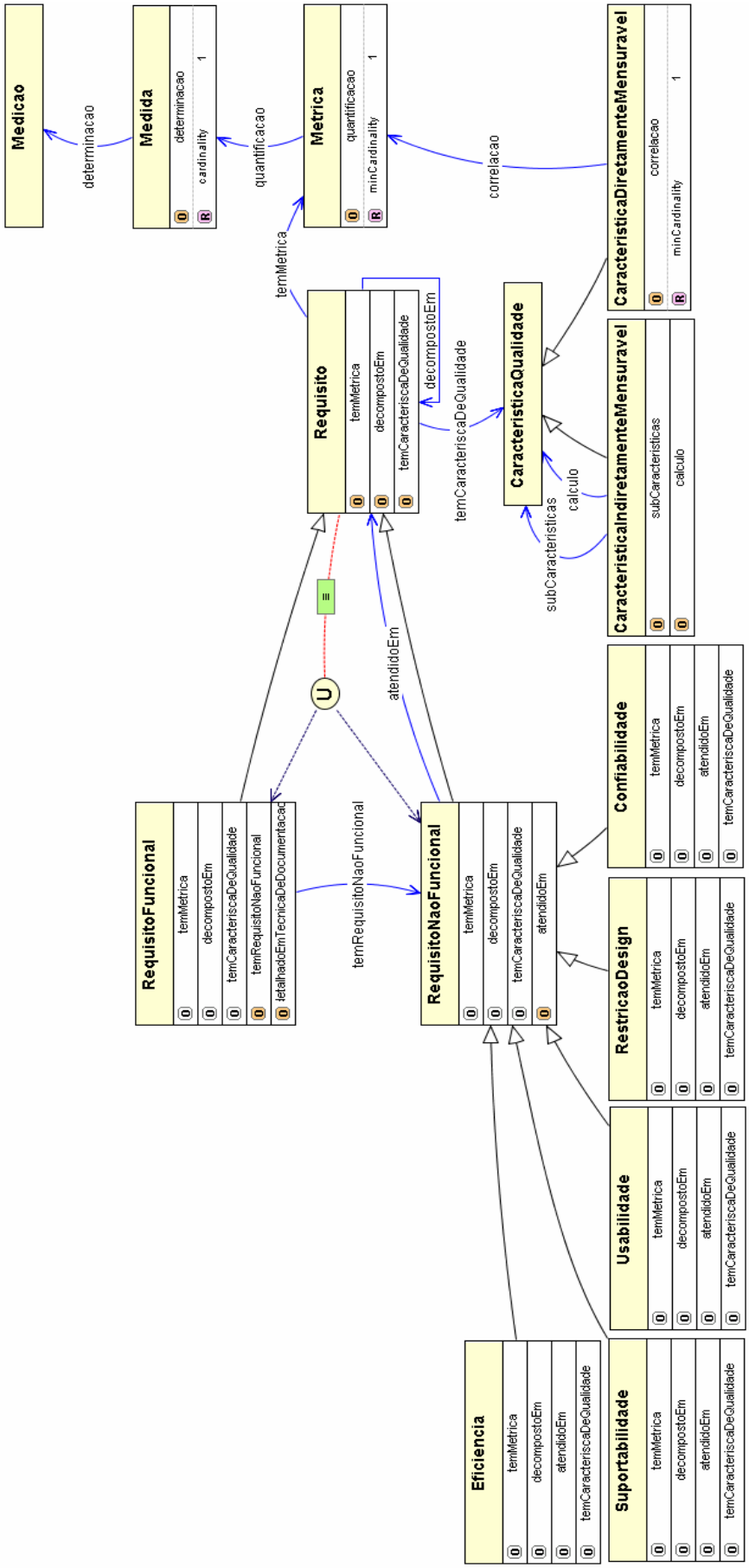

Figura 4.3: Modelo requisitos

Na classe `RequisitoNaoFuncional`, foram modeladas algumas características não funcionais como subclasses. O objetivo dessa decisão, já que elas são muito parecidas, é a herança da propriedade `atendidaEmRequisito`, a facilidade de sua referência em outras classes e um refinamento da característica de rastreabilidade.

O rastreamento dos requisitos não funcionais nem sempre é simples. Um requisito não funcional como desempenho deve ser atendido em vários requisitos funcionais, podendo inclusive ser atendido por um outro requisito não funcional, como, por exemplo, uma restrição física (classe `RestricaoFisica`), que especificaria uma configuração de uma máquina onde o sistema deveria ser executado.

A classe `RequisitoFuncional` possui uma propriedade `temRequisitoNaoFuncional`. Isto se deve ao fato de um requisito funcional poder possuir requisitos não funcionais. Por exemplo, um dado requisito funcional pode exigir um tempo de resposta máximo, que se aplique somente a ele.

A classe `ArtefatoDeDocumentacao` refina os requisitos (Figura 4.4). Ela possui uma propriedade chamada `temRegraDeNegocio`, que é uma instância da classe `RegraDeNegocio`. A `RegraDeNegocio`, além de descrever as regras de negócio, sendo subdividida em `RegraDeNegocioAmbiental` e `RegraDeNegocioOrganizacional`.

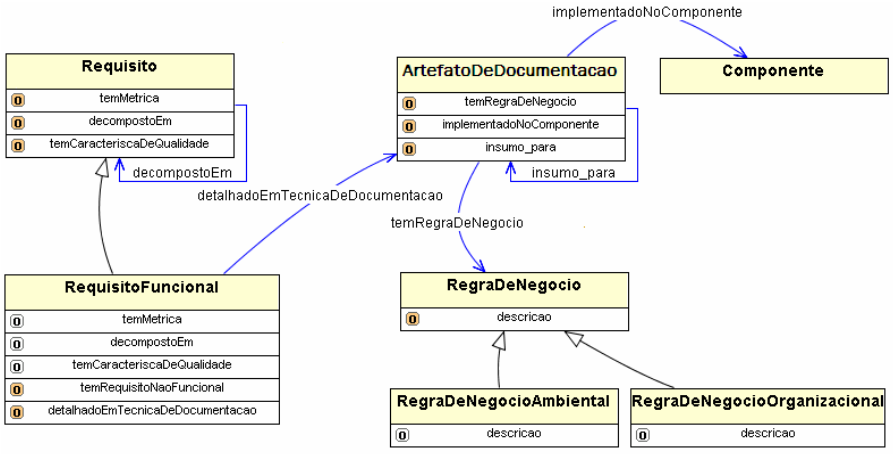

Figura 4.4: Artefato de documentação e componente

Na empresa foi realizado o estudo de caso da ontologia proposta, foi derivada a classe `CasoDeUso` da classe `ArtefadoDeDocumentação`, por ser esse o artefato utilizada nessa empresa.

A classe `Componente` representa a implementação de um requisito em um artefato de documentação. A propriedade `implementadaNoComponente` da classe `ArtefatoDeDocumentacao` relaciona um artefato de documentação à sua implementação. Essa propriedade ajuda na avaliação de impacto, quando um requisito ou um componente necessita ser alterado.

O modelo da ontologia para engenharia de requisitos de software é apresentado na Figura 4.5.

