UFES - Universidade Federal do Espírito Santo

ESTENDENDO ODED: AXIOMATIZAÇÃO E ADEQUAÇÃO AO META-MODELO DA UML

Projeto Final de Graduação

Vítor Estêvão Silva Souza

VITÓRIA, ES
ABRIL DE 2004

ESTENDENDO ODED: AXIOMATIZAÇÃO E ADEQUAÇÃO AO META-MODELO DA UML

Vítor Estêvão Silva Souza

MONOGRAFIA APRESENTADA COMO EXIGÊNCIA PARCIAL PARA OBTENÇÃO DO TÍTULO DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO À UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, NA ÁREA DE CONCENTRAÇÃO ENGENHARIA DE SOFTWARE, SOB A ORIENTAÇÃO DO PROF. RICARDO DE ALMEIDA FALBO.

Aprovada por:

Prof. Orientador Ricardo de Almeida Falbo, D.Sc.

Prof. Davidson Cury, D. Sc.

Prof. Crediné Silva de Menezes, D. Sc.

AGRADECIMENTOS

Gostaria de agradecer primeiramenta à Deus por todas as opotunidades de aprendizado que tive e que ainda terei em minha vida.

Agradeço à minha família pelo apoio incondicional em todos os momentos.

Um muito obrigado aos meus amigos, em especial a minha namorada, por terem me acompanhado e incentivado durante esta tão importante etapa que se encerra.

Por fim, mas não menos importante, obrigado professor Ricardo Falbo, por ser além de orientador, um grande amigo. Sua didática, disciplina e ética na universidade são exemplos a serem seguidos.

VÍTOR ESTÊVÃO SILVA SOUZA

RESUMO

Ontologias têm sido bastante utilizadas na Engenharia de Software, porém, sua construção não é tarefa simples. Dentro do contexto do ambiente de desenvolvimento de software ODE foi criado ODEd, um editor de ontologias desenvolvido para apoiar a orientação a domínio no ambiente. Este trabalho propõe adequar ODEd ao meta-modelo da UML e adicionar algumas novas funcionalidades para melhor apoiar as fases de formalização e avaliação de ontologias, através da construção de um Editor de Axiomas e da integração de uma máquina de inferência.

ÍNDICE

Capítulo 1

Introdução

Nas últimas décadas presenciamos o rápido crescimento da importância do software para a civilização. Ele é a força propulsora das empresas modernas, está presente nas tomadas de decisão, na investigação científica, na resolução de problemas e embutido em sistemas de todo o tipo. O software se tornou um elemento do cotidiano e, muitas vezes, nossas próprias vidas dependem dele, de forma que se faz necessário garantir que ele funcione corretamente (PRESSMAN, 2001).

A busca pela qualidade no desenvolvimento de software vem, portanto, aumentando a cada dia. Para suportar a demanda do mercado, os desenvolvedores precisam primar cada vez mais pela qualidade, procurando atender a todos os requisitos impostos ao sistema. Entretanto, à medida que os softwares requeridos pelos usuários crescem em complexidade, o processo de desenvolvimento de software (ou *processo de software*) também se torna mais complexo e passa a ser imprescindível a utilização de métodos adequados e ferramentas automatizadas para apoiar suas tarefas.

As ferramentas CASE (*Computer Aided Software Engineering*) vieram atender a esta demanda. Contudo, elas falham em tratar o processo como um todo, atacando apenas suas atividades isoladamente. Assim, surgiram os Ambientes de Desenvolvimento de Software (ADS), que buscam combinar técnicas, métodos e ferramentas para apoiar o Engenheiro de Software na construção de produtos de software, abrangendo todas as atividades inerentes ao processo, tais como planejamento, gerência, desenvolvimento, documentação e controle da qualidade (FALBO, 1998).

Um ADS está sendo desenvolvido no Laboratório de Engenharia de Software (LabES) do Departamento de Informática da UFES: o ambiente ODE (*Ontology-based software Development Environment*), que tem como base ontologias (FALBO et al., 2003).

Além da qualidade, é importante também primar pela produtividade. Nesse contexto, o reuso de software tem sido apontado como uma das mais promissoras abordagens para

atacar problemas de qualidade e produtividade, principalmente quando aplicamos o conceito de desenvolvimento para reuso no processo de software (FALBO et al., 2002). Nessa abordagem, ontologias têm um importante papel, principalmente quando utilizadas no contexto da Engenharia de Domínio, cujo principal objetivo é a modelagem dos elementos de um domínio para serem reutilizados em vários projetos de software para aquele domínio.

O ambiente ODE possui uma ferramenta que apóia a atividade de Engenharia de Domínio, provendo funcionalidades de edição de ontologias: ODEd (*ODE's Ontology Editor*) (MIAN et al., 2002). Este trabalho propõe alterações nessa ferramenta, incluindo melhorias na infra-estrutura e adição de novas funcionalidades.

1.1. Objetivos

É objetivo deste projeto incluir melhorias na infra-estrutura de ODEd, a saber:

- Adequação dos modelos e diagramas de ontologia ao meta-modelo da UML, da mesma forma como foi proposto em (SILVA, 2003) com relação à ferramenta de modelagem UML OODE;
- Substituição da camada de persistência vigente (COSTA, 2000) por uma mais nova (RUY, 2003);
- Criação de uma interface mais amigável para instanciação de conceitos da ontologia;
- Ajuste do código aos novos padrões de implementação;
- Documentação.

Também é objetivo deste trabalho adicionar as seguintes novas funcionalidades, algumas já apresentadas como protótipo em (SOUZA et al., 2003):

- Permitir que o engenheiro de domínio escreva qualquer axioma, não ficando limitado somente às oito propriedades de relação, chamadas de "teorias" em (MIAN, 2003);
- Integração de JTP (FIKES et al., 2003), uma máquina de inferência que permite que se façam consultas à ontologia, provendo um melhor suporte à

atividade de avaliação;

- Integração da mesma máquina de inferência à infra-estrutura de objetos gerada, de forma que o engenheiro de ontologias leve o conhecimento do domínio para outras fases do processo de software.

1.2. Metodologia

Este trabalho foi desenvolvido em cinco grandes fases: revisão bibliográfica, pesquisa, desenvolvimento de um protótipo, planejamento das funcionalidades da nova versão e desenvolvimento dessa versão.

As três primeiras fases foram desenvolvidas como projeto de Iniciação Científica no LabES. Foi feita uma revisão bibliográfica sobre Ambientes de Desenvolvimento de Software e Ontologias, seguida de uma pesquisa sobre linguagens de representação de ontologia e máquinas de inferência. Como resultado do projeto, foi implementado um protótipo denominado AxE (*Axiom Editor*) (SOUZA et al., 2003).

As duas últimas fases foram realizadas durante o projeto de graduação, tendo sido delimitadas as funcionalidades que estariam presentes na nova versão de ODEd, apresentada nesta monografia e desenvolvida segundo o processo definido no capítulo 3.

1.3. Organização da Monografia

Este documento está dividido em 7 capítulos.

No capítulo 2 é feito um apanhado geral dos temas relevantes a esta monografia: Engenharia de Domínio, Ontologias, Ambientes de Desenvolvimento de Software e Orientação a Domínio em ADSs. O capítulo encerra com uma visão geral de ODEd e sua evolução.

O capítulo 3, como citado anteriormente, apresenta o processo de software utilizado para desenvolvimento deste trabalho. Os capítulos 4, 5 e 6 são resultados desse processo e apresentam os resultados das atividades de Especificação de Requisitos, Análise, Projeto, Implementação e Testes.

No capítulo 7 encontram-se as conclusões e perspectivas futuras.

Capítulo 2

Uso de Ontologias na Engenharia de Software

Atualmente, qualidade e produtividade têm sido duas das principais preocupações dos desenvolvedores de software. É preciso atender à demanda de desenvolvimento do mercado, que cada dia precisa de mais produtos de software, em prazos cada vez menores, aliando, no entanto, qualidade suficiente para que o cliente fique satisfeito com o resultado.

Ontologias podem auxiliar nesse processo de obtenção de produtividade com qualidade. Elas estão intimamente ligadas ao processo de engenharia de domínio que, por sua vez, está relacionado ao reuso de artefatos num nível mais alto de abstração, abrangendo toda uma classe de problemas, e não só um problema específico.

Este capítulo começa ressaltando alguns conceitos da Engenharia de Domínio e o importante papel que ontologias podem desempenhar em suas atividades. Como este trabalho se propõe a aplicar estes conceitos em um Ambiente de Desenvolvimento de Software (ADS), o capítulo segue com uma breve introdução aos ADSs, focando nos ADSs Orientados a Domínio e no ambiente ODE, no qual está inserido este trabalho. Na seqüência é apresentado ODEd, o editor de ontologias de ODE e como ele apóia a abordagem para engenharia de domínio em ODE. O capitulo finaliza descrevendo o que esse trabalho proporciona para a evolução de ODEd.

2.1. Engenharia de Domínio e Ontologias

Engenharia de Domínio é uma atividade cujo propósito consiste em identificar, modelar, construir, catalogar e disseminar artefatos de software a serem utilizados em um projeto de software existente ou em futuros projetos para o mesmo domínio de aplicação (FALBO et al., 2002).

Em outras palavras, antes de analisar um problema específico a ser resolvido, faz-se uma modelagem dos elementos que compõem o domínio no qual o problema está inserido:

os conceitos que pertencem ao domínio e as relações que existem entre os conceitos, além de suas definições, propriedades e restrições.

A Engenharia de Domínio traz benefícios que podem reduzir o tempo de desenvolvimento de software através do reuso de componentes. Por exemplo, o modelo de domínio serve de base para a análise dos requisitos e projeto do sistema, introduzindo os conceitos existentes e definindo o vocabulário a ser usado. Além disso, esses elementos de domínio podem ser convertidos em artefatos utilizáveis na implementação (classes, no caso de desenvolvimento orientado a objetos), já contendo todas as propriedades e restrições modeladas no domínio (PRESSMAN, 2001).

Nesse contexto, ontologias podem servir de modelo de domínio e como estrutura do repositório de componentes reutilizáveis (FALBO et al., 2002).

Uma ontologia é uma especificação formal de uma conceituação compartilhada (GRUBER, 1993). Uma conceituação é um modelo abstrato que descreve os conceitos relevantes de algum fenômeno. É formal porque deve ser lida e interpretada por computadores e é compartilhada no sentido de capturar conhecimento consensual, ou seja, difundido e aceito por uma comunidade (BROEKSTRA et al., 2001).

ODEd, conforme mencionado anteriormente e detalhado na seção 2.3, é a ferramenta de apoio à abordagem adotada para Engenharia de Domínio em ODE. Nela, ontologias são usadas como modelos de domínio, sendo posteriormente convertidas em modelos de classes, seguindo três fases (FALBO et al., 2002):

- **Análise de Domínio:** a primeira fase do processo consiste em identificar os requisitos do modelo do domínio (quais informações ele deve representar) e, então, modelar os elementos do domínio. Nesta fase, ontologias são utilizadas como modelo de domínio;
- **Projeto de Domínio:** a segunda fase tem como objetivo converter os modelos de análise (ontologias) em modelos de projeto (modelos de classe), adaptados às tecnologias utilizadas;
- **Implementação de Domínio:** a última fase consiste em implementar os artefatos modelados na fase de projeto de domínio para serem utilizados durante a implementação de produtos de software para aquele domínio.

A Figura 2.1 detalha a fase de análise de domínio, que segue a abordagem sistemática para construção de ontologias proposta em (FALBO et al., 1998). O processo inicia-se com a identificação do propósito e especificação de requisitos, utilizando questões de competência para definir o que a ontologia deve responder ao final do processo. As fases de captura e formalização da ontologia que se seguem alternam-se até que todos conceitos, relações e restrições sejam identificados e formalizados. As atividades de avaliação, documentação e integração apóiam todo o processo, que resulta numa ontologia formal (FALBO et al., 2002) (MIAN, 2003).

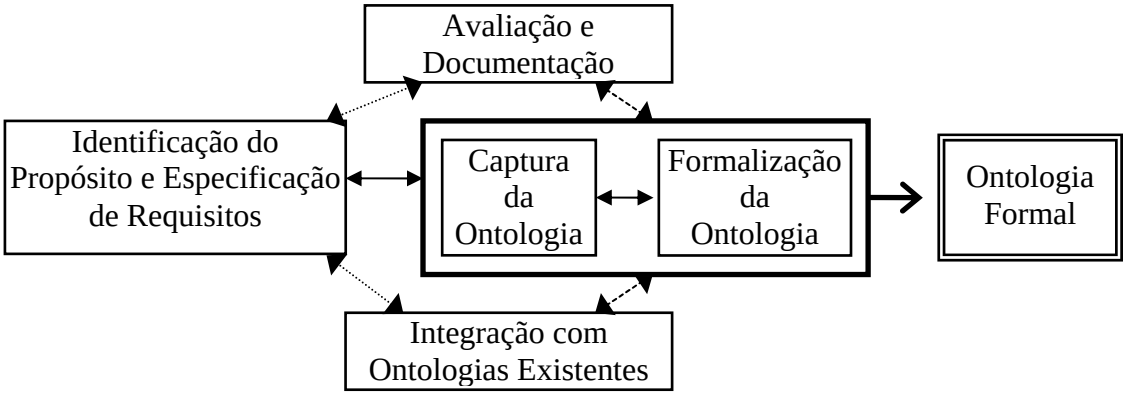

Figura 2.1. Análise de Domínio em ODEd.

A Figura 2.2 mostra como ODEd apóia as fases de projeto e implementação de domínio. Unindo as duas fases em uma, a ontologia gerada na fase anterior é convertida diretamente em classes Java, sem geração de nenhum modelo de projeto. Tais classes são passíveis de serem utilizadas posteriormente em outras fases do processo de software. Para tal, a abordagem proposta em (GUIZZARDI et al., 2001) é seguida.

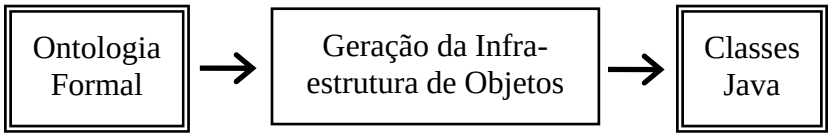

Figura 2.2. Projeto e Implementação de Domínio em ODEd.

2.2. Ambientes de Desenvolvimento de Software e o Projeto ODE

O ambiente de trabalho ideal, seja qual for a especialidade, deve incluir (PRESSMAN, 2001):

- Uma coleção de ferramentas úteis que auxiliem em cada passo do processo de construção do produto;
- Disposição organizada das ferramentas que permita que elas sejam encontradas facilmente e usadas eficientemente;
- Um especialista capacitado em utilizar as ferramentas da maneira correta.

Com relação à Engenharia de Software, as ferramentas CASE atendem ao primeiro requisito e representaram um grande avanço e um enorme benefício. Porém, tais ferramentas trabalham isoladamente, tratando atividades específicas do processo de software, e maiores vantagens só podem ser alcançadas com sua integração.

Essa é a idéia por traz dos Ambientes de Desenvolvimento de Software. Um ADS é um ambiente que integra ferramentas que apóiam atividades do processo de software de maneira coordenada através de todo o ciclo de vida do software (HARRISON et al., 2000). Os ADSs atendem ao segundo requisito listado acima.

Ao integrar as ferramentas CASE, conseguimos diversos benefícios, como: (a) transferência transparente de informações (modelos, documentos, dados) de uma ferramenta para outra e de uma etapa do processo para outra; (b) diminuição do esforço para execução de tarefas presentes em todas as fases do processo, como controle de qualidade; (c) melhor planejamento, monitoramento e comunicação, resultando num maior controle sobre o projeto; (d) aprimoramento na coordenação entre diferentes pessoas trabalhando num mesmo projeto, dentre outros (PRESSMAN, 2001).

Existem vários níveis de integração de ferramentas, diferindo no suporte fornecido. Dentre eles, é possível citar (TRAVASSOS, 1994) (PFLEEGER, 2001): integração de dados, de apresentação, de controle, de plataforma, de processo e de conhecimento.

O advento da integração de processo deu origem aos ADSs Centrados em Processo (ADSCP), que apóiam a criação e a exploração de modelos de processo e permitem que o desenvolvimento de um projeto seja guiado por um processo definido.

Neste contexto, encontra-se o ambiente ODE. ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003) é um ADS centrado em processo. Portanto, possui uma ferramenta de definição e acompanhamento de processos de software e reúne diversas ferramentas CASE que apóiam as mais diversas atividades, como análise de riscos, modelagem orientada a objetos, gerência de conhecimento, etc.

No entanto, além da integração de processo, é desejável que o ambiente alcance também a integração de conhecimento. É sabido que a produtividade e a qualidade no desenvolvimento de produtos de software são fortemente influenciadas pelo grau de conhecimento que os desenvolvedores têm sobre o domínio (OLIVEIRA, 1999). Ao incorporar esse conhecimento aos ADSs, surge o conceito de Ambiente de Desenvolvimento de Software Orientado a Domínio (ADSOD).

OLIVEIRA (1999) descreve que dois atores interagem com um ADSOD: os Engenheiros de Domínio e os Desenvolvedores (usuários das ferramentas de desenvolvimento). Os primeiros modelam o conhecimento do domínio através de ferramentas de definição, enquanto os demais acessam e utilizam esse conhecimento em cada etapa do processo de software.

Uma abordagem possível para essa modelagem de conhecimento de domínio é a Engenharia de Domínio utilizando ontologias, como apresentado na seção 2.1. Ontologias podem auxiliar no entendimento de um problema ou tarefa ao definir o vocabulário utilizado, facilitar processo de identificação dos requisitos do sistema e o entendimento das relações entre os diversos componentes, o que é extremamente importante em sistemas que envolvem uma equipe trabalhando em diferentes domínios (OLIVEIRA, 1999).

Um dos objetivos atuais do Projeto ODE é transformar o ambiente em um ADS Orientado a Domínio. Com essa finalidade foi desenvolvido ODEd. Em ODEd, os engenheiros de domínio executam as fases de análise, projeto e implementação de domínio e os desenvolvedores utilizam os artefatos gerados por elas: ontologias (modelos de análise) e infra-estruturas de classes (artefatos implementados). No presente momento, ODEd não trabalha com modelos de projeto.

Nas seções seguintes são apresentados mais detalhadamente ODEd e sua evolução, resultado deste trabalho.

2.3. ODEd: O Editor de Ontologias do Ambiente ODE

ODEd (*ODE's Ontology Editor*) é um editor de ontologias que busca apoiar a tarefa de Engenharia de Domínio em ODE, através do processo já apresentado na seção 2.1, possibilitando a um engenheiro de domínio modelar conhecimento sobre uma classe de aplicações e compartilhá-lo através das etapas do processo de desenvolvimento (MIAN et al., 2002) (MIAN, 2003).

ODEd provê as seguintes funcionalidades (MIAN, 2003):

- Definição de questões de competência para apoiar a tarefa de especificação de requisitos da ontologia;
- Criação de ontologias utilizando representações gráficas em LINGO (FALBO, 1998) e UML, apoiando a fase de análise de domínio;
- Integração de ontologias, através da importação de conceitos;
- Geração automática de alguns tipos de axiomas pré-definidos, a saber: atomicidade, disjunção, exclusividade, reflexibilidade, anti-reflexibilidade, simetria, anti-simetria e transitividade;
- Derivação de uma infra-estrutura de objetos a partir da ontologia, apoiando as fases de projeto e implementação de domínio de uma só vez;
- Geração automática de aplicações para instanciação de conceitos e relações da ontologia;
- Geração de tutoriais sobre o domínio em HTML.

ODEd, no entanto, não permite que o engenheiro de domínio escreva seus próprios axiomas, limitando, assim, duas das atividades do processo de desenvolvimento de ontologias (Figura 2.1):

- Formalização: com suporte limitado à definição de axiomas, o engenheiro de domínio não consegue formalizar diversos tipos de restrição que as relações entre os conceitos podem apresentar, pois fica restrito aos oito tipos de axiomas pré-definidos;
- Avaliação: sem uma máquina de inferência integrada não é possível haver um processo automatizado de verificação das questões de competência.

2.4. Evolução de ODEd

A motivação para este trabalho está centrada exatamente nas limitações apresentadas anteriormente, sendo seus objetivos principais, como já listados na seção 1.1, a definição de uma linguagem para formalização de axiomas, a criação de um editor de axiomas para ODEd, a integração de uma máquina de inferência para avaliação da ontologia e a adição da capacidade de inferência à infra-estrutura de objetos gerados, para que o conhecimento sobre as restrições do domínio representado na forma de axiomas possa estar disponível para uso em outras fases do processo de desenvolvimento.

Uma parte desse trabalho foi feita durante o período de Iniciação Científica (SOUZA et al., 2003), o que incluiu:

- A definição de DAML+OIL (*DARPA Agent Markup Language + Ontology Inference Layer*) (CONNOLLY et al., 2001) como linguagem de representação de ontologias por ser a base para OWL (*Ontology Web Language*), padrão da W3C (*World Wide Web Consortium*) para formalização de ontologias;
- A escolha de JTP (*Java Theorem Prover*) (FIKES et al., 2003) como máquina de inferência para apoiar a avaliação de ontologias em ODEd. JTP foi desenvolvido em Java, linguagem padrão do projeto ODE, é gratuito e possui código aberto;
- Devido ao limitado poder de expressão de DAML+OIL quando se trata de axiomas e também devido ao fato de JTP já possuir suporte a KIF (*Knowledge Interchange Format*) (GENESRETH, 1992), esta linguagem foi escolhida como linguagem para escrita de axiomas;
- Foi implementado um protótipo de um editor de axiomas, denominado AxE (*Axiom Editor*), inicialmente não integrado a ODEd.

Tomando por base o trabalho previamente desenvolvido, novos requisitos foram identificados, a saber:

- Re-implementação de ODEd com vistas a: (i) adequação ao meta-modelo UML, (ii) utilização da nova camada de persistência de ODE, (iii) criação de interfaces humano-máquina mais amigáveis, (iv) adequação aos padrões de

implementação e documentação do projeto ODE, (v) uso de padrões de projeto (*design patterns*) e novas tecnologias, como XSLT;

- Integração de AxE a ODEd e de ODEd a ODE;
- Documentação de ODEd: casos de uso (Capítulo 4), modelos de análise (Capítulo 5) e de projeto (Capítulo 6);
- Integração da máquina de inferência à infra-estrutura de objetos gerada.

Capítulo 3

O Processo de Desenvolvimento de Software

Com o crescimento da importância do software para a sociedade moderna, vimos o surgimento de uma nova área de pesquisa, a Engenharia de Software, que, como qualquer engenharia, procura analisar, projetar, construir, verificar e manter, neste caso, software (PRESSMAN, 2001).

Para garantir a qualidade do software construído, é fundamental definir um processo de software e garantir que este seja um processo de qualidade. Também se faz necessário escolher um paradigma para o desenvolvimento do software, dentre os quais os mais comuns são os paradigmas estruturado e orientado a objetos.

Este trabalho foi desenvolvido segundo o paradigma orientado a objetos e através de um processo de software composto por cinco fases técnicas: levantamento de requisitos, análise, projeto, implementação e testes.

O objetivo deste capítulo é apresentar brevemente o paradigma utilizado e discutir o processo de software que norteou o desenvolvimento do trabalho.

3.1. Desenvolvimento Orientado a Objetos

Em qualquer paradigma de desenvolvimento, o objetivo é mapear conceitos relevantes do mundo real para o mundo computacional, gerando, assim, uma solução informatizada para um problema detectado.

O paradigma estruturado, baseado na arquitetura de Von Neumann, mapeia esses conceitos do mundo real para dois elementos no mundo computacional: funções e dados. Dados são repositórios de informação passivos, enquanto que as funções são ativas, isto é, possuem comportamento, e manipulam os dados.

Já o paradigma orientado a objetos pressupõe que o mundo real é povoado por objetos. Um objeto é uma entidade que combina estrutura de dados com comportamento

funcional. Cabe ao desenvolvedor identificar quais objetos e suas características são relevantes para a solução de um problema. Esta forma de ver o problema diminui a distância conceitual entre o que existe no mundo real e o que é modelado como solução, ao trabalhar com noções intuitivas e mais naturais.

São características da orientação a objetos:

- **Abstração**: consiste em construir modelos do mundo real compostos somente pelos elementos mais relevantes para o entendimento de determinado aspecto;
- **Encapsulamento**: é a capacidade de utilizar os objetos conhecendo apenas sua interface externa, ou seja, como usá-lo, sendo desnecessário saber o seu funcionamento interno;
- **Modularidade**: propriedade de sistemas compostos por módulos coesos e fracamente acoplados, ou seja, cada componente é autônomo, possui uma função simples e bem definida;
- **Hierarquia**: forma de organizar abstrações, identificando relações de superclasse/subclasse e definindo as características de cada elemento da taxonomia;
- **Polimorfismo**: estabelece que um objeto pode assumir diversas formas e que os elementos que com ele interagem não precisam saber qual forma ele está assumindo, se todas elas possuem uma mesma interface externa.

A orientação a objetos não é a solução definitiva para os problemas na construção de produtos de software de qualidade, mas se for praticada corretamente e se forem observadas outras técnicas da Engenharia de Software (definição de processo, uso de métricas, reuso etc) é um paradigma que pode levar a melhorias substanciais (YOURDON, 1994).