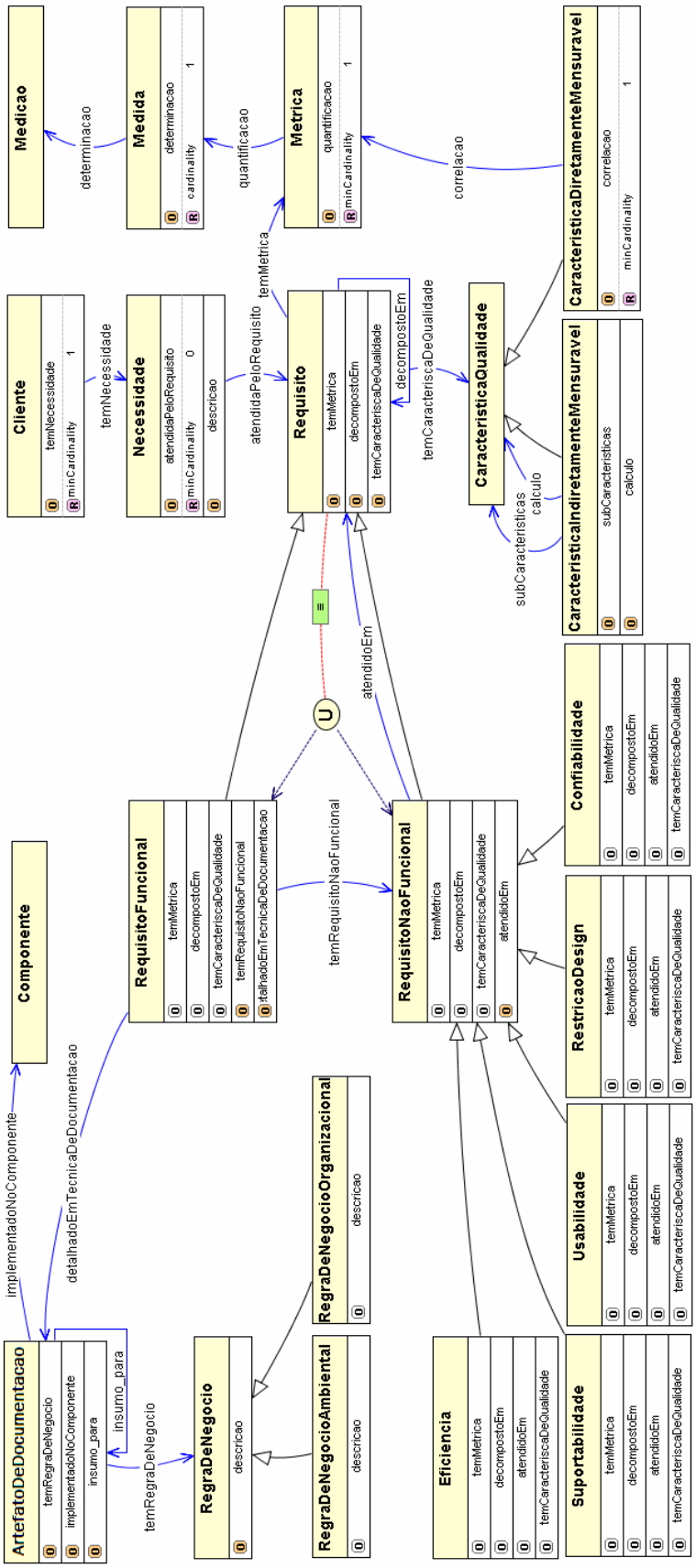

Figura 4.5: Modelo da ontologia para Engenharia de Requisitos

A rastreabilidade de requisitos foi implementada utilizando-se uma superpropriedade chamada `rastreia`. A superpropriedade `rastreia` é transitiva. As propriedades que ligam uma classe à outra classe e que necessitam ser rastreadas são derivadas da propriedade rastreia. A representação gráfica da superpropriedade, assim como a representação gráfica da OWL, ainda não é padronizada na atual versão utilizada neste trabalho.

Para representar graficamente uma superpropriedade foi utilizada a notação descrita na Figura 4.6.

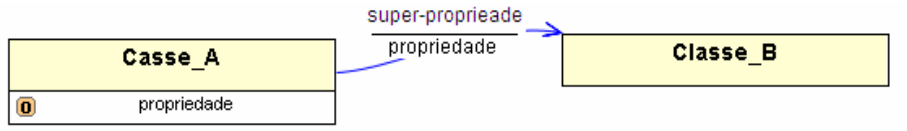

Figura 4.6: Representação gráfica da superpropriedade

Foi incluída a notação de superpropriedade no modelo para as classes que precisavam ser rastreadas, conforme a Figura 4.7.

4.6 Conclusão

Neste capítulo, foi apresentada a proposta de uma ontologia para engenharia de requisitos, utilizando-se a notação da OWL e o modelo gráfico EzOWL (EzOWL, 2004). O modelo apresentado seguiu as questões de competências apresentadas.

No capítulo seguinte, será apresentado um estudo de caso, onde a ontologia proposta é aplicada em uma empresa de desenvolvimento de software.

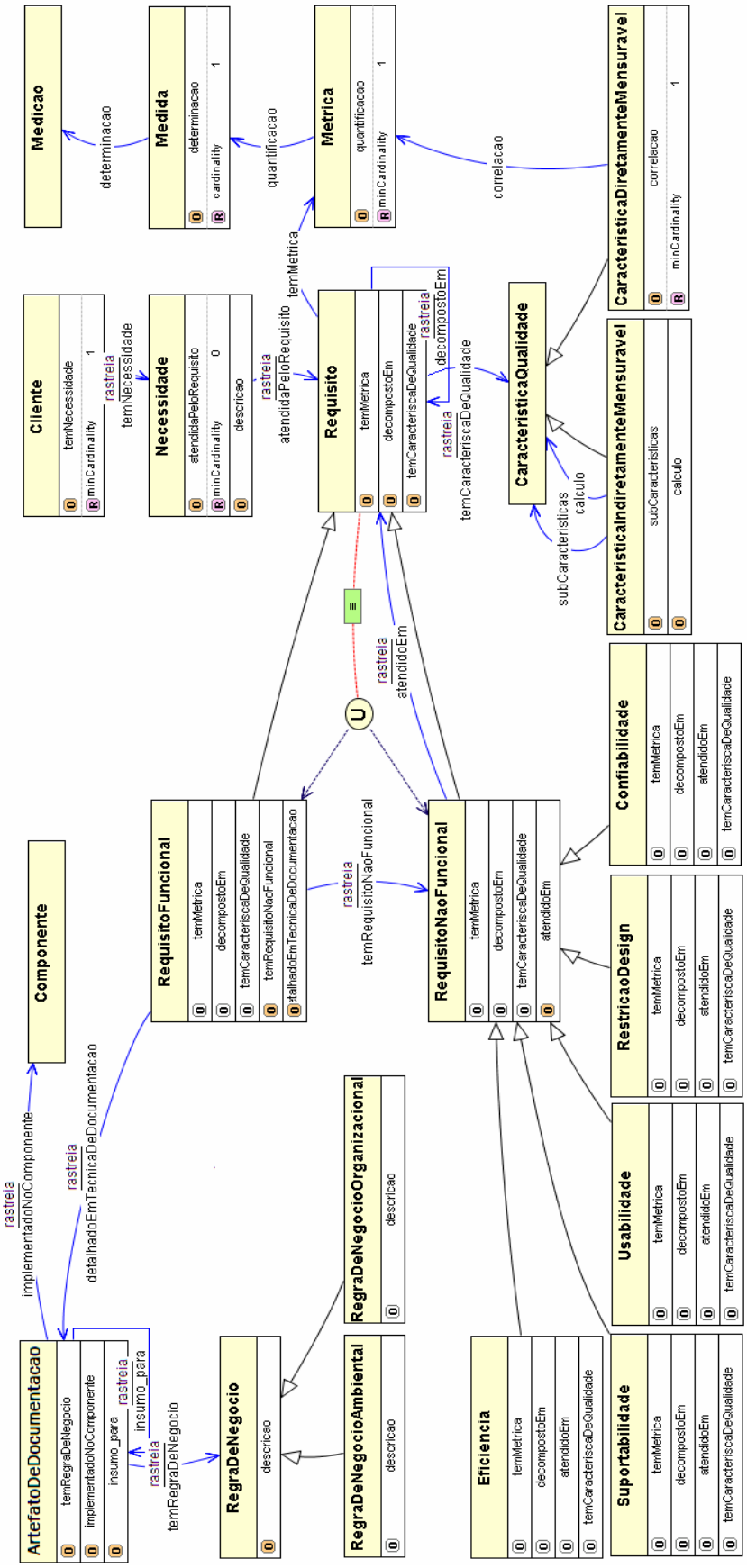

Figura 4.7: Modelo da ontologia para Engenharia de Requisitos com a notação de superpropriedade `rastreia`

CAPÍTULO 5

ESTUDO DE CASO

Neste capítulo, serão apresentados experimentos realizados em uma empresa de software, objetivando validar a ontologia para engenharia de requisitos.

5.1 Aplicação da Ontologia para Engenharia de Requisitos

Para aplicar os conceitos da ontologia para engenharia de requisitos, foi utilizado o *Sistema de Treinamento* de uma Empresa de software brasileira, avaliada como SW-CMM, nível 2. Neste trabalho, foi apresentado apenas um subconjunto (relevante) dos requisitos desse sistema.

As instâncias das classes da ontologia estão apresentadas em tabelas. Para facilitar o entendimento da rastreabilidade dos requisitos desse sistema, foi utilizada a coluna "Item", que corresponde a uma instância de cada `Classe`.

A classe `Cliente` (Tabela 5.1) identifica os clientes do sistema. Na empresa pesquisada, também é documentada a responsabilidade de cada cliente no sistema. A informação "responsabilidade do cliente no sistema" foi disposta abaixo de cada cliente. A propriedade `tem necessidade` identifica as necessidades relacionadas com o cliente, descrito na Tabela 5.2.