3.2. Processo de Software

Um processo de software pode ser visto como o conjunto de atividades, métodos e práticas que guiam pessoas na produção de software. Um processo de desenvolvimento de software deve determinar quais atividades serão realizadas, ou seja, um modelo de ciclo de

vida do software, quais artefatos são consumidos e produzidos por cada atividade, quais procedimentos serão adotados para realização das atividades e quais recursos serão necessários (FALBO, 2000).

Um processo de software não pode ser definido de forma universal. Ele deve ser adequado ao domínio da aplicação e ao problema específico a ser resolvido. Contudo, de maneira geral, geralmente encontramos as atividades de planejamento, especificação de requisitos, análise, projeto, implementação e testes (FALBO, 2000).

Para o desenvolvimento desse trabalho, definiu-se um ciclo de vida evolutivo, dado que esse trabalho é resultado de uma reestruturação da primeira versão de ODEd, integrado a um protótipo desenvolvido como trabalho de Iniciação Científica. Estes dois produtos são, de fato, os resultados dos dois primeiros ciclos de desenvolvimento de ODEd. Este trabalho representa mais um ciclo, como mostra a Figura 3.1.

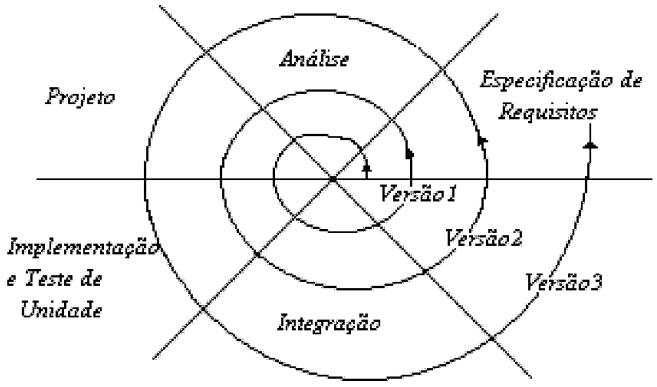

Figura 3.1. Ciclo de vida em espiral.

Neste terceiro ciclo, foram realizadas as atividades de especificação de requisitos, análise, projeto, implementação e testes. Novamente, por se tratar de uma reestruturação de ODEd, todas as fases foram realizadas levando em consideração todo o sistema, e não só as contribuições que este trabalho trouxe.

Desta maneira, as fases de especificação de requisitos e análise geraram uma documentação mais completa de ODEd, enquanto que as atividades de projeto e implementação focaram na evolução do editor, utilizando *design patterns*, adequando o código aos novos padrões definidos pelo projeto ODE, adequando os modelos de domínio ao meta-modelo da UML e propondo uma melhoria para a interface com o usuário.

Os capítulos que se seguem apresentam o principais produtos resultantes das atividades de especificação de requisitos (Capítulo 4), análise (Capítulo 5), projeto, implementação e testes (Capítulo 6), realizadas neste último ciclo.

Capítulo 4

Especificação de Requisitos Funcionais

A especificação de requisitos descreve o sistema sob uma perspectiva externa, utilizando modelos de caso de uso com suas respectivas descrições para estruturar esta visão. Os requisitos capturados nesta fase modelam as funcionalidades que o sistema deve oferecer a seus usuários.

Como uma especificação de requisitos estava fora do escopo de (MIAN, 2003), são apresentados neste capítulo os diagramas e as descrições dos casos de uso de ODEd, documentando as funcionalidades que o software já possui e adicionando os novos requisitos para este trabalho.

Os requisitos de ODEd foram agrupados em dois pacotes principais, como mostra a Figura 4.1: edição de ontologias e geração de artefatos que podem ser utilizados externamente.

Figura 4.1. Diagrama de Pacotes

4.1. Especificação de Requisitos da Edição de Ontologias

O diagrama de casos de uso mostrado na Figura 4.2 modela as funcionalidades disponíveis para edição de ontologias. O Engenheiro de Domínio, único ator do sistema, representa todos os usuários que interagem com a ferramenta.

Os casos de uso *Controlar Ontologia*, *Controlar Diagrama de Ontologia*, *Editar Questão de Competência*, *Editar Ontologia* e *Avaliar Ontologia* compõem este diagrama.

Os três primeiros possuem suas descrições a seguir, enquanto os dois últimos são agrupamentos de casos de uso e são detalhados nas seções 4.2 e 4.3.

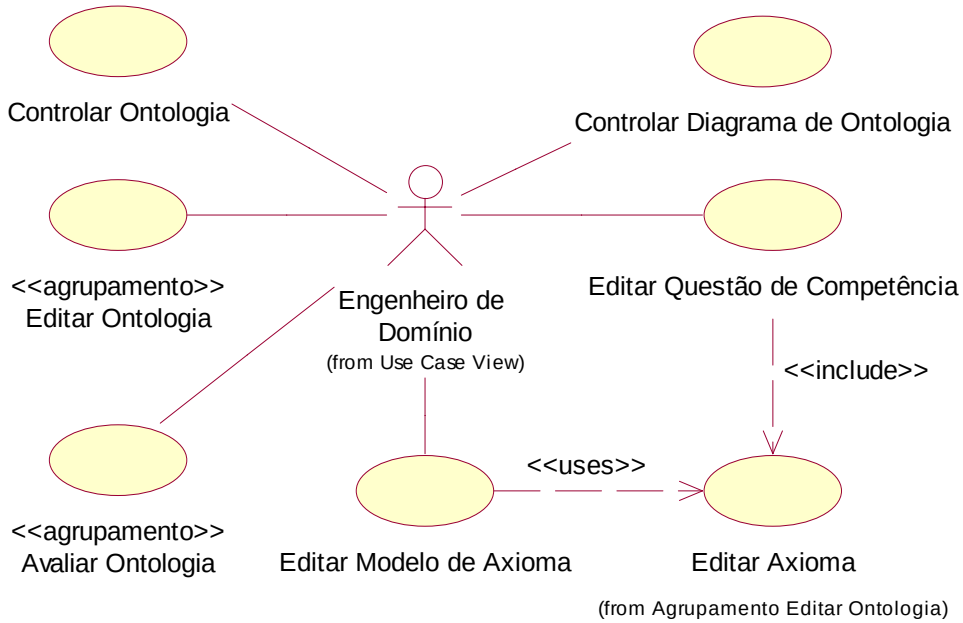

Figura 4.2. Diagrama de Casos de Uso Edição de Ontologias.

4.1.1. Caso de Uso Controlar Ontologia

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo funcionalidades de controle de ontologia, ou seja, criar uma ontologia, abrir e fechar ontologias criadas, alterar seus dados e salvá-los. Os eventos *Criar Nova Ontologia*, *Abrir Ontologia*, *Fechar Ontologia*, *Salvar Ontologia*, *Consultar Dados de Ontologia*, *Alterar Dados de Ontologia* e *Excluir Ontologia* compõem este caso de uso.

Curso Normal

Criar Nova Ontologia

O engenheiro de domínio solicita a criação de uma nova ontologia e informa o nome e a descrição da mesma. A ontologia, então, é criada e o cenário Abrir Ontologia é executado

automaticamente para ela.

Abrir Ontologia

O engenheiro de domínio informa qual ontologia deseja abrir e a mesma é aberta. Abrir uma ontologia significa exibir os elementos que a compõem.

Fechar Ontologia

O engenheiro de domínio informa qual ontologia deseja fechar, dentre as que estão abertas, e a mesma é fechada. Fechar uma ontologia significa não mais exibir os elementos que a compõem.

Salvar Ontologia

O engenheiro de domínio informa qual ontologia deseja salvar e a mesma é salva. Salvar uma ontologia significa gravar em mídia persistente suas informações e de seus componentes.

Consultar Dados de Ontologia

O engenheiro de domínio informa de qual ontologia deseja ver as características e essas são exibidas. São características de uma ontologia: nome e descrição.

Alterar Dados de Ontologia

O engenheiro de domínio informa de qual ontologia deseja alterar as características e informa os novos nome e descrição da mesma. Os dados são alterados.

Excluir Ontologia

O engenheiro de domínio informa qual ontologia deseja excluir e uma mensagem é exibida, pedindo a confirmação da exclusão. Caso seja confirmada, a ontologia é excluída. Excluir uma ontologia significa excluir também os diagramas e questões de competência que a compõem. Conceitos e relações que fazem parte somente desta ontologia também são excluídos.

Cursos Alternativos / de Exceção

Criar Nova / Alterar Dados de Ontologia

Uma ontologia deve possuir um nome e não deve haver duas ontologias com o mesmo nome. Caso não cumpra estes requisitos, uma mensagem de erro é mostrada e a

ação não é efetivada.

4.1.2. Caso de Uso Controlar Diagrama de Ontologia

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo funcionalidades de controle de diagrama de ontologia, ou seja, criar um diagrama, abrir e fechar diagramas criados, alterar seus dados e salvá-los. Os eventos *Criar Novo Diagrama de Ontologia*, *Abrir Diagrama de Ontologia*, *Fechar Diagrama de Ontologia*, *Salvar Diagrama de Ontologia*, *Consultar Dados de Diagrama de Ontologia*, *Alterar Dados de Diagrama de Ontologia* e *Excluir Diagrama de Ontologia* compõem este caso de uso.

Curso Normal

Criar Novo Diagrama de Ontologia

O engenheiro de domínio solicita a criação de um novo diagrama de ontologia e informa o nome e a descrição do mesmo. O diagrama, então, é criado e o cenário Abrir Diagrama de Ontologia é executado automaticamente para ele.

Abrir Diagrama de Ontologia

O engenheiro de domínio informa qual diagrama de ontologia deseja abrir e o mesmo é aberto. Abrir um diagrama de ontologia significa exibir os elementos que o compõem.

Fechar Diagrama de Ontologia

O engenheiro de domínio informa qual diagrama de ontologia deseja fechar, dentre os que estão abertos, e o mesmo é fechado. Fechar um diagrama de ontologia significa não mais exibir os elementos que o compõem.

Salvar Diagrama de Ontologia

O engenheiro de domínio informa qual diagrama de ontologia deseja salvar e o mesmo é salvo. Salvar um diagrama de ontologia significa gravar em mídia persistente suas informações e de seus componentes.

Consultar Dados de Diagrama de Ontologia

O engenheiro de domínio informa de qual diagrama de ontologia deseja ver as características e essas são exibidas. São características de um diagrama de ontologia: nome

e descrição.

Alterar Dados de Diagrama de Ontologia

O engenheiro de domínio informa de qual diagrama de ontologia deseja alterar as características e informa os novos nome e descrição do mesmo. Os dados são alterados.

Excluir Diagrama de Ontologia

O engenheiro de domínio informa qual diagrama de ontologia deseja excluir e uma mensagem é exibida, pedindo a confirmação da exclusão. Caso seja confirmada, o diagrama é excluído. Vale ressaltar que os conceitos e relações presentes no diagrama continuam existindo na ontologia.

Cursos Alternativos / de Exceção

Criar Novo / Alterar Dados de Diagrama

Um diagrama de ontologia deve possuir um nome e não deve haver dois diagramas na mesma ontologia com o mesmo nome. Caso não cumpra estes requisitos, uma mensagem de erro é mostrada e a ação não é efetivada.

4.1.3. Caso de Uso Editar Questão de Competência

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo funcionalidades de cadastro e edição de questões de competência, ou seja, criá-las, excluí-las, consultar e alterar seus dados e formalizá-las através de axiomas. Os eventos *Criar Nova Questão de Competência*, *Consultar Dados de Questão de Competência*, Alterar *Dados de Questão de Competência*, *Excluir Questão de Competência* e *Formalizar Questão de Competência* compõem este caso de uso.

Curso Normal

Criar Nova Questão de Competência

O engenheiro de domínio solicita a criação de uma nova questão de competência e informa a descrição da mesma. A questão, então, é criada.

Consultar Dados de Questão de Competência

O engenheiro de domínio informa de qual questão de competência deseja ver as características e essas são exibidas. São características de uma questão de competência: descrição e formalização.

Alterar Dados de Questão de Competência

O engenheiro de domínio informa de qual questão de competência deseja alterar as características e informa a nova descrição da mesma. Os dados são alterados.

Excluir Questão de Competência

O engenheiro de domínio informa qual questão de competência deseja excluir e uma mensagem é exibida, pedindo a confirmação da exclusão. Caso seja confirmada, a questão é excluída.

Formalizar Questão de Competência

Pré-condição: a ontologia deve possuir algum conjunto de conceitos e relações já definido.

O engenheiro de domínio informa qual questão de competência deseja formalizar e, realizando o caso de uso *Editar Axioma*, informa o axioma que a formaliza. O axioma informado é associado àquela questão de competência.

Cursos Alternativos / de Exceção

Criar Nova / Alterar Dados de Questão de Competência

Uma questão de competência deve possuir uma descrição. Caso não cumpra este requisito, uma mensagem de erro é mostrada e a ação não é efetivada.

4.1.4. Caso de Uso Editar Modelo de Axioma

Este caso de uso permite que o engenheiro de domínio crie, exclua, consulte e altere os modelos de axioma que estão disponíveis para todas as ontologias. Modelos de axioma são atalhos para a criação de axiomas mais usados, como transitividade, reflexibilidade, simetria, dentre outros. Este caso de uso é composto pelos eventos *Criar Novo Modelo de Axioma*, *Consultar Dados de Modelo de Axioma*, *Alterar Dados de Modelo de Axioma*,

Excluir Modelo de Axioma e *Criar Novo Axioma a partir de Modelo*.

Curso Normal

Criar Novo Modelo de Axioma

Como um modelo de axioma é um axioma, o engenheiro de domínio usa o caso de uso *Editar Axioma* para criar o novo modelo, através do evento *Criar Novo Axioma*.

Consultar Dados de Modelo de Axioma

O engenheiro de domínio informa de qual modelo de axioma deseja ver as características e essas são exibidas, através da utilização do cenário *Consultar Dados de Axioma*, do caso de uso *Editar Axioma*.

Alterar Dados de Modelo de Axioma

O engenheiro de domínio informa de qual questão de competência deseja alterar as características executa o cenário *Alterar Dados de Axioma*, do caso de uso *Editar Axioma*.

Excluir Modelo de Axioma

O engenheiro de domínio informa qual questão de competência deseja excluir e executa o cenário *Excluir Axioma* do caso de uso *Editar Axioma*.

Criar Novo Axioma a partir de Modelo

O engenheiro de ontologias informa em qual relação quer criar um novo axioma e qual modelo de axioma quer utilizar para tal. Um novo axioma é criado a partir do modelo, assim como explicado na seção 5.3.

4.2. Especificação de Requisitos do Caso de Uso Editar Ontologia

O caso de uso Editar Ontologia é composto por seis casos de uso e modela as funções de edição da ontologia propriamente dita. Novamente, o único ator presente é o Engenheiro de Domínio.

Os casos de uso que o compõem são: *Editar Axioma*, *Editar Conceito*, *Editar Relação*, *Editar Propriedade*, *Importar Conceito* e *Definir Axiomatização de Relação*. Tais

casos de uso estão representados na Figura 4.3 e são descritos a seguir.

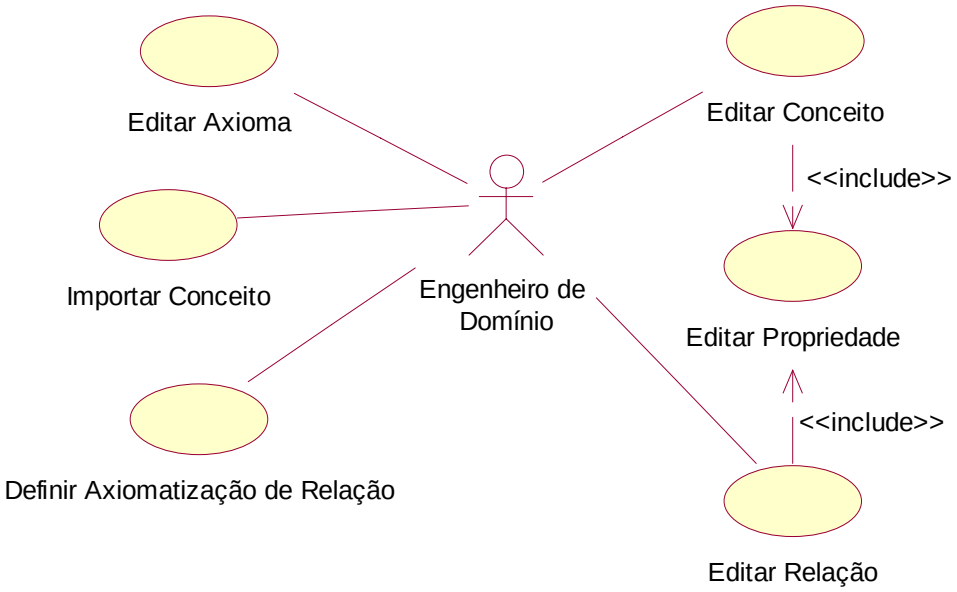

Figura 4.3. Diagrama de Casos de Uso Editar Ontologias.

4.2.1. Caso de Uso Editar Axioma

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo funcionalidades de cadastro de axiomas, ou seja, criá-los, excluí-los, consultar e alterar seus dados. Os eventos *Criar Novo Axioma*, *Consultar Dados de Axioma*, *Alterar Dados de Axioma* e *Excluir Axioma* compõem este caso de uso.

Curso Normal

Criar Novo Axioma

O engenheiro de domínio solicita a criação de um novo axioma e informa a descrição do mesmo, determina se o axioma será escrito na forma infixa ou pré-fixa e informa a sentença que formaliza o axioma. O axioma, então, é criado.

Consultar Dados de Axioma

O engenheiro de domínio informa de qual axioma deseja ver as características e essas são exibidas. São características de um axioma: descrição, formalização e se ele está escrito em notação infixa ou pré-fixa.

Alterar Dados de Axioma

O engenheiro de domínio informa de qual axioma deseja alterar as características e informa as novas descrição e formalização, além de determinar se ele está escrito em notação infixa ou pré-fixa. Os dados são alterados.

Excluir Axioma

O engenheiro de domínio informa qual axioma deseja excluir e uma mensagem é exibida, pedindo a confirmação da exclusão. Caso seja confirmada, o axioma é excluído.

Cursos Alternativos / de Exceção

Criar Novo / Alterar Dados de Axioma

Um axioma deve possuir uma sentença que o formaliza e a definição de notação (infixa ou pré-fixa). Caso não cumpra este requisito, uma mensagem de erro é mostrada e a ação não é efetivada.

4.2.2. Caso de Uso Editar Conceito

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo funcionalidades de cadastro de conceitos, ou seja, criá-los, excluí-los, consultar e alterar seus dados. Os eventos *Criar Novo Conceito*, *Incluir Conceito em Diagrama*, *Consultar Dados de Conceito*, *Alterar Dados de Conceito*, *Remover Conceito de Diagrama* e *Remover Conceito de Ontologia* compõem este caso de uso.

Curso Normal

Criar Novo Conceito

O engenheiro de domínio solicita a criação de um novo conceito e informa o nome e a descrição do mesmo. O conceito, então, é criado na ontologia atual e é incluído no diagrama atual.

Incluir Conceito em Diagrama

O engenheiro de domínio solicita a inclusão de um conceito da ontologia em um determinado diagrama. O conceito é, então, incluído naquele diagrama. Ao incluir um

conceito num diagrama, as relações que ele possui com os conceitos que já pertencem ao diagrama são também incluídas.

Consultar Dados de Conceito

O engenheiro de domínio informa de qual conceito deseja ver as características e essas são exibidas. São características de um conceito: nome, descrição e suas propriedades. Para consultar os dados de suas propriedades, utiliza-se o caso de uso Editar Propriedades.

Alterar Dados de Conceito

O engenheiro de domínio informa de qual conceito deseja alterar as características e informa os novos nome e descrição. Os dados são alterados.

O engenheiro de domínio pode, também, criar, alterar dados de e excluir as propriedades de um conceito a partir desse cenário. Para isso, utiliza-se o caso de uso Editar Propriedade.

Remover Conceito de Diagrama

O engenheiro de domínio informa qual conceito deseja remover do diagrama atual e o mesmo é removido.

Remover Conceito da Ontologia

O engenheiro de domínio informa qual conceito deseja remover da ontologia atual e uma mensagem é exibida, pedindo a confirmação da exclusão. Caso seja confirmada, o conceito é removido da ontologia, juntamente as relações que o associam a outros conceitos. Caso ele não pertença a nenhuma outra ontologia, é excluído definitivamente, ocorrendo o mesmo com suas propriedades e as relações que o associam a outros conceitos.

Cursos Alternativos / de Exceção

Criar Novo / Alterar Dados de Conceito

Um conceito deve possuir um nome. Caso não cumpra este requisito, uma mensagem de erro é mostrada e a ação não é efetivada.

4.2.3. Caso de Uso Editar Relação

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo funcionalidades de cadastro de relações, ou seja, criá-las, excluí-las, consultar e alterar seus dados. Os eventos *Criar Nova Relação*, *Consultar Dados de Relação*, *Alterar Dados de Relação* e *Excluir Relação* compõem este caso de uso.

Curso Normal

Criar Nova Relação

O engenheiro de domínio solicita a criação de uma nova relação e informa quais conceitos a relação associa e a descrição da mesma. A relação, então, é criada.

Incluir Relação em Diagrama

O engenheiro de domínio solicita a inclusão de uma relação da ontologia em um determinado diagrama. A relação é, então, incluída naquele diagrama. Vale ressaltar que uma relação não será exibida em um diagrama caso todos os conceitos que ela associa não estejam também presentes nesse diagrama.

Consultar Dados de Relação

O engenheiro de domínio informa de qual relação deseja ver as características e essas são exibidas. São características de uma relação: os conceitos que ela associa, sua descrição e suas propriedades. Para consultar os dados de suas propriedades utiliza-se o caso de uso Editar Propriedades.

Alterar Dados de Relação

O engenheiro de domínio informa de qual relação deseja alterar as características e informa a nova descrição. Os dados são alterados.

O engenheiro de domínio pode, também, criar, alterar e excluir as propriedades de uma relação a partir desse cenário. Para isso, utiliza-se o caso de uso Editar Propriedade.

Editar Extremidade de Relação

O engenheiro de domínio informa qual extremidade de qual relação deseja alterar as características e informa a multiplicidade e o papel da extremidade. Os dados são registrados.

Remover Relação de Diagrama

O engenheiro de domínio informa qual relação deseja remover do diagrama atual e a mesma é removida.

Remover Relação da Ontologia

O engenheiro de domínio informa qual relação deseja remover da ontologia atual e uma mensagem é exibida, pedindo a confirmação da exclusão. Caso seja confirmada, a relação é removida da ontologia. Caso ela não pertença a nenhuma outra ontologia, é excluída definitivamente, assim como suas propriedades.

Cursos Alternativos / de Exceção

Criar Nova / Alterar Dados de Relação

Uma relação deve possuir um nome. Caso não cumpra este requisito, uma mensagem de erro é mostrada e a ação não é efetivada.

4.2.4. Caso de Uso Editar Propriedade

Este caso de uso é iniciado a partir dos casos de uso *Editar Conceito* ou *Editar Relação*. Ele oferece ao engenheiro de domínio funcionalidades de edição de propriedades de conceito ou de relação, ou seja, criá-las, excluí-las, consultar e alterar seus dados. Os eventos *Criar Nova Propriedade*, *Consultar Dados de Propriedade*, *Alterar Dados de Propriedade* e *Excluir Propriedade* compõem este caso de uso.

Curso Normal

Criar Nova Propriedade

O engenheiro de domínio solicita a criação de uma nova propriedade para um conceito ou uma relação e informa seu nome, descrição e tipo. A propriedade, então, é criada.

Consultar Dados de Propriedade

O engenheiro de domínio informa de qual propriedade deseja ver as características e essas são exibidas. São características de uma propriedade: nome, descrição e tipo.

Alterar Dados de Propriedade

O engenheiro de domínio informa de qual propriedade deseja alterar as características e informa os novos nome, descrição e tipo. Os dados são alterados.

Excluir Propriedade

O engenheiro de domínio informa qual propriedade deseja excluir e uma mensagem é exibida, pedindo a confirmação da exclusão. Caso seja confirmada, a propriedade é excluída.

Cursos Alternativos / de Exceção

Criar Nova / Alterar Dados de Propriedade

Uma propriedade deve possuir pelo menos um nome. Caso não cumpra este requisito, uma mensagem de erro é mostrada e a ação não é efetivada.

4.2.5. Caso de Uso Importar Conceito

Este caso de uso, que é iniciado pelo engenheiro de domínio, fornece ao mesmo a capacidade de importar de outras ontologias conceitos que julgar interessante reutilizar.

Curso Normal

O engenheiro de domínio informa qual conceito quer importar para a ontologia corrente. O conceito, caso já não esteja na ontologia corrente, é inserido nela. Todas as relações entre esse conceito e outros conceitos presentes na ontologia informada são, automaticamente, inseridas na mesma.

4.2.6. Caso de Uso Definir Axiomatização de Relação

Este caso de uso, que é iniciado pelo engenheiro de domínio, fornece ao mesmo a capacidade de atribuir a uma relação um axioma pré-estabelecido como uma propriedade da relação, como por exemplo transitividade, reflexibilidade, simetria etc. Os eventos *Definir Axiomatização* e *Remover Axiomatização* compõem este caso de uso.