Tabela 5.1. Classe `Cliente`

Item	Cliente	Tem Necessidade
C1	**Gestor de Programa de Treinamento (GPT)** Responsabilidades: o Cadastrar treinamentos o Indicar o coordenador de treinamento o Acompanhar treinamentos o Produzir relatórios	N1, N3, N4
C2	**Coordenador de Treinamento** Responsabilidades: o Complementar dados do treinamento o Gerar artefatos de controle	N1, N2, N3, N5, N7
C3	**Instrutor Interno** Responsabilidades: o Ministrar treinamento o Preencher informações do projeto de treinamento o Preencher ementa.	N1, N3
C4	**Revisor** Responsabilidades: o Revisar ementa	N1, N3
C5	**Participante de Treinamento** Responsabilidades: o Preencher pesquisa de efetividade do treinamento	N3

	o Consultar dados dos treinamentos	
C6	**Chefia Imediata** Responsabilidade: o Indicar participantes do treinamento o Complementar pesquisa de efetividade do treinamento o Consultar dados dos treinamentos	N3
C7	**Gerência Sênior** Responsabilidade: o Indicar participantes do treinamento o Consultar dados de treinamento	N1, N3
C8	**Líder de Projeto** Responsabilidade: o Indicar participantes do treinamento o Consultar dados dos treinamentos	N1, N3
C9	**Setor de Recursos Humanos** Responsabilidade: o Consultar dados dos treinamentos	N4

A classe `Necessidade` está descrita na Tabela 5.2, onde é apresentada a propriedade `descrição` e a propriedade `atendida pelo requisito`. A propriedade `atendida pelo requisito` liga uma necessidade a um requisito. O requisito pode ser do tipo "funcional" (Tabela 5.3) ou do tipo "não funcional (Tabela 5.5)".

Tabela 5.2 Classe `Necessidade`

Item	Necessidade	Descrição	Atendida pelo Requisito
N1	Viabilizar treinamento	Realizar operações necessárias para que um treinamento seja implementado.	RF1, RF2, RF3, RF4, RF5
N2	Acompanhar treinamento	Tratar todas as atividades de acompanhamento dos treinamentos.	RF6, RF7, RF8, RF9, RF10, RF11, RF12
N3	Disponibilizar consultas	Possibilitar a realização de consultas.	RF13, RF14, RF15, RF16
N4	Gerar relatórios	Gerar e disponibilizar relatórios periódicos do sistema.	RF17, RF18, RF19, RF20
N5	Conduzir pesquisa de efetividade	Realizar pesquisa de efetividade de cursos.	RF21, RF22, RF23, RF24, RF25
N6	Gerenciar emissão de certificados	Controlar o processo de emissão de certificados.	RF26, RF27, RF28, RF29
N7	Realizar reserva de sala de aula	Controlar as reservas das salas de aula para os treinamentos.	RF30

Na Tabela 5.3, descreve-se a classe `Requisito Funcional`. O atributo "Artefato doc." associa um requisito funcional a seu artefato de documentação (Tabela 5.6). A coluna "Métrica: PF" (Ponto por Função) representa a ligação do atributo `tem métrica` e a instanciação da classe Métrica e Medida. O atributo `tem métrica` é herdado da classe `Requisito`. Os valores da coluna "Métrica: PF" são valores descaracterizados.

Tabela 5.3 Classe `RequisitoFuncional`

Item	Requisito Funcional	Descrição	Artefato Doc.	Métrica PF
RF1	Manter treinamento	Possibilita ao usuário criar, excluir e alterar treinamentos (cursos ou eventos) no sistema.	T17	18
RF2	Manter ementa	Possibilita ao usuário criar, excluir e alterar ementas de treinamentos no sistema.	T12	12
RF3	Manter lista de participantes	Possibilita ao usuário incluir, excluir e alterar participantes de treinamentos.	T14	15
RF4	Manter turmas	Possibilita ao usuário criar, excluir ou alterar turmas de treinamento.	T18	16
RF5	Revisar ementa	Possibilita ao usuário revisar e dar um parecer sobre a ementa de um treinamento.	T12	14
RF6	Gerar listas de presença	Gerar listas de presença de cursos e eventos para impressão.	T6	10
RF7	Gerar avaliação do treinamento pelo instrutor	Gerar avaliação do treinamento pelo instrutor do curso para impressão.	T10	13
RF8	Gerar registro de notas da avaliação de aprendizado	Gerar registro de notas da avaliação de aprendizado para impressão.	T13	12
RF9	Manter presenças	Cadastrar e alterar presenças de cursos e eventos.	T15	13
RF10	Manter avaliação do treinamento pelo instrutor	Cadastrar e alterar resultados da avaliação do treinamento pelo instrutor.	T10	13
RF11	Manter notas da avaliação de aprendizado	Cadastrar e alterar notas da avaliação de aprendizado dos participantes.	T13	15
RF12	Encerrar treinamento	Dar o treinamento como concluído.	T7	12
RF13	Consultar agenda de treinamentos	Permite visualizar a agenda de treinamentos por equipe e ou participante.	T3	10
RF14	Consultar pessoas que não fizeram determinado treinamento	Permite a identificação de participantes que não realizaram determinado treinamento.	T2	9
RF15	Consultar informações de treinamentos	Possibilita a consulta de informações de treinamentos, através refinamentos como visualizar turmas, seus participantes, ementas e projetos adotados nos treinamentos.	T2	10
RF16	Consultar horas de treinamento	Possibilita a consulta de horas gastas com treinamento por pólo, equipe e/ou pessoa.	T2	12
RF17	Gerar relatório de avaliação de treinamento	Possibilita a geração de relatório final de avaliação de treinamento.	T9	8
RF18	Gerar relatório mensal	Possibilita a geração de relatório mensal de treinamentos da organização.	T8	13
RF19	Gerar relatório anual	Possibilita a geração de relatório anual de treinamentos da organização.	T8	12
RF20	Gerar indicadores de resultado	Possibilita a geração de indicadores de resultados dos treinamentos, com base nas informações cadastradas.	T8	17
RF21	Disparar pesquisa de efetividade	Inicia o processo para pesquisa da efetividade do curso.	T5	12

RF22	Registrar informações da pesquisa de efetividade	Permite preencher a pesquisa de efetividade.	T10	13
RF23	Complementar informação da pesquisa de efetividade	Permite completar a pesquisa de efetividade.	T1	13
RF24	Consultar informações de efetividade de treinamentos	Possibilita a consulta de informações de efetividade de treinamentos.	T2	12
RF25	Consultar participantes que responderam à pesquisa de efetividade	Possibilita a consulta dos participantes que responderam à pesquisa de efetividade.	T2	8
RF26	Gerar registro de recebimento de certificado	Permite a geração de uma lista para confirmação de recebimento de certificado pelos participantes de um treinamento.	T4	5
RF27	Gerar solicitação de emissão de certificados	Permite a geração e envio de um documento aos responsáveis pelo sistema, solicitando a emissão de certificados para os participantes de um determinado treinamento.	T4	12
RF28	Cadastrar certificados recebidos	Permite o cadastro dos participantes que já receberam o certificado de um determinado treinamento.	T4	8
RF29	Consultar certificados recebidos/pendente	Possibilita a consulta de participantes e/ou treinamentos com certificados recebidos ou com certificados pendentes.	T4	8
RF30	Reservar sala de aula	Possibilita a reserva de uma sala de aula, para um treinamento, em um determinado período.	T16	10

Os atributos de qualidade para requisitos de software foram selecionados por meio de uma pesquisa de campo, através de um questionário (Apêndice B). Esta pesquisa foi realizada no segundo semestre de 2004 e foi respondida por 17 especialistas em requisitos de software.

Segundo as instruções do questionário (Apêndice B), os especialistas avaliaram a importância dos seguintes atributos para garantir a qualidade de requisitos de software na coluna N1 de acordo com a escala ordinal de 0 a 4 (0 – sem importância; 1 – pouca importância; 2 – desejável; 3 – muito importante; e 4 – imprescindível).

Quando houvesse atributos com a mesma nota, foi solicitado ao especialista que reavaliasse esses atributos na coluna N2, alterando a nota dos mesmos, de tal forma que se caracterizasse que um atributo fosse mais importante que o outro. Por exemplo: os atributos “a”, “b”, e “c” tiveram a nota 4. Na reavaliação, poderiam passar a ter as notas: “a” = 3,9; “b” = 4,0, e “c” = 3,8.

O resultado dessa pesquisa é apresentado na Tabela 5.4. Esse resultado foi obtido da seguinte forma:

- Quando a coluna N2 não foi preenchida pelo especialista, foi atribuído a ela o mesmo valor da coluna N1.

- Foi calculada a partir da coluna N2 a média aritmética das pontuações dadas pelos especialistas a cada atributo de qualidade para requisitos.

Foi apresentado o resultado da pesquisa de qualidade à empresa pesquisada, que selecionou seis atributos de qualidade de requisitos para o projeto deste estudo de caso.

Tabela 5.4 Classificação dos atributos de qualidade de requisitos

Atributos de qualidade	Pontuação	Atributos selecionados pela empresa
Consistência	3,70	X
Completitude	3,56	X
Não Ambigüidade	3,41	X
Rastreabilidade	3,36	X
Manutenibilidade	3,35	
Necessidade	3,07	
Disponibilidade	3,02	
Correção no Uso do Método	3,01	
Simplicidade	2,99	
Uniformidade de terminologia	2,93	
Não Redundância	2,87	X
Avaliabilidade	2,83	X
Uniformidade de Abstração	2,75	
Concisão	1,89	

A classe `RequisitoNaoFuncional` é apresentada na Tabela 5.5 para o sistema deste estudo de caso. A coluna "Tipo de Requisito Não-Funcional" (RNF) indica a subclasse de requisito não funcional. A propriedade "atendida em" indica onde o requisito não funcional foi atendido.

Tabela 5.5. Classe `RequisitoNaoFuncional`

Item	Tipo de Requisito Não-Funcional	Requisito Não-Funcional	Atendido em
RNF1	Usabilidade	O sistema deve conter ajudas em todas as telas.	RF1, RF2, RF3, RF4, RF5, RF6, RF7, RF8, RF9, RF10, RF11, RF12, RF13, RF14, RF15, RF16, RF17, RF18, RF19, RF20, RF21, RF22, RF23, RF24, RF25, RF26, RF27, RF28, RF29, RF30
RNF2	Eficiência	O sistema deve estar disponível nos cinco dias úteis da semana, funcionando 12 horas por dia, com início às 8h e término às 20h.	
RNF3	Restrição física	O sistema deve estar disponível na *Intranet* da empresa.	

A classe Artefato de Documentação, descrita na Tabela 5.6, identifica os artefatos utilizados para documentar um requisito. Na empresa pesquisada, o artefato de documentação utilizado foi o caso de uso.

Tabela 5.6 Classe `ArtefatoDeDocumentacao`

Item	Artefato de Documentação
T1	Caso de Uso: Completar Pesquisa de Efetividade
T2	Caso de Uso: Consultar Treinamento
T3	Caso de Uso: Consultar Agenda
T4	Caso de Uso: Controlar Certificado
T5	Caso de Uso: Disparar Pesquisa de Efetividade aos Participantes
T6	Caso de Uso: Emitir Lista de Presença
T7	Caso de Uso: Encerrar Treinamento
T8	Caso de Uso: Gerar Indicadores
T9	Caso de Uso: Gerar Relatório de Avaliação
T10	Caso de Uso: Manter Avaliação do Treinamento pelo Instrutor
T11	Caso de Uso: Manter Avaliação de Aprendizagem
T12	Caso de Uso: Manter Ementa
T13	Caso de Uso: Manter Notas de Avaliação de Aprendizagem
T14	Caso de Uso: Manter Participante
T15	Caso de Uso: Manter Presença
T16	Caso de Uso: Manter Reserva de Sala
T17	Caso de Uso: Manter Treinamento
T18	Caso de Uso: Manter Turma

Neste sistema, cada regra de negócio está estreitamente relacionada com um artefato de documentação (com cada caso de uso). Cada artefato de documentação está relacionado com apenas uma instância da classe `RegraDeNegocio` e vice-versa. Desta forma, cada regra de negócio possui o mesmo nome do artefato de documentação. Outra característica da regra de negócio na empresa deste estudo de caso foi que o modelo[2] de regra de negócio possuía três grupos de regras:

- *Regras de apresentação*: regras responsáveis pela formatação e apresentação dos dados para os clientes (chamados de atores na técnica de caso de uso), em um caso de uso. Exemplos:
  - Um dado do tipo "data" deverá ser apresentado com no formato "dd/mm/aaaa", onde:
    - "dd", representa do dia do mês;
    - "mm", representa o mês e;
    - "aaaa", representa o ano.
  - Um dado do tipo "unidade federativa" deverá apresentar uma lista com as Unidades federativas para que o ator possa selecionar.

[2] Documento padrão utilizado para este fim.

- *Regras de validação*: regras responsáveis pela validação dos dados informados pelos clientes em um caso de uso. Exemplo:
    - Um dado do tipo "data" deverá pertencer ao calendário civil para que seja considerado válido.
    - Um dado do tipo "unidade federativa" deverá conter uma das unidades federativas do Brasil para ser considerado válido.
- *Regras gerais*: regras que não se enquadram em nenhum dos dois tipos de regras acima (apresentação ou validação). Exemplo:
    - Regra para cálculo de tempo total de treinamento de um cliente do tipo "participante interno".

O sistema foi implementado utilizando, para cada "caso de uso", o padrão de navegação de uma tela de pesquisa que depois de executada leva a uma tela de resultados. Essa transição sempre feita por uma classe Java controlando a interação, conforme apresentado na Figura 5.1. Assim sendo, cada caso de uso possui três componentes relacionados a ele: uma página de pesquisa, uma página de resultado e uma classe Java que controla a iteração.

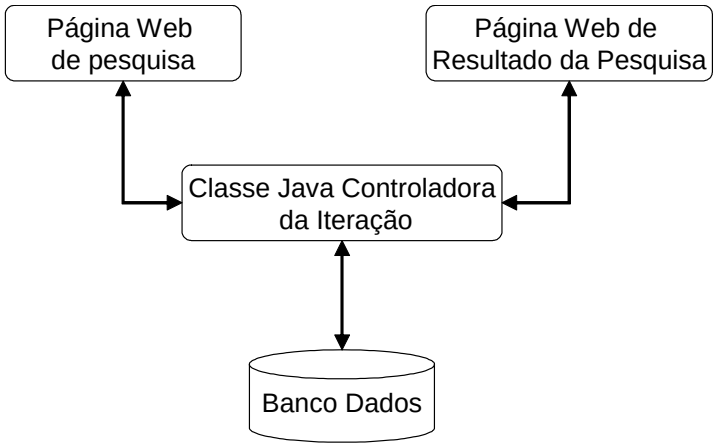

Figura 5.1: Padrão de implementação para os casos de uso

Neste sistema, temos, por exemplo:

- `ArtefatoDeDocumentacao`: "Caso de Uso: Completar Pesquisa de Efetividade" é implementado nas classes:
    - `Componente`: "PesquisaEfetividade.jsp";
    - `Componente`: "PesquisaEfetividadeResultadoPesquisa.jsp";
    - `Componente`: "PesquisaEfetividadeControle.java";
- `ArtefatoDeDocumentacao`: "Caso de Uso: Consultar Treinamento" é implementado nas classes:

- `Componente`: “ConsultarTreinamento.jsp”
- `Componente`: “ConsultarTreinamentoResultado.jsp”
- `Componente`: “ConsultarTreinamentoControle.java”

5.2 Rastreabilidade no Modelo

A rastreabilidade auxilia tanto aos técnicos a gerência do projeto. Na ontologia proposta, a rastreabilidade foi tratada com a utilização de uma superpropriedade chamada `rastreia`, que é transitiva. As propriedades que ligam uma classe à outra classe e que precisão ser rastreadas são derivadas da propriedade `rastreia`. Como definido no capítulo 4, as propriedades derivadas da propriedade `rastreia` são:

- `temNecessidade`
- `atendidaPeloRequisito`
- `decompostoEm`
- `atendidoEm`
- `temRequisitoNaoFuncional`
- `detalhadoEmArtefatoDeDocumentação`
- `temRegraDeNegocio`
- `implementadoNoComponente`

Para este estudo de caso foi utilizado o servidor *Racer* (*Renamed ABox and Concept Expression Reasoner*) *Professional*. O *Racer* é um servidor para Web semântica, de inferência e prova de teoremas. Esse servidor pode ser utilizado para gerenciar Web semântica baseada em OWL, isto é, pode ser usado com provador de teoremas para editores como o Protégé. O *Racer* foi utilizado em conjunto com o Protégé para a descoberta da rastreabilidade dos requisitos do sistema deste estudo de caso.

Primeiramente, transferiu-se a ontologia proposta para o servidor *Racer*. A transferência foi realizada utilizando-se os itens de *menu* “*Check Consistency*”, “*Classify taxonomy*”, “*Compute inferred types*” do menu OWL do Protégé (Figura 5.2).