Curso Normal

O engenheiro de domínio informa para qual relação quer definir a axiomatização e quais axiomas pré-estabelecidos quer usar. A axiomatização daquela relação é definida.

4.3. Especificação de Requisitos do Caso de Uso Avaliar Ontologia

O caso de uso Avaliar Ontologia é composto por seis casos de uso e modela as funções de avaliação da ontologia. Mais uma vez, o único ator presente é o Engenheiro de Domínio.

Os casos de uso que o compõem são: *Cadastrar Instância de Conceito*, *Cadastrar Instância de Relação* e *Realizar Consulta à Ontologia*. Tais casos de uso estão representados na Figura 4.4 e são descritos a seguir.

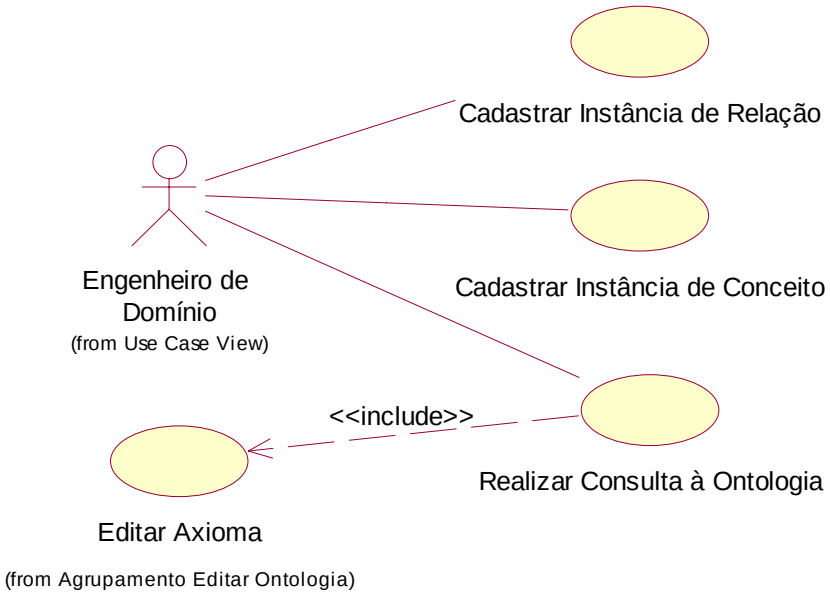

Figura 4.4. Diagrama de Casos de Uso Avaliar Ontologias.

4.3.1. Caso de Uso Cadastrar Instância de Conceito

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo

funcionalidades de cadastro de instâncias de conceitos, ou seja, criá-las, excluí-las, consultar e alterar seus dados. Os eventos *Criar Nova Instância de Conceito*, *Consultar Dados de Instância de Conceito*, *Alterar Dados de Instância de Conceito* e *Excluir Instância de Conceito* compõem este caso de uso.

Curso Normal

Criar Nova Instância de Conceito

O engenheiro de domínio solicita a criação de uma nova instância de conceito e informa seu nome, que é uma propriedade inerente a todos os conceitos, e outros valores para as propriedades definidas para aquele conceito. A instância, então, é criada.

Consultar Dados de Instância de Conceito

O engenheiro de domínio informa de qual instância de conceito deseja ver as características e essas são exibidas. São características de uma instância de conceito os valores associados às propriedades definidas para aquele conceito.

Alterar Dados de Instância de Conceito

O engenheiro de domínio informa de qual instância de conceito deseja alterar as características e informa os novos valores para as propriedades daquele conceito. Os dados são alterados.

Excluir Instância de Conceito

O engenheiro de domínio informa qual instância de conceito deseja excluir e a mesma é excluída. As instâncias de relações que associam esta instância de conceito com outras são também excluídas.

Cursos Alternativos / de Exceção

Criar Nova / Alterar Dados de Instância de Conceito

Uma instância de conceito deve possuir um nome. Caso não cumpra este requisito, uma mensagem de erro é mostrada e a ação não é efetivada.

4.3.2. Caso de Uso Cadastrar Instância de Relação

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo a capacidade de relacionar duas instâncias de conceito, o que comumente é chamado de cadastro de instâncias de relações, ou seja, criá-las, excluí-las, alterar e consultar seus dados. Os eventos *Relacionar Instâncias de Conceito*, *Consultar Dados de Instância de Relação*, *Alterar Propriedades de Instância de Relação* e *Excluir Instância de Relação* compõem este caso de uso.

Curso Normal

Relacionar Instâncias de Conceito

O engenheiro de domínio informa as instâncias de conceito que deseja relacionar e as mesmas são associadas por uma nova instância de relação.

Consultar Dados de Instância de Relação

O engenheiro de domínio informa de qual instância de relação deseja ver as características e essas são exibidas. São características de uma instância de relação: as instâncias de conceito relacionadas e os valores associados às propriedades definidas para aquela relação.

Alterar Propriedades de Instância de Relação

O engenheiro de domínio informa de qual instância de relação deseja alterar as propriedades e informa os novos valores associados à elas. Os dados são alterados.

Excluir Instância de Relação

O engenheiro de domínio informa qual instância de relação deseja excluir e a mesma

é excluída.

Cursos Alternativos / de Exceção

Relacionar Instâncias de Conceito

Uma instância de conceito deve ser informada para cada conceito relacionado pela relação. Caso não cumpra este requisito, uma mensagem de erro é mostrada e a ação não é efetivada.

4.3.3. Caso de Uso Realizar Consulta à Ontologia

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo funcionalidades de avaliação de uma ontologia através de realização de consultas a um mecanismo de inferência. Os eventos *Realizar Consulta* e *Verificar Questão de Competência* compõem este caso de uso.

Curso Normal

Realizar Consulta

O engenheiro de domínio cria um axioma, realizando o caso de uso *Editar Axioma*, e o envia como consulta à ontologia. A consulta é enviada ao mecanismo de inferência acoplado à ferramenta e o resultado lhe é apresentado.

Verificar Questão de Competência

O engenheiro de domínio informa qual questão de competência gostaria de verificar. Se a questão possuir um axioma associado, o mesmo é enviado como consulta à ontologia. A consulta é enviada ao mecanismo de inferência acoplado à ferramenta e o resultado lhe é apresentado.

4.4. Especificação de Requisitos da Geração de Artefatos

O diagrama de casos de uso mostrado na Figura 4.5 modela as funcionalidades disponíveis para geração de artefatos. Novamente, quem interage com os casos de uso é o

único ator do sistema: o Engenheiro de Domínio.

Os casos de uso *Gerar Infra-estrutura de Objetos* e *Gerar Documentação sobre o Domínio* compõem este diagrama e possuem suas descrições a seguir.

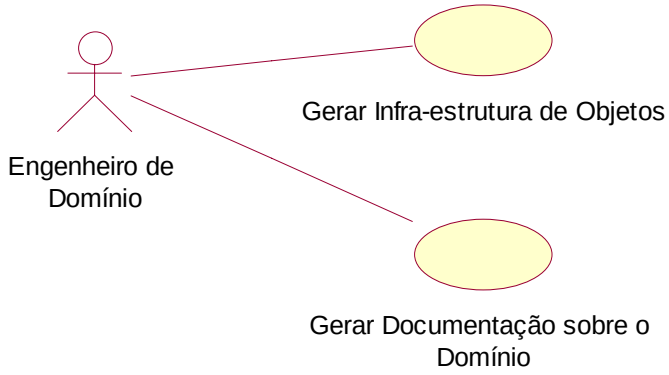

Figura 4.5. Diagrama de Casos de Uso Geração de Artefatos.

4.4.1. Caso de Uso Gerar Infra-estrutura de Objetos

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo a capacidade de gerar uma infra-estrutura de objetos Java a partir dos conceitos modelados na ontologia.

Curso Normal

O engenheiro de domínio solicita a geração da infra-estrutura de objetos e a mesma é gerada para a ontologia atual, seguindo as principais diretrizes apresentadas em (GUIZZARDI et al., 2001).

4.4.2. Caso de Uso Gerar Documentação sobre o Domínio

Este caso de uso, que é iniciado pelo engenheiro de domínio, oferece ao mesmo a capacidade de gerar uma documentação em hipertexto sobre uma ontologia.

Curso Normal

O engenheiro de domínio solicita a geração da documentação e a mesma é gerada para a ontologia atual, a partir da abordagem descrita em (MIAN, 2003).

Capítulo 5

Análise

A atividade de Análise dentro do paradigma da Orientação a Objetos tem por objetivo identificar objetos do mundo real, características destes objetos e relacionamentos entre eles que são relevantes para o problema a ser resolvido, especificando e modelando o problema de forma que seja possível criar um projeto orientado a objetos efetivo (PRESSMAN, 2001). A seguir, são apresentados os modelos de classes como um dos resultados desta fase.

5.1. Adequação ao Meta-modelo da UML

Uma vez que um dos objetivos deste trabalho é adequar ODEd ao meta-modelo da UML, faz-se necessário apresentar sucintamente a estrutura desenvolvida em ODE para tratar modelos e diagramas em conformidade com a UML 1.4 (OMG, 2001). A Figura 5.1 mostra o diagrama de pacotes dessa estrutura.

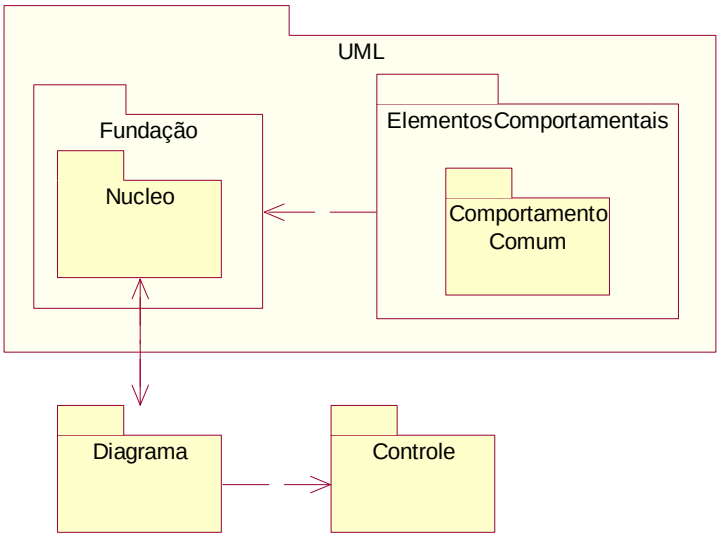

Figura 5.1. Diagrama de Pacotes da Estrutura para Tratar Diagramas e Modelos segundo o Meta-modelo da UML.

O pacote *UML* representa o meta-modelo da UML, sendo que somente os sub-pacotes que são utilizados pelas ferramentas de ODE foram modelados. O pacote *Nucleo*, apresentado na Figura 5.2, contém as classes principais do meta-modelo da UML, referenciadas pela maioria dos diagramas da UML, como *ElementoModelo*, *Classificador*, *Associacao*, dentre outros (SILVA, 2003).

O pacote *Diagrama*, apresentado na Figura 5.3, é o módulo de ligação entre os pacotes *UML* e *Controle*, sendo que este último trata do controle de processos de software no ambiente ODE (MIAN, 2001). O pacote *Diagrama* possui as classes-base para modelagem de diagramas e de modelos de qualquer natureza em ODE.

Além destes pacotes, se fez necessária a criação, em ODE, de mais um pacote do meta-modelo da UML. Trata-se do pacote *ComportamentoComum*, sub-pacote do pacote *ElementosComportamentais*. Este pacote modela classes relacionadas às instâncias de elementos de modelo, como mostra a Figura 5.4.