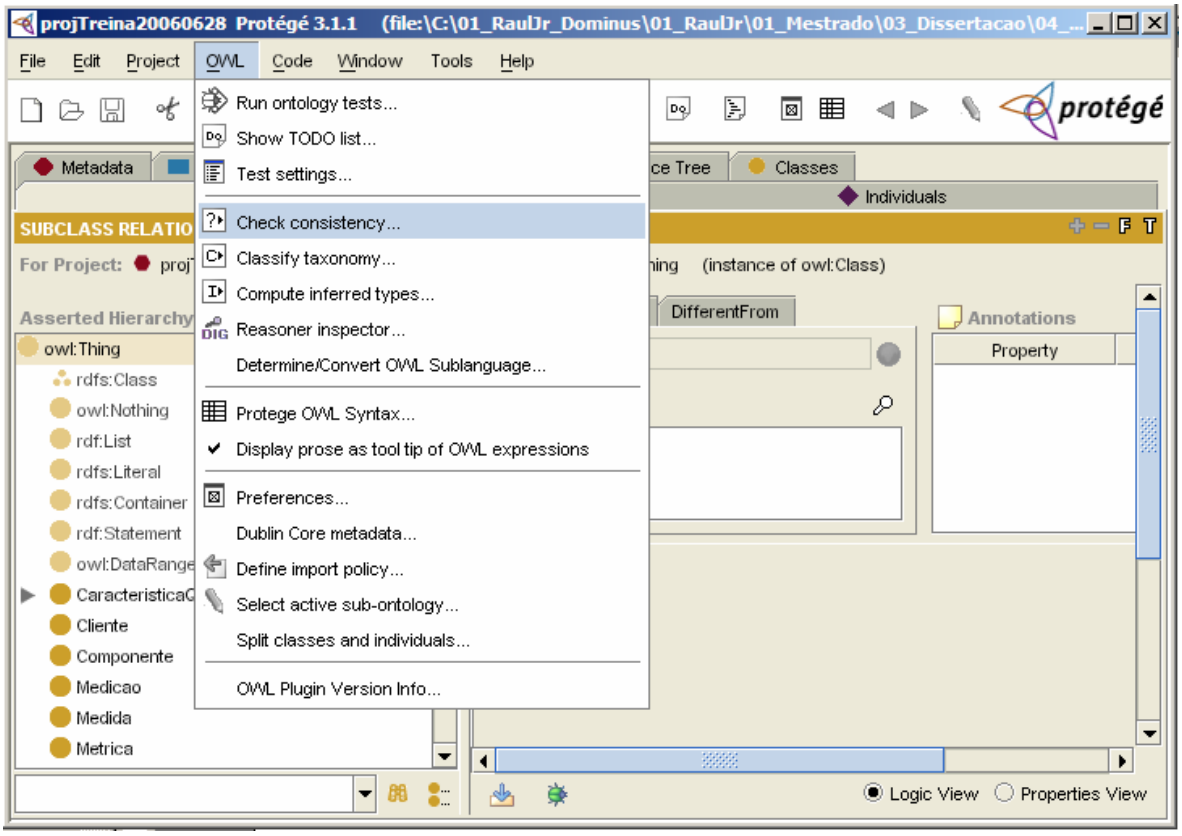

Figura 5.2: Conexão do Protégé com o Racer

Após a transferência da ontologia, utilizou-se o *RacerPorter* para executar as consultas da ontologia (Figura 5.3). Foi utilizada a linguagem de pesquisa RQL (*Racer Query Language*).

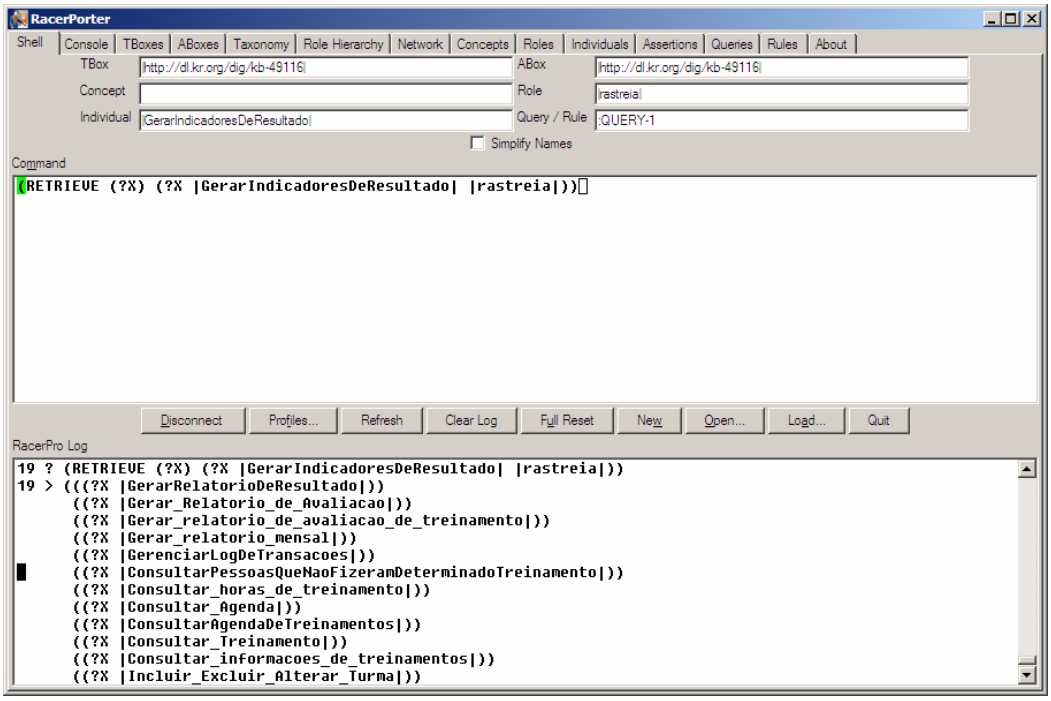

Figura 5.3: *RacerPorter*

Por exemplo, executando-se a consulta abaixo, obtém-se o resultado apresentado na Tabela 5.7.

```
(RETRIEVE (?X) (?X |GerarIndicadoresDeResultado| |rastreia|))
```

Tabela 5.7 Resultado consulta

Classe Correspondente	Item	Linha de Resultado
Artefato de Documentação	T2	((?X |ConsultarTreinamento|))
	T3	((?X |ConsultarAgenda|))
	T7	((?X |EmitirListaDePresenca|))
	T9	((?X |GerarRelatorioDeAvaliacao|))
	T10	((?X |ManterAvaliacaoDoTreinamentoPelo Instrutor|))
	T11	((?X |ManterAvaliacaoDeAprendizagem|))
	T12	((?X |ManterEmenta|))
	T13	((?X |ManterNotasDe AvaliacaoDeAprendizagem|))
	T15	((?X |ManterPresenca|))
	T17	((?X |ManterTreinamento|))
	T18	((?X |ManterTurma|))
Cliente	C1	((?X |GestorDeProgramaDeTreinamento|))
	C2	((?X |CoordenadorDeTreinamento|))
	C3	((?X |InstrutorInterno|))
	C4	((?X |Revisor|))
	C5	((?X |ParticipanteDeTreinamento|))
	C6	((?X |ChefiaImediata|))
	C7	((?X |GerenciaSenior|))
	C8	((?X |LíderDeProjeto|))
	C9	((?X |Setor_Recursos_Humanos|))
Necessidade	N1	((?X |ViabilizarTreinamento|))
	N2	((?X |AcompanharTreinamento|))
	N3	((?X |DisponibilizarConsultas|))
	N4	((?X |GerarRelatorios|))
	N5	((?X |ConduzirPesquisaDeEfetividade|))
	N6	((?X |GerenciarEmissaoDeCertificados|))
	N7	((?X |RealizarReservaDeSalaDeAula|))
Requisito Funcional	RF1	((?X |ManterTreinamento|))
	RF2	((?X |ManterEmenta|))
	RF3	((?X |ManterListaDeParticipantes|))
	RF4	((?X |ManterTurmas|))
	RF5	((?X |RevisarEmenta|))
	RF6	((?X |GerarListasDePresenca|))
	RF7	((?X |GerarAvaliacaoDoTreinamentoPeloInstrutor|))
	RF8	((?X |GerarRegistroDeNotasDaAvaliacaoDeAprendizado|))
	RF9	((?X |ManterPresencas|))
	RF10	((?X |ManterAvaliacaoDoTreinamentoPeloInstrutor|))
	RF11	((?X |ManterNotasDaAvaliacaoDeAprendizado|))
	RF12	((?X |EncerrarTreinamento|))
	RF13	((?X |ConsultarAgendaDeTreinamentos|))
	RF14	((?X |ConsultarPessoasQueNaoFizeramDeterminado Treinamento|))
	RF15	((?X |Consultar_informacoes_de_treinamentos|))
	RF16	((?X |ConsultarHorasDeTreinamento|))
	RF17	((?X |GerarRelatorioDeAvaliacaoDeTreinamento|))

	RF18	((?X |GerarRelatorioMensal|))
	RF19	((?X |GerarRelatorioAnual|))
	RF20	((?X |GerarIndicadoresDeResultado|)))
	RF21	((?X |DispararPesquisaDeEfetividade|))
	RF22	((?X |RegistrarInformacoesDaPesquisaDeEfetividade|))
	RF23	((?X |CompletarInformacaoDaPesquisaDeEfetividade|))
	RF24	((?X |ConsultarInformacoesDeEfetividadeDe Treinamentos|))
	RF25	((?X |ConsultarParticipantesQueResponderamAPesquisaDe Efetividade|))

Um resultado mais detalhado pode ser obtido através da aba *Network* do *RacerPorter*, que gera grafos e árvores (com profundidade personalizada) de uma ontologia. A árvore apresentada na Figura 5.4 pode ser usada para seguir a rastreabilidade na ordem direta como na ordem inversa. Na ordem direta, têm-se as arestas (formadas pelas propriedades das classes): "`detalhadoEmTecnicaDeDocumentacao`" e "`temRegraDeNegocio`". Na ordem inversa, têm-se as arestas: "`atendidaPeloRequisito`" e "`temNecessidade`".

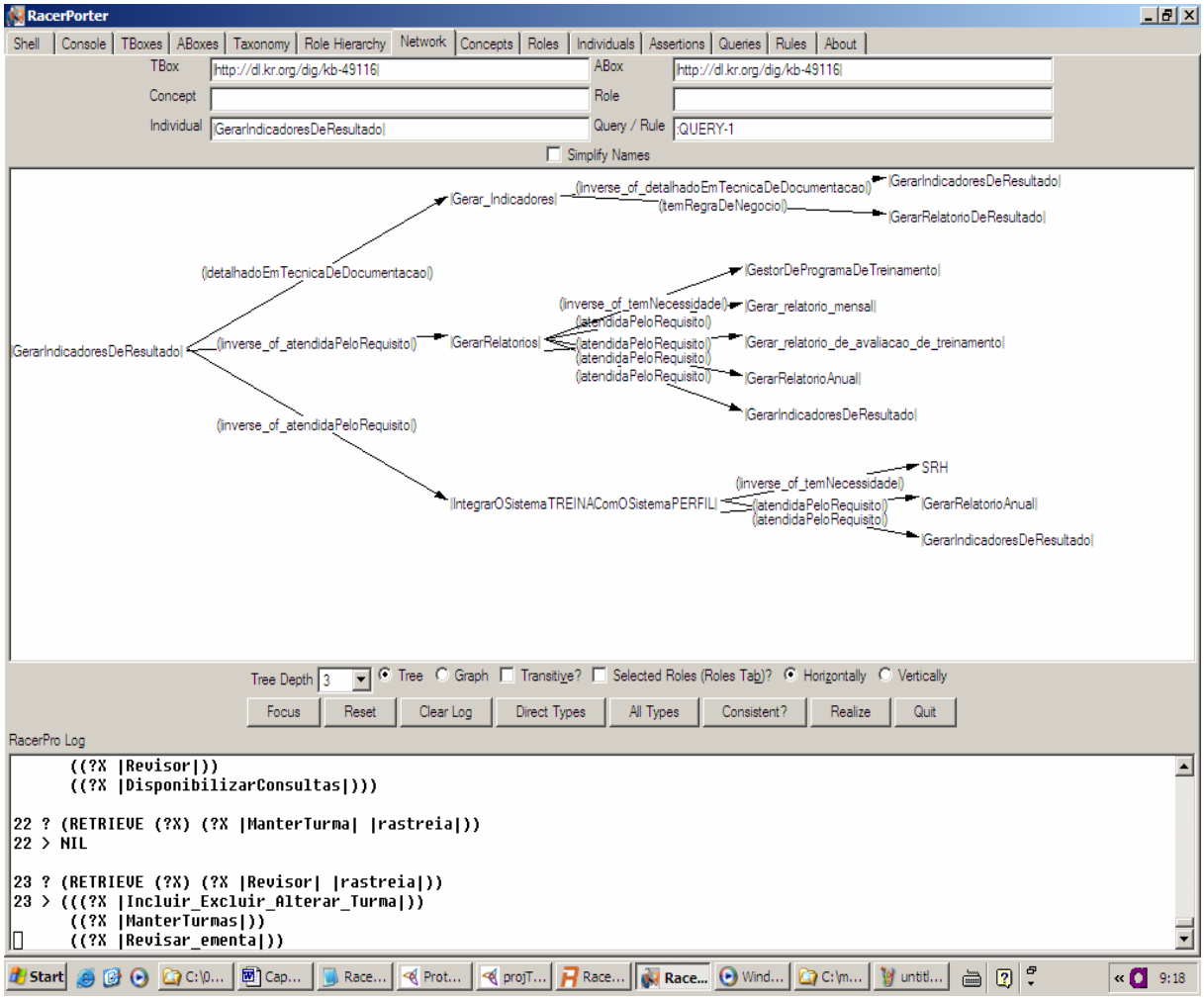

Figura 5.4: Árvore de rastreabilidade de 3 níveis

Com o uso destas ferramentas pode se fazer uma melhor gerencia dos requisitos do sistema.

5.3 Análise dos Resultados

A ontologia proposta foi aplicada a mais três sistemas (Tabela 5.8), além do sistema de treinamentos apresentado na seção anterior. O Sistema *A* e o Sistema *B* são do domínio de aplicação financeira, e Sistema *C* é da área comercial. O Sistema *A* era um sistema legado; o Sistema *B* era um sistema em desenvolvimento; e o Sistema *C* era um sistema que vinha sofrendo freqüentes manutenções.

Tabela 5.8. Características dos Sistemas em que a Ontologia foi Aplicada

Classe / Instância de Classes	Quantidade		
	Sistema *A*	**Sistema *B***	**Sistema *C***
Necessidades	08	05	06
Funcionalidades	26	15	30
Casos de Uso	94	30	86
Informações Ambientais	10	05	04
Informações Organizacionais	05	07	03
Regra de Negócio	30	12	35
Componentes	172	85	226

Com a aplicação da ontologia proposta nos sistemas deste estudo de caso, percebeu-se que quando ocorre uma mudança de requisitos nesses sistemas, pode-se facilmente percorrer o modelo e identificar os requisitos e componentes impactados e suas respectivas métricas, para se planejamento melhor mudanças no escopo do projeto de software.

Em um dos sistemas deste estudo de caso, por exemplo, havia uma regra de negócio de informação ambiental, que era o cálculo da CPMF (Contribuição Provisória sobre Movimentação Financeira). Caso a lei do CPMF seja alterada, pode-se percorrer o modelo procurando os casos de uso que utilizam essa regra de negócio. Em seguida, podem-se procurar os componentes associados a esses casos de uso e suas métricas (ponto de função). Assim sendo, pode-se mensurar o tempo necessário para a adequação do sistema à nova lei.

Os requisitos desses sistemas tiveram sua qualidade melhorada, pois cada um desses requisitos possui atributos de qualidade explícitos. Esses requisitos são revisados de acordo com marcos previstos no processo de desenvolvimento.

Podem-se destacar os seguintes resultados:

- A aplicação da ontologia melhorou o entendimento dos requisitos como um todo, tanto pelo cliente quanto pelos desenvolvedores.
- Houve uma maior clareza das transformações do requisito no processo de construção do software.
- Incrementou-se o poder de rastreabilidade no sistema, permitindo uma melhor análise de impacto em mudanças de requisitos.

- O acompanhamento de métricas e qualidade dos requisitos foi melhorado.

5.4 Conclusão

Neste capítulo, foi apresentada uma aplicação da ontologia para engenharia de requisitos proposta. Também foram abordados os resultados da aplicação dessa ontologia em quatro sistemas de uma empresa de software.

O próximo capítulo tratará das conclusões deste trabalho.

Capítulo 6

CONCLUSÃO

Este capítulo apresenta as principais conclusões deste trabalho, cujo objetivo foi propor uma ontologia para engenharia de requisitos. Estão descritas suas contribuições e sugestões para trabalhos futuros.

Uma ontologia é essencialmente um acordo, e esse acordo não necessariamente precisa abranger toda a conceituação de um determinado domínio. O acordo pode abranger apenas parte dele, ou seja, a ontologia pode oferecer uma visão para o domínio. Assim, uma ontologia atua como um contrato entre parceiros, permitindo que se comuniquem com segurança dentro do domínio de informação (DAUM & MERTEN, 2002).

Este trabalho apresentou uma ontologia para engenharia de requisitos de software, em que foram definidas classes e propriedades consideradas relevantes para o levantamento e o gerenciamento de requisitos de software. Para o desenvolvimento da ontologia proposta, utilizou-se a metodologia de Noy e McGuiness (2001). Essa ontologia foi elaborada a partir da OWL (W3C, 2004), utilizando-se conceitos de classes e propriedades.