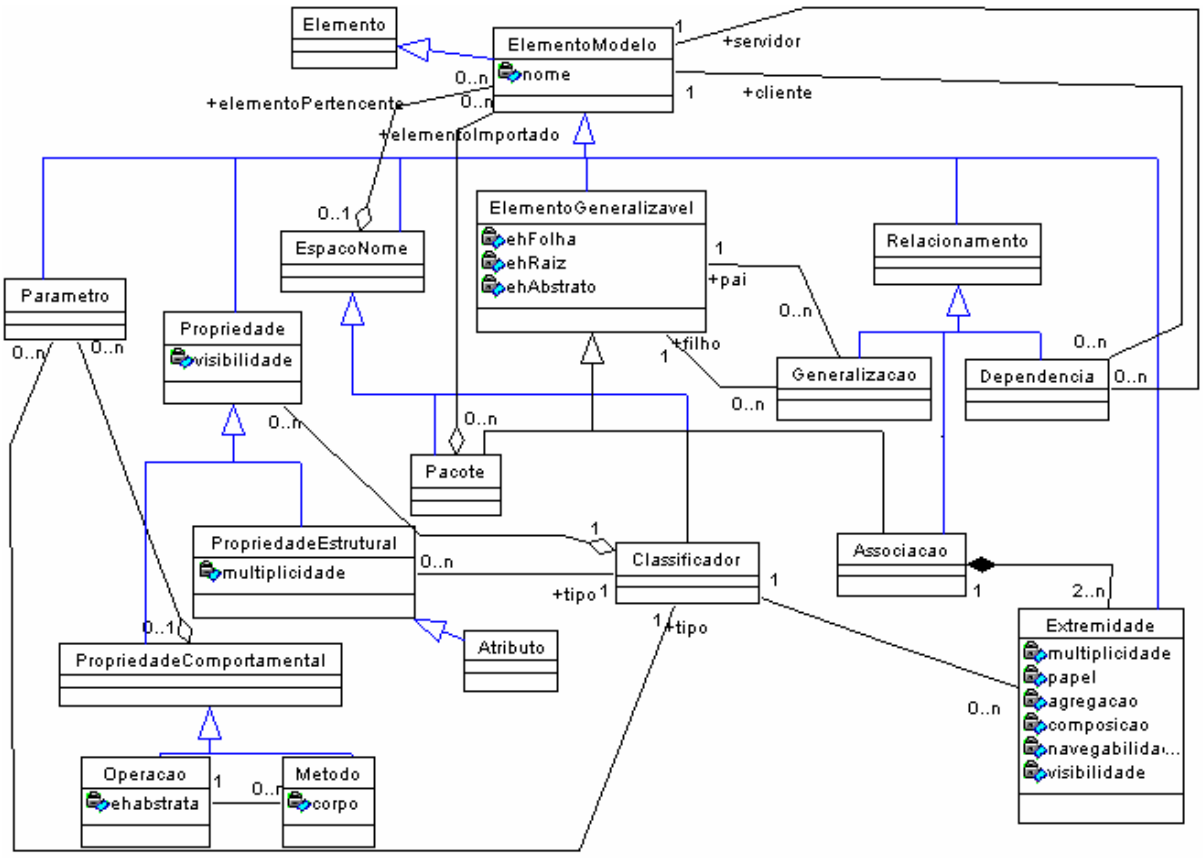

Figura 5.2. Diagrama de Classes do pacote Núcleo (SILVA, 2003).

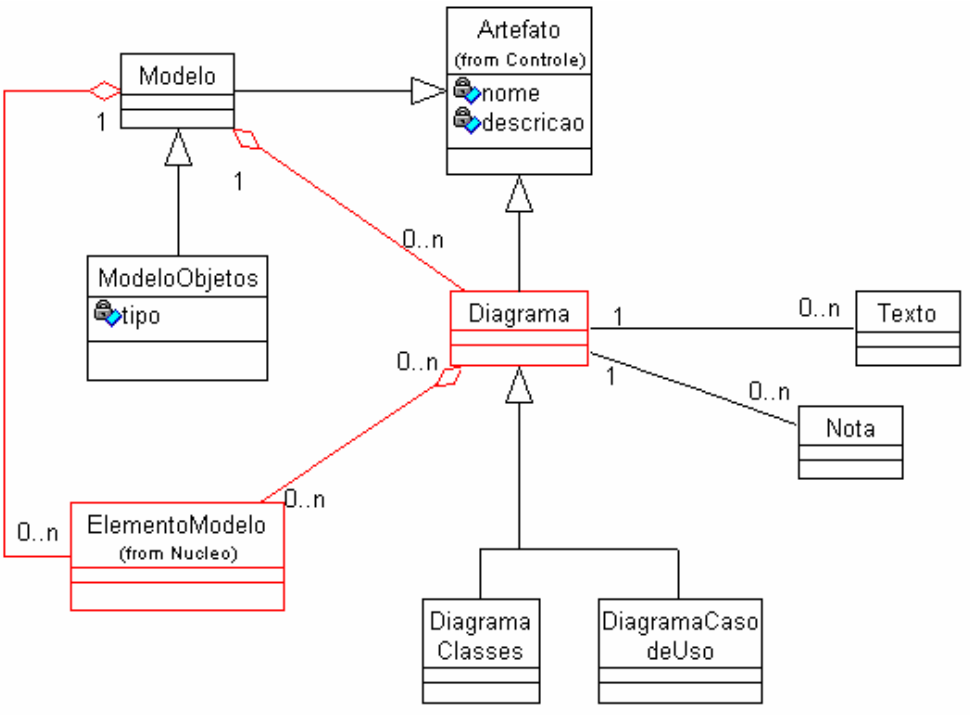

Figura 5.3. Diagrama de Classes do pacote Diagrama (SILVA, 2003).

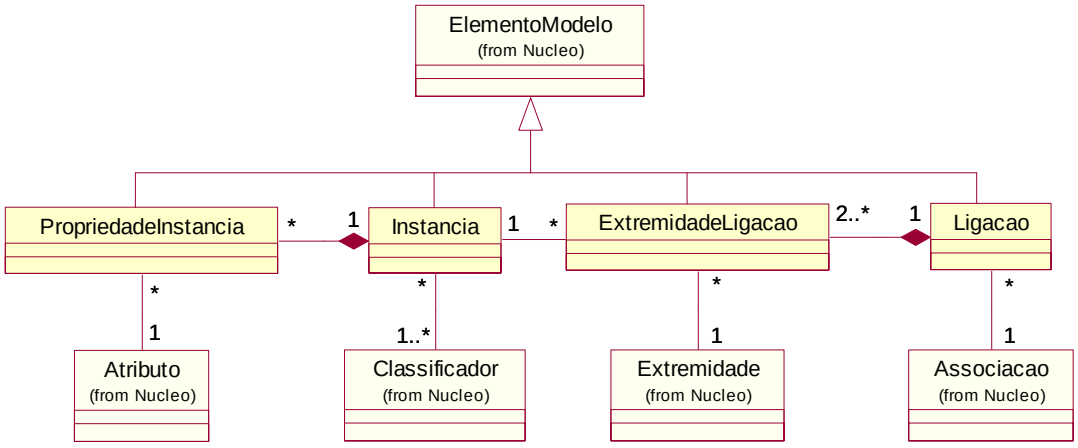

Figura 5.4. Diagrama de Classes do pacote ComportamentoComum.

Devido à quantidade de alterações propostas, ODEd foi remodelado. Algumas classes já identificadas em (MIAN, 2003) foram mantidas, enquanto outras foram excluídas e algumas novas foram criadas. As classes modeladas foram divididas em dois pacotes, *Axioma* e *Ontologia*. O pacote *Ontologia* (detalhado na seção 5.2) reúne as classes necessárias à modelagem de ontologias e dos elementos que a compõem. Esse pacote

utiliza os pacotes *Nucleo* e *ComportamentoComum*, para adequação ao meta-modelo da UML e o pacote *Diagrama* para integração a ODE. Já o pacote *Axioma* (detalhado na seção 5.3) contém as classes necessárias à modelagem de axiomas e usa o pacote *Ontologia*, pois axiomas são compostos por elementos da ontologia. A Figura 5.5 mostra os pacotes utilizados para desenvolver a nova versão de ODEd.

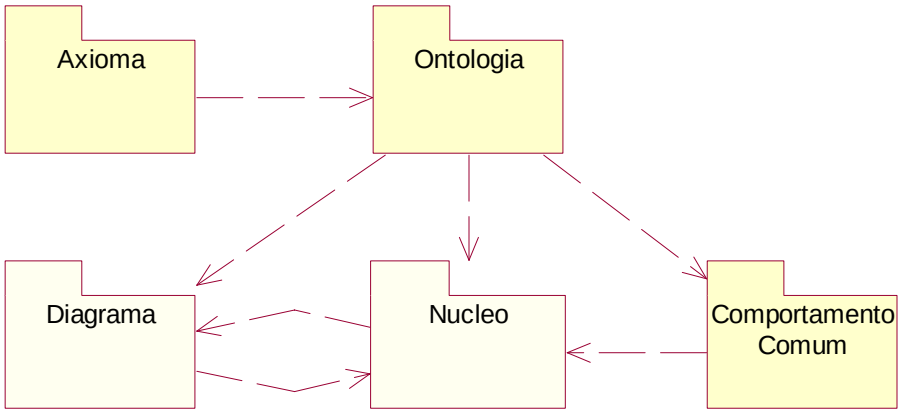

Figura 5.5. Diagrama de Pacotes de ODEd.

5.2. Pacote Ontologia

As classes que modelam ontologias e os elementos que as compõem (diagramas, conceitos, relações, questões de competência, axiomas etc.) foram agrupadas no pacote Ontologia, mostrado na Figura 5.6.

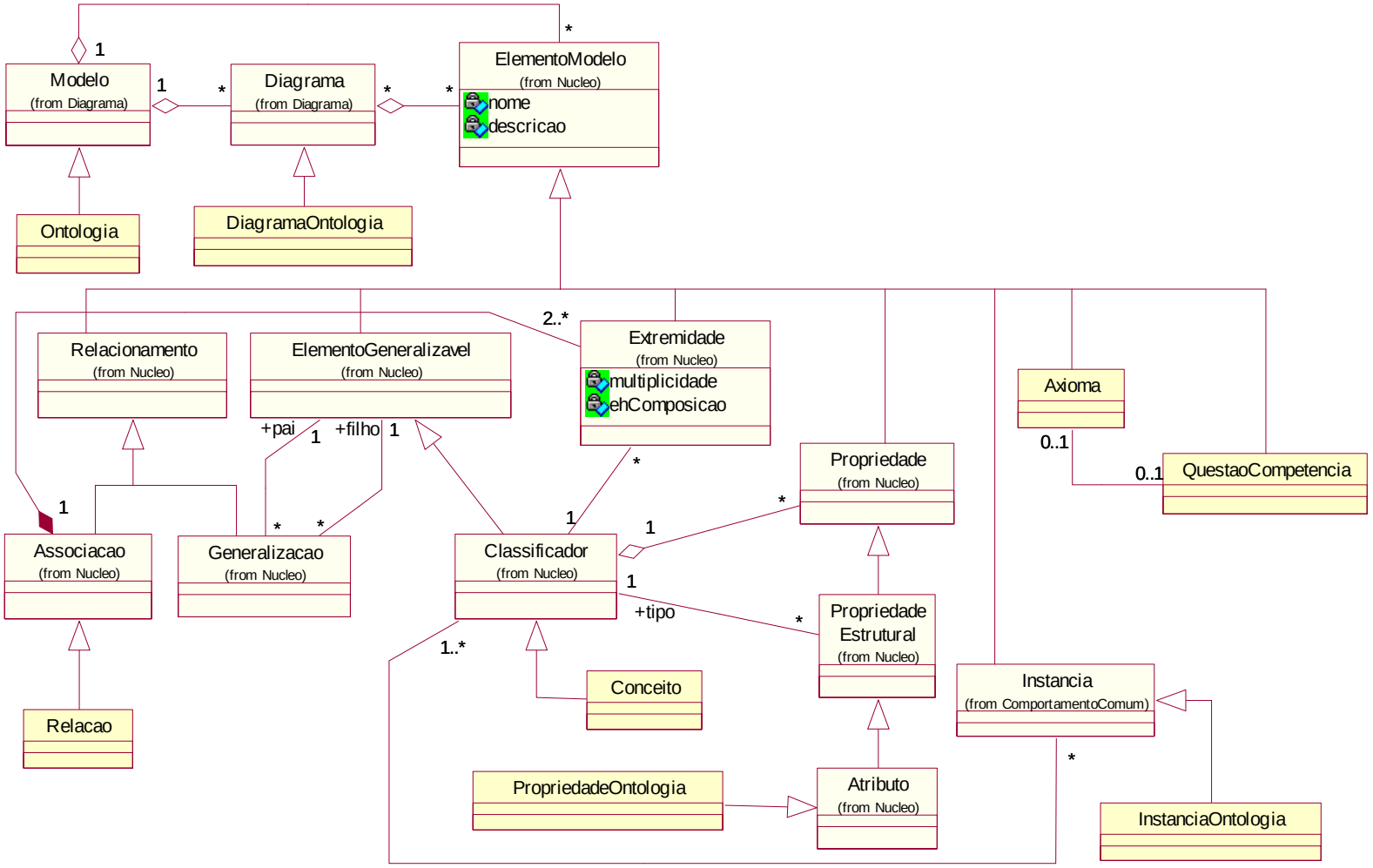

Figura 5.6. Diagrama de Classes do pacote Ontologia

O modelo de ODEd adere à representação de modelos de ODE, que, por sua vez, está em conformidade com o meta-modelo da UML 1.4 (OMG, 2001): uma ontologia é um modelo, um diagrama de ontologia é um diagrama e os elementos que compõem a ontologia são todos elementos de modelo. Através das associações de agregação definidas no pacote *Diagrama*, estabelece-se que os elementos de modelo pertencem a uma e somente uma ontologia, mas podem estar associados a vários diagramas diferentes (e até a nenhum diagrama). Diagramas de ontologia devem, também, pertencer a uma e somente uma ontologia.