A elaboração de uma ontologia para engenharia de requisitos de software objetivou auxiliar em um melhor entendimento dos requisitos de software entre as principais partes envolvidas no processo de desenvolvimento: clientes e desenvolvedores.

A ontologia proposta também foi aplicada a quatro sistemas em uma empresa de desenvolvimento de software brasileira, avaliada como SW-CMM nível 2. Com a aplicação da ontologia proposta percebeu-se que quando ocorre uma mudança de requisitos nesses sistemas, pode-se facilmente percorrer o modelo da ontologia e identificar os requisitos e componentes impactados e suas respectivas métricas, para planejamento das alterações.

Nos sistemas avaliados, pudemos também perceber uma melhora na qualidade de seus requisitos. Todos os requisitos desses sistemas passaram a ter atributos de qualidade explicitamente apresentados. Os atributos foram alimentados e revisados ao longo das etapas do processo de desenvolvimento / manutenção de software.

Podem-se destacar como principais contribuições deste trabalho:

- A elaboração de uma ontologia para a engenharia de requisitos.
- A aplicação da ontologia em quatro sistemas favoreceu um melhor entendimento e gerenciamento dos requisitos levantados nesses sistemas, tanto pelo cliente quanto pelos desenvolvedores.

- Houve uma maior clareza das transformações dos requisitos no processo de construção do software.
- Incrementou-se o poder de rastreabilidade no sistema, permitindo uma melhor análise de impacto em mudanças de requisitos.
- O acompanhamento de métricas e qualidade de requisitos de software para os sistemas pesquisados neste trabalho foi facilitado e simplificado.

6.1 Trabalhos Futuros

Dentre os trabalhos futuros que podem dar continuidade à pesquisa aqui apresentada, sugerem-se:

- Utilização da ontologia proposta como forma de automatizar o processo descoberta de inconsistências de requisitos.
- Utilização deste trabalho em outras empresas como objetivo de consolidar e aprimorar os conceitos propostos.
- Comparar os resultados obtidos com outras ontologias em requisitos de software.

Referências Bibliográficas

ANTONIOU, GRIGORIS; VAN HARMELEN, FRANK. *A Semantic Web Primer.* Massachusetts: TLFeBOOK, 2004.

BEAUFOND, CLIFTON E. C. *Avaliação da qualidade de especificações orientadas a objetos.* Tese (Doutorado em 1997) – Coordenação dos Programas de Pós-graduação em Engenharia. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1997.

BOEHM, B. W.; PAPACCION, P N. *Understanding and Controlling Software Costs.* IEEE Transaction on Software Engineering, pp. 1462-1477, October 1988.

BREITMAN, KARIN K.; LEITE, JÚLIO C. S. P. *Ontology as a Requirements Engineering Product.* 11th IEEE International Requirements Engineering Conference, 2003.

CMMI, CMMI for Development, V1.2 model. CMU/SEI-2006-TR-008. Software Engineering Institute, Carnegie Mellon. Pittsburgh, 2006.

DAUM, BERTHOLD; MERTEN. Udo, *Arquitetura de sistemas com XML.* Conteúdo, processo e apresentação. Rio de Janeiro. Editora Campus, 2002.

DUARTE, KÁTIA C.; FALBO, RICARDO A. *Uma ontologia de qualidade de software.* In: Workshop de Qualidade de Software, 7. SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 14. João Pessoa, 2000.

EZOWL. *Editor visual para OWL no Protege-2000*. Disponível em: <http://iweb.etri.re.kr/ezOWL/>. Acesso em Maio de 2004.

FALBO, RICARDO DE ALMEIDA. *Integração de Conhecimento em um Ambiente de Desenvolvimento de Software.* Tese de Doutorado UFRJ 1998.

FENSEL, D. *Ontologic: A Silver Bullet For Knowledge Management And Eletronic Commerce*, Springer 2001.

FENTON, N E, PFLEEGER, S L. *Software Metrics: A Rigorous and Practical Approach.* 2.ed. Londres: PWS Publishing, 1997.

FOX, M S, ET AL. *A Common-Sense Model of the Enterprise.* In: Proceedings of the 2nd Industrial Engineering Research Conference, 1993

GAVA, T B S; MENEZES C S. *Uma ontologia de domínio para a aprendizagem cooperativa*. In: Simpósio Brasileiro de Informática na Educação, 14., 2003, Rio de Janeiro. Rio de Janeiro: SBIE, 2003. Disponível em: <http://www.nce.ufrj.br/sbie2003/publicacoes/paper36.pdf>. Acesso em Setembro de 2005.

GOTEL, O C Z; FINKELSTEIN, A C W. *An Analysis of the Requirements Traceability Problem.* Proceeding of ICRE94, 1st International Conference on Requirements Engineering. 1994; Colorado Springs, Co; IEEE CS Press, 1994.

GRUBER, THOMAS R. *A Translation Approach to Portable Ontology Specifications*, *Knowledge Acquisiton* – 5: 199:200, 1993.

GRUBER, THOMAS R. *Towards Principles for the Desingn of Ontologies Used for Knowledge Sharing*. Int. J. Human-Computer Studies, v. 43, n. 5/6. 1995.

GRÜNINGER, M., FOX, M.S. *The Design and Evaluation of Ontologies for Enterprise Engineering. Technical Report*, University of Toronto. May, 1994.

GRÜNINGER, MICHAEL, FOX, MARK S. *Methodology for Design and Evaluation of Ontologies*, Proceedings of Workshop on basic Ontological Issues in Knowledge Sharing. Canada, 1995.

GRÜNINGER, MICHAEL; Lee, Jintae. *Ontology Application and Design.* Communication ACM, February, 2002.

GUARINO, NICOLA. *Formal Ontology and Information Systems.* FOIS'98, June 1998.

IEEE Computer Society. ANSI/IEEE Standard 830-1984, 1984.

ISO. NBR ISO 9001. ISO 9001. *Sistemas de Gestão de Qualidade – Requisitos*. Associação Brasileira de Normas Técnicas – ABNT. Rio de Janeiro, 2000.

JACOBSON L, CHRISTERON M ET AL. *Object Oriented Software Engineering*. Wokingham: Addison Wesley. (Cap. 5, 6 e 12). 1993.

LEITE, J C S P; DOORN, J H. *Perspectives on Software Requirements*. Capítulo 1: "Perspectives on Software Requirements: An Introduction". Kluwer Academic Press, ISBN: 1-4020-7625-8, 2003.

LIMA, KARINA VILLELA DE CARVALHO. *Definição e Construção de Ambientes de Desenvolvimento de Software Orientados a Organização*. Tese de Doutorado – Coordenação dos Programas de Pós-graduação em Engenharia, Universidade Federal do Rio de Janeiro, (pp 116 e 117). Rio de Janeiro, 2004.

LÓPEZ, F. *Overview of methodologies for building ontologies*. In: Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends. CEUR Publications, 1999. Intelligent Systems, 16(1):26-34, 2001.

MARTIN, MARIA A.; OLSINA, LUÍS. *Towards an ontology for software metrics and indicators as the Foundation for a Cataloging Web System*. In: FIRST LATIN AMERICAN WEB CONGRESS, 2003.

NOY, NATALYA F.; MCGUINNESS, DEBORAH L. *Ontology Devepment 1001: A Guide to Creating Your First Ontology*; KSL Technical Report, Standford University, 2001.

NOY, NATALYA. F.; SINTEK, MICHAEL; DECKER, STEFAN; CRUBEZY, MONICA; FERGERSON, RAY W.; MUSEN, MARK A. *Creating Semantic Web Contents with Protege-2000*. IEEE Intelligent Systems 16(2): 60-71, 2001.

OWL W3C; 2004, *Ontology Web Language*. Disponível em: <http://www.w3c.org/TR/2004/REC-owl-features-20040210>. Acesso em Julho de 2004.

PFLEEGER, S L. *Software Engineering: Theory and Practice*. 2.ed. New Jersey: Prentice Hall, ISBN 0-13-029049-1. pp135 – 184, 2001.

PINHEIRO, F A C. *Perspectives on Software Requirements*; Chapter 5: "Requirements Traceability". Kluwer Academic Press, ISBN: 1-4020-7625-8; 2003.

RACER - RENAMED ABOX AND CONCEPT EXPRESSION REASONER PROFESSIONAL, version 1.9.

RAMESH B. *Fators Influencing Requirements Traceability Practice*. Communication of ACM, 41(12):37-44, 1998.

RDFS W3c; 2004; *RDF Vocabulary Description Language 1.0: RDF Schema*. Disponível em: <http://www.w3.org/TR/rdf-schema/>. Acesso em Julho de 2004.

ROBERTSON, S, ROBERTSON J. *Mastering the Requirements Process*. Reading, MA: Addison Wesley, 1999.