No que se refere à adequação ao meta-modelo da UML, conceitos da ontologia são classificadores que, por sua vez, são elementos generalizáveis (formam estruturas hierárquicas) e podem relacionar-se uns aos outros através das extremidades das associações. No nosso caso, estamos interessados numa subclasse específica das associações: as relações da ontologia. Essas herdam as características das associações do meta-modelo da UML sendo, portanto, compostas por duas ou mais extremidades que, como foi dito acima, estão ligadas aos conceitos que elas associam. Devido aos atributos herdados da classe *Extremidade*, é possível registrar se as associações são todo-parte (composições ou agregações) e sua multiplicidade.

Conceitos da ontologia podem possuir propriedades, característica herdada da relação de agregação entre *Classificador* e *Propriedade*. Por fim, questões de competência podem ser formalizadas em axiomas, daí a associação entre essas duas classes que são, ambas, elementos de modelo.

Com relação às instâncias dos conceitos e relações da ontologia, elas são possíveis graças à criação do pacote *ComportamentoComum*, mostrado anteriormente na Figura 5.4. Nesta figura, a classe *Instancia* representa as instâncias de um classificador, enquanto a classe *Ligação* representa as instâncias de uma associação. Duas instâncias podem ser ligadas através de extremidades de ligação e instâncias podem possuir propriedades.

5.3. Pacote Axioma

Para permitir a definição de axiomas, é importante modelar os elementos que os compõem – operadores (conectivos e relações da ontologia), operandos (variáveis,

conceitos da ontologia e instâncias destes) e quantificadores. Tais classes estão representadas no diagrama da Figura 5.7.

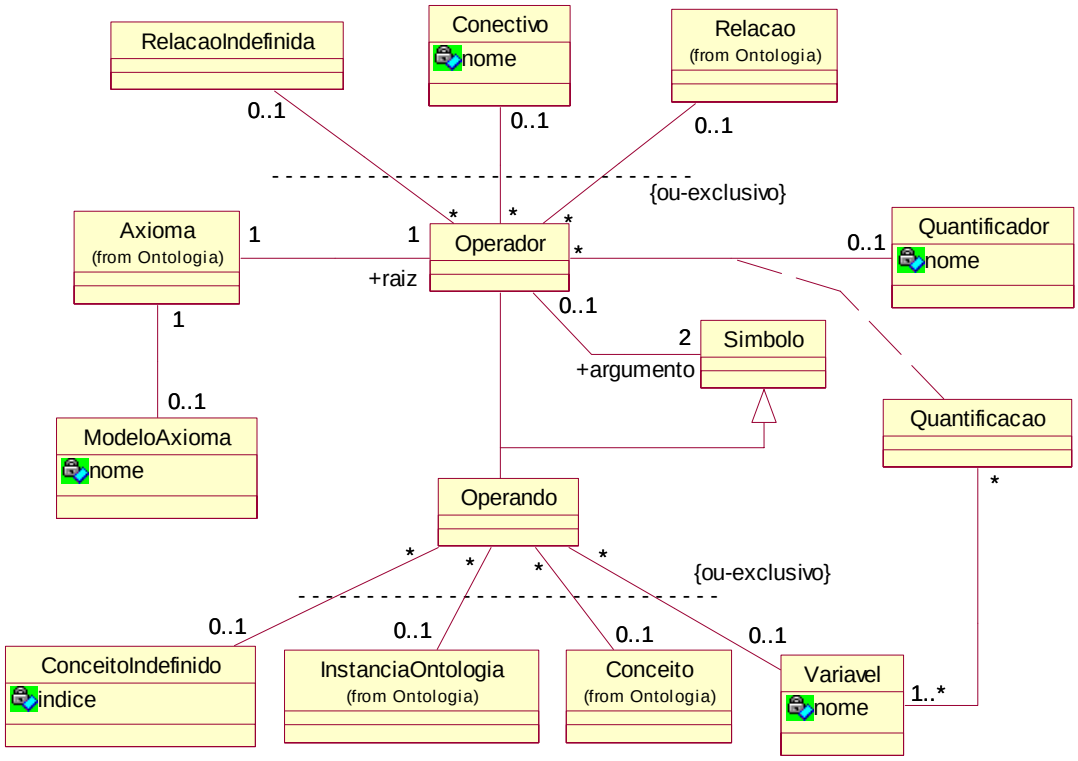

Figura 5.7. Diagrama de Classes do pacote Axioma

Foi escolhida uma representação em árvore para os axiomas em ODEd: a raiz da árvore é um operador, que pode ser um conectivo, uma relação ou uma relação indefinida. Um operador tem sempre dois argumentos, que podem ser outros operadores ou operandos. Operandos, por sua vez, consistem de conceitos, instâncias de conceito, conceitos indefinidos ou variáveis. A Figura 5.8 exemplifica como um axioma pode ser representado por uma árvore binária.

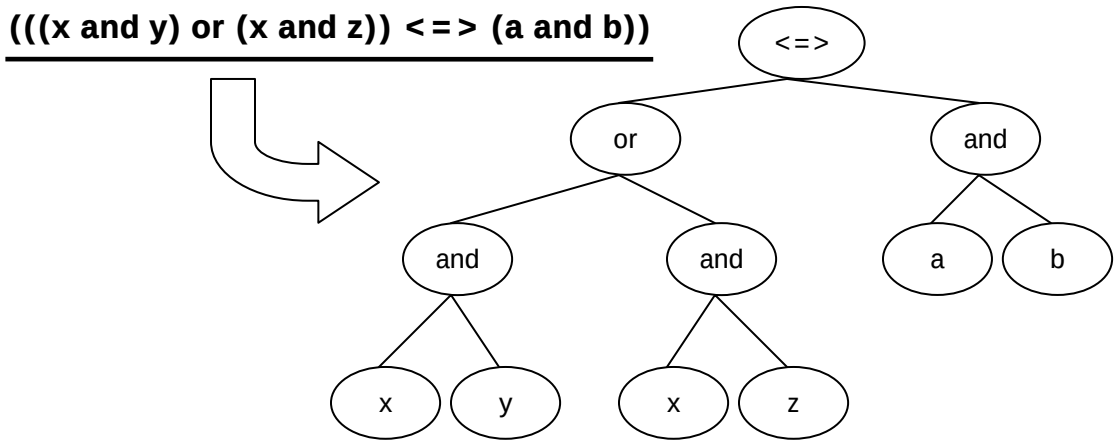

Figura 5.8. Um axioma e sua representação em árvore binária.

Relações e conceitos indefinidos têm como propósito a criação de modelos (*templates*) de axiomas. A criação de modelos de axiomas é uma proposta de evolução de ODEd no que se refere aos tipos de axiomas definidos em (MIAN, 2003). Nesta nova versão, o engenheiro de domínio pode modelar qualquer tipo de axioma como axiomas que deixam os conceitos e as relações em aberto e podem ser aplicados a qualquer relação de qualquer ontologia. Estes *templates* são representados pela classe *ModeloAxioma*.

Por exemplo, a simetria poderia ser modelada pelo axioma: `(C1 R C2) <=> (C2 R C1)`. Neste exemplo, C1 e C2 são conceitos indefinidos e R é uma relação indefinida. Criado este modelo de axioma, o engenheiro de domínio pode associá-lo a qualquer relação de qualquer ontologia. Ao fazer isso, um novo axioma será criado para a ontologia específica, a relação escolhida tomará o lugar da relação indefinida "R" e os conceitos associados pela relação escolhida tomarão o lugar dos conceitos indefinidos C1 e C2.

Por fim, um operador pode ser modificado pelos quantificadores ∀ (para todo) e ∃ (existe, para algum). No exemplo acima, caso o axioma fosse "(∀ x ((x and y) or (x and z))) <=> (a and b))", o operador "or" seria quantificado por ∀ em relação à variável "x".

Capítulo 6

Projeto e Implementação

Na fase de análise desconsideramos aspectos tecnológicos para que não seja tão grande a distância semântica (ou *gap* semântico) entre o que modelamos e o que existe no mundo real. Desta maneira, facilitamos um melhor entendimento do problema e de como deverá ser implementada sua solução. No entanto, antes de avançar para a construção do sistema, é preciso passar por mais uma fase, a de projeto, que traz a análise do sistema para a plataforma computacional na qual o software será implementado. No caso do Projeto ODE e suas ferramentas, tal plataforma é composta pela linguagem de programação *Java* e um banco de dados relacional.

Neste capítulo é apresentado o projeto de arquitetura do sistema (seções 6.1 a 6.7), as fases de implementação e testes são discutidas (seção 6.8) e o protótipo implementado é apresentado (seção 6.9).

6.1. Arquitetura do Sistema

A fase de Projeto do Sistema começa com a definição da Arquitetura do Sistema, que consiste em mapear os conceitos modelados na fase de análise para a plataforma tecnológica de desenvolvimento, adicionando aspectos de implementação e diminuindo o nível de abstração para chegar mais próximo do nível de implementação em uma linguagem de programação.

A primeira atividade da elaboração do projeto de arquitetura do sistema é a divisão das classes em pacotes. Desta forma, são estabelecidos níveis de abstração, que são organizados em camadas e tratados separadamente durante esta fase (FALBO, 2000). As classes de ODEd foram divididas em pacotes tal como mostra a Figura 6.1.

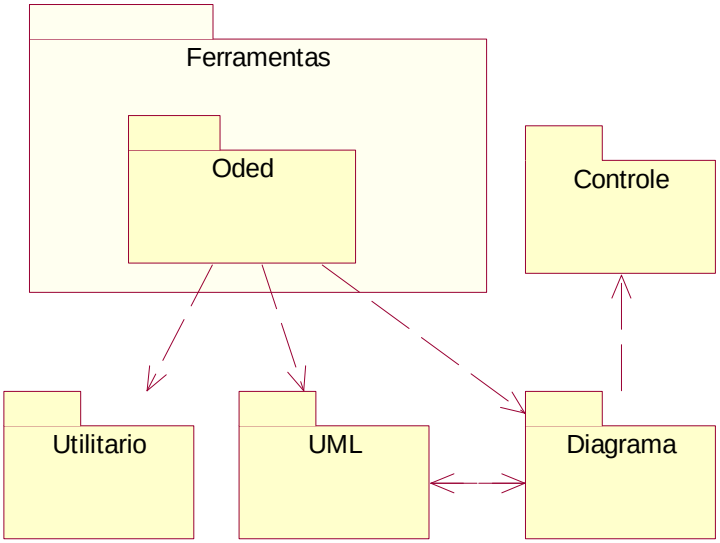

Figura 6.1. Diagrama de Pacotes.

Os pacotes mostrados na figura pertencem ao pacote principal de ODE. As classes referentes a ODEd são agrupadas no pacote *Oded*, sub-pacote do pacote *Ferramentas*. É uma sugestão deste trabalho que os pacotes das ferramentas integradas a ODE sejam agrupados no pacote *Ferramentas* para uma maior organização do sistema.

Os pacotes *UML* e *Diagrama* (SILVA, 2003) contêm, respectivamente, as classes relacionadas ao meta-modelo da UML 1.4 e à construção de modelos e diagramas, que são utilizadas por ODEd. O pacote *Controle* (MIAN, 2001) agrupa as classes que implementam o controle de processos em ODE e, dado que modelos e diagramas são artefatos do processo de software, o pacote *Diagrama* utiliza o pacote *Controle*.

A Figura 6.2 mostra os sub-pacotes de ODEd: *Axioma*, *DiagramaODEd*, *Ontologia*, *Geracao* e *ControleODEd*. Os três primeiros possuem as mesmas características dos pacotes homônimos modelados na fase de análise. O sub-pacote *Geracao* contém classes que implementam os casos de uso do pacote *Geração de Artefatos* e o sub-pacote *ControleODEd* congrega classes relacionadas à execução do editor de ontologias ODEd.

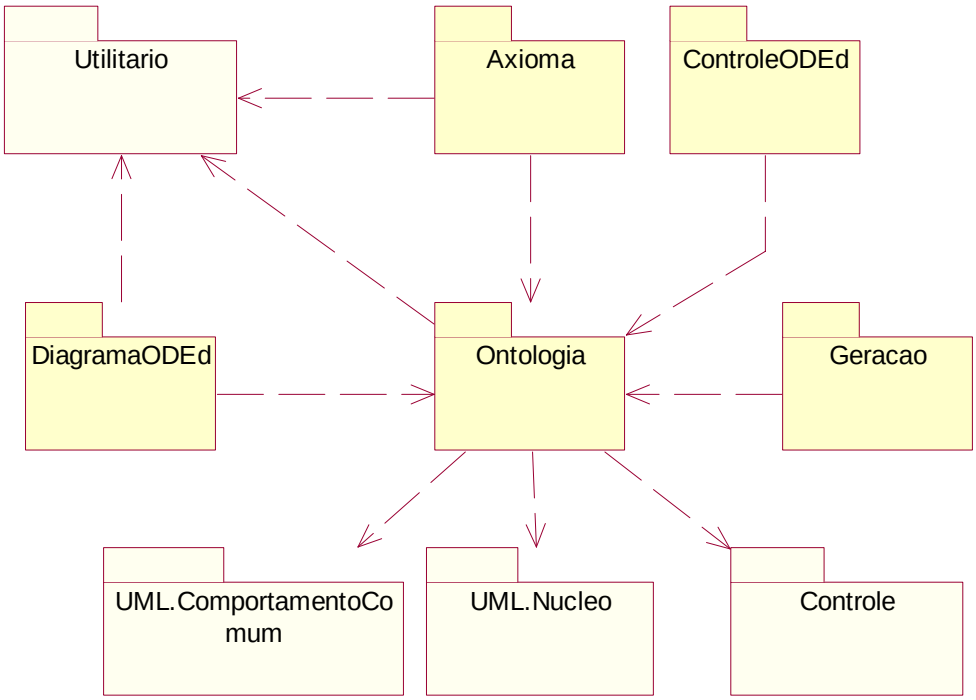

Figura 6.2. Pacote Oded.

Cada um desses pacotes foi, ainda, sub-dividido em quatro camadas (COAD et al., 1993), como mostra a Figura 6.3, a saber:

- **Camada de Domínio do Problema (CDP)**: agrupa as classes que modelam os objetos do mundo real no mundo computacional, também chamados de objetos de domínio ou objetos de negócio, pois definem as regras de negócio do sistema;
- **Camada de Gerência de Dados (CGD)**: detém o acesso aos repositórios de dados, encapsulando o mecanismo de persistência e isolando os impactos da tecnologia de gerenciamento de dados sobre a arquitetura do software;
- **Camada de Gerência de Tarefas (CGT)**: responsável pela execução das funcionalidades do sistema, ou seja, contém classes que executam os casos de uso descritos na fase de especificação de requisitos;
- **Camada de Interação Humana (CIH)**: congrega classes que realizam a interface do sistema com o usuário, tais como janelas, caixas de diálogo, menus, elementos diagramáticos etc.

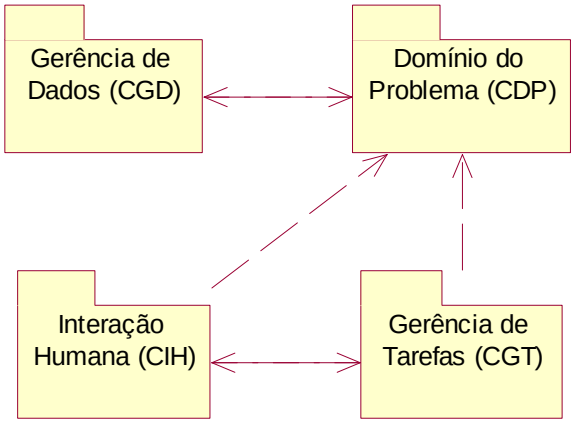

Figura 6.3. Divisão da Arquitetura em Camadas.

A seguir, são detalhados cada um dos sub-pacotes do pacote *Oded*: *Ontologia* (seção 6.2), *ControleODEd* (seção 6.3), *DiagramaODEd* (seção 6.4), *Axioma* (seção 6.5) e *Geracao* (seção 6.6). A seção 6.7 apresenta as classes de utilitário que foram usadas e também as que são contribuições deste trabalho e que poderão ser utilizadas por outras ferramentas de ODE.

6.2. Pacote Ontologia

Conforme apresentado no Capítulo 5, este pacote contém as classes que modelam ontologias e os elementos que as compõem.

6.2.1. Camada de Domínio do Problema

As classes da Camada de Domínio do Problema são aquelas que foram identificadas na fase de análise (Figura 5.6). Porém, sofreram algumas alterações com o objetivo de simplificar a implementação e adequar o modelo às tecnologias utilizadas. Também foram inseridas as navegabilidades entre as classes, restringindo o fluxo de informação entre elas, e os tipos de dados. A Figura 6.4 mostra como ficou o modelo de classes alterado.

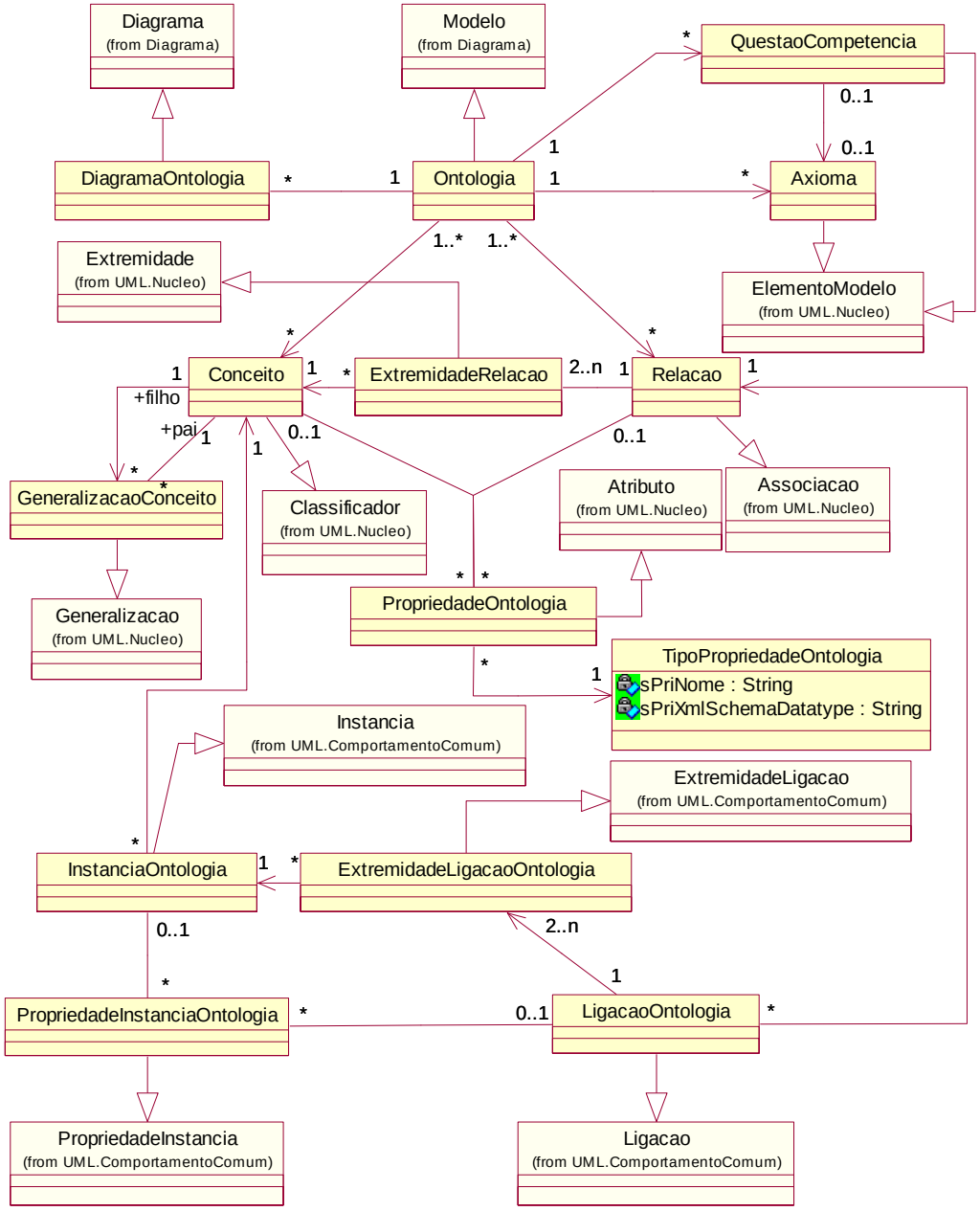

Figura 6.4. Camada de Domínio do Problema do Pacote Ontologia

As classes dos pacotes *UML (Nucleo* e *ComportamentoComum*) e *Diagrama*, mostradas na Figura 6.4 em tons mais claros, são mais detalhadas na Figura 6.5, na qual podemos ver que *DiagramaOntologia* e *Ontologia* herdam as propriedades e métodos de *Artefato*. Voltando ao diagrama de classes da Figura 6.4, mas observando paralelamente o

diagrama da Figura 6.5, pode-se notar que os elementos da ontologia (*Conceito*, *Relacao*, *QuestaoCompetencia* etc) herdam indiretamente de *ElementoModelo*, enquanto *ExtremidadeRelacao* herda as características de *Extremidade*.

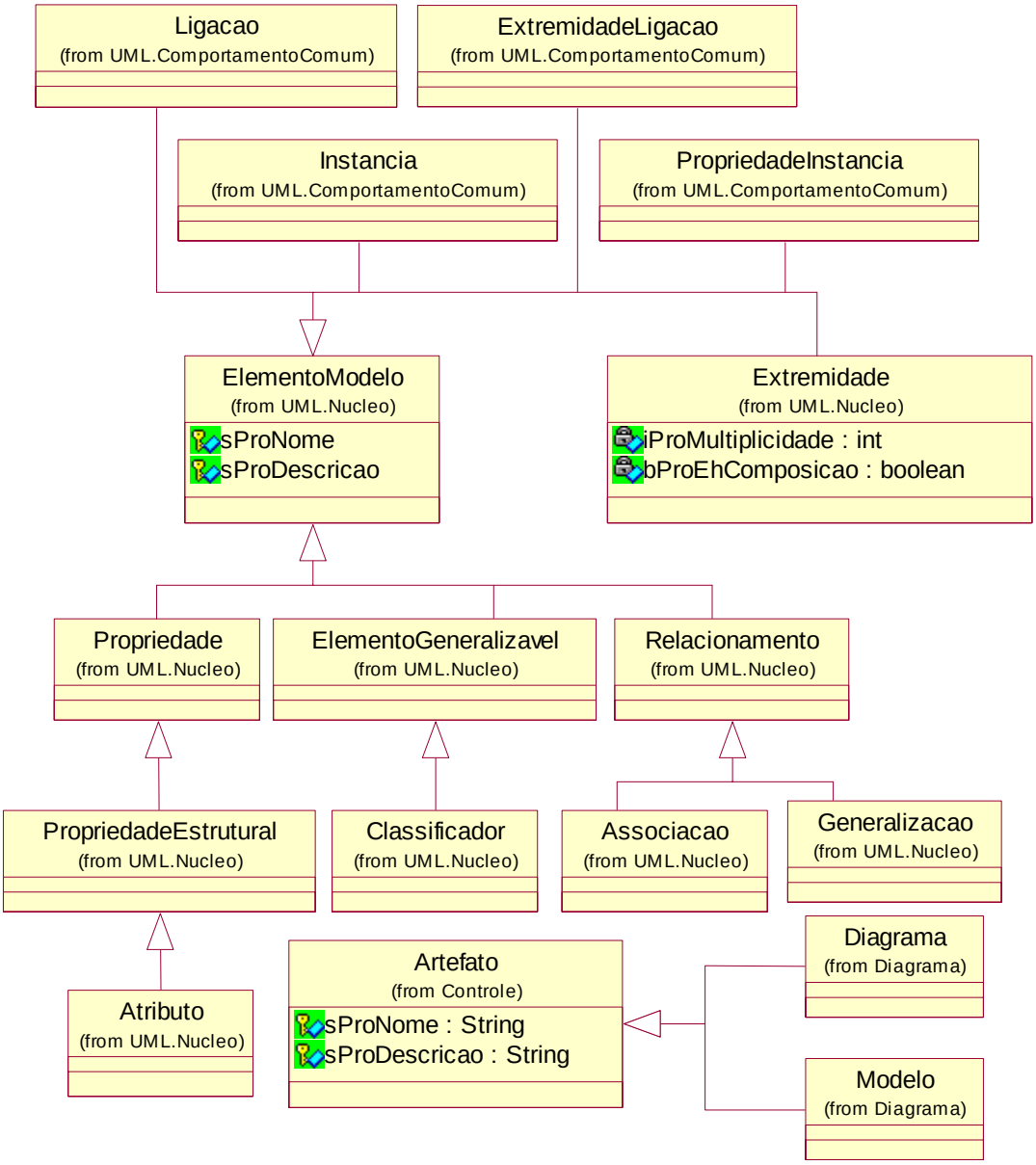

Figura 6.5. Hierarquia entre as classes utilizadas dos pacotes UML, Diagrama e