RATIONAL UNIFIED PROCESS Tutorial. Version 2003 06 00

SOMMERVILLE, IAN. *Engenharia de software*. 6.ed. Tradução: Maurício de Andrade. São Paulo: Addison Wesley, 2003.

TORANZO, MARCO; CASTRO, JAELSON F. B.; MELLO, ELTON. *Uma proposta para melhorar o rastreamento de requisitos*. WER, 2002.

W3C. *Ontology web language*. Disponível em: <http://www.w3c.org/TR/2004/REC-owl-features-20040210>. Acesso em Julho de 2006.

Wikikipedia. Enciclopédia da Internet. Disponível em: <http://en.wikipedia.org/wiki/XML_namespace> . Acesso em Julho de 2006.

Apêndice A

Códigos OWL da Ontologia

A.1. Introdução

Neste apêndice mostramos os códigos OWL gerados na construção da ontologia para engenharia de requisitos de software.

A.2. Código OWL de Medição da Qualidade

```
<owl:Class rdf:ID="Medida">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
       1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#determinacao"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="CaracteristicaDeQualidade"/>

<owl:Class rdf:ID="Metrica">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
      1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#quantificacao"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="CaracteristicaIndiretamenteMensuravel">
  <rdfs:subClassOf rdf:resource="#CaracteristicaDeQualidade"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
      1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#subcaracteristicas"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#calculo"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="CaracteristicaDiretamenteMensuravel">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#correlacao"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
      1</owl:cardinality>
```

```
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#CaracteristicaDeQualidade"/>
</owl:Class>

<owl:Class rdf:ID="Medicao"/>

<owl:ObjectProperty rdf:ID="correlacao">
  <rdfs:domain rdf:resource="#CaracteristicaDiretamenteMensuravel"/>
  <rdfs:range rdf:resource="#Metrica"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="subcaracteristicas">
  <rdfs:domain rdf:resource="#CaracteristicaIndiretamenteMensuravel"/>
  <rdfs:range rdf:resource="#CaracteristicaDeQualidade"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="calculo">
  <rdfs:domain rdf:resource="#CaracteristicaIndiretamenteMensuravel"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="quantificacao">
  <rdfs:range rdf:resource="#Medida"/>
  <rdfs:domain rdf:resource="#Metrica"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="determinacao">
  <rdfs:range rdf:resource="#Medicao"/>
  <rdfs:domain rdf:resource="#Medida"/>
</owl:ObjectProperty>
```

A.3. Código OWL – Modelo para Requisito do Cliente

```
<owl:Class rdf:ID="Cliente">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#temNecessidade"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
       1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Necessidade">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#atendidaPeloRequisito"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
      0</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Requisito">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#RequisitoFuncional"/>
        <owl:Class rdf:about="#RequisitoNaoFuncional"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:ID="RequisitoNaoFuncional">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Requisito"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="RequisitoFuncional">
  <rdfs:subClassOf rdf:resource="#Requisito"/>
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID="decompostoEm">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Requisito"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#Requisito"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="atendidaPeloRequisito">
  <rdfs:range rdf:resource="#Requisito"/>
  <rdfs:domain rdf:resource="#Necessidade"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="temNecessidade">
  <rdfs:domain rdf:resource="#Cliente"/>
  <rdfs:range rdf:resource="#Necessidade"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="descricao">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Necessidade"/>
</owl:DatatypeProperty>
```

A.4. Modelo de Requisito

```
<owl:Class rdf:ID="Eficiencia">
  <rdfs:subClassOf rdf:resource="#RequisitoNaoFuncional"/>
</owl:Class>

<owl:Class rdf:ID="Confiabilidade">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#RequisitoNaoFuncional"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Suportabilidade">
  <rdfs:subClassOf rdf:resource="#RequisitoNaoFuncional"/>
</owl:Class>

<owl:Class rdf:ID="RestricaoDesign">
  <rdfs:subClassOf rdf:resource="#RequisitoNaoFuncional"/>
</owl:Class>

<owl:Class rdf:ID="Usabilidade">
  <rdfs:subClassOf rdf:resource="#RequisitoNaoFuncional"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="correlacao">
  <rdfs:domain rdf:resource="#CaracteristicaDiretamenteMensuravel"/>
  <rdfs:range rdf:resource="#Metrica"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="temRequisitoNaoFuncional">
  <rdfs:range rdf:resource="#RequisitoNaoFuncional"/>
  <rdfs:domain rdf:resource="#RequisitoFuncional"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="temMetrica">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Requisito"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#Metrica"/>
</owl:ObjectProperty>
```

Apêndice B

Questionário sobre Qualidade de Requisitos de Software

B.1. Introdução

Este apêndice apresenta o questionário aplicado a especialistas em requisitos de software, para a seleção de atributos de qualidade de requisitos a serem incluídos na ontologia para engenharia de requisitos de software.

B.2. Questionário

B.2.1. Instrumentação

Este questionário visa identificar atributos de qualidade para requisitos de software. Para isto, estará disponível uma lista de atributos, onde seu grau de importância deverá ser avaliado. Novos atributos poderão ser sugeridos. A pesquisa faz parte desta dissertação.

B.2.2. Caracterização do Especialista

Marque as opções que se aplicam (podem ser marcadas mais de uma opção para um mesmo item):

Nome:		**E-mail**:	
Área de Atuação			
Empresa		*Universidade*	
	Gerente de Projeto		Professor
	Analista de Sistemas		Pesquisador
	Arquiteto		Aluno de Doutorado
	Desenvolvedor		Aluno de Mestrado
	Gerente de Qualidade		Aluno de Graduação
	Outro		Consultor
Formação: *Nível e Área*			
	Doutorado ( ) Eng de Software ( ) Computação / Informática ( ) Outro		
	Mestrado ( ) Eng de Software ( ) Computação / Informática ( ) Outro		
	Especialização ( ) Eng de Software ( ) Computação / Informática ( ) Outro		
	Graduação ( ) Eng de Software ( ) Computação / Informática ( ) Outro		
	Tempo de Atuação na área (em anos)		

B.2.3. Questionário sobre Qualidade de Requisitos de Software

Instruções:

- Avalie em **N1** de acordo com a escala ordinal de **0 a 4** (0 – sem importância; 1 – pouca importância; 2 – desejável; 3 – muito importante; e 4 – imprescindível), o valor

que lhe parece melhor representar a importância dos seguintes atributos para garantir a qualidade de requisitos de software.

- Quando houver atributos com a mesma nota, reavalie esses atributos em **N2**, alterando a nota dos mesmos, de tal forma que se caracterize que um seja mais importante que o outro. Por exemplo: os atributos "a", "b", e "c" tiveram a nota 4. Na reavaliação, por exemplo, passaram a ter as notas: "a" = 3,9; "b" = 4,0, e "c" = 3,8

Atributo / descrição	N1	N2
1. Correção no Uso do Método: o requisito deve ser descrito de acordo com a técnica ou o método de desenvolvimento utilizado.		
2. Uniformidade de terminologia: o requisito deve ser descrito através de notações e termos padronizados.		
3. Uniformidade de Abstração: o requisito possui um nível de detalhamento uniforme, considerando-se um determinado estágio de desenvolvimento**.**		
4. Concisão: o requisito contém um volume mínimo de texto, por se ter avaliado um volume mínimo de informação por unidade de texto.		
5. Simplicidade: o requisito deve ser o mais simples possível, já que requisitos complexos são muito mais difíceis de serem entendidos, testados, implementados, e reutilizados.		
6. Disponibilidade: o requisito deve poder ser facilmente manipulado na sua versão mais atualizada.		
7. Rastreabilidade: o requisito deve poder ser rastreado, identificando-se a agregação de detalhes de um dado aspecto, desde sua visão mais global até a mais detalhada e vice-versa.		
8. Avaliabilidade: o requisito pode ser avaliado com relação a sua forma e seu conteúdo, para se garantir o desenvolvimento de uma especificação de software boa qualidade.		
9. Manutenibilidade: o requisito deve poder ser facilmente modificado e detalhado sem perder a sua qualidade.		
10. Consistência: o requisito deve está isento de contradições entre os aspectos especificados para o mesmo.		
11. Não Ambigüidade: os requisitos expressam um conteúdo de tal forma que não se tenham diferentes interpretações para qualquer um dos aspectos tratados.		
12. Necessidade: o requisito deve abordar apenas aspectos considerados imprescindíveis ao entendimento do mesmo, evitando-se, assim, descrições desnecessárias e irrelevantes.		
13. Não Redundância: o requisito não deve abordar aspectos tratados em outros requisitos, evitando que uma mesma questão seja tratada em vários lugares.		
14. Completitude: o requisito deve garantir que os aspectos abordados nele foram totalmente cobertos.		

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo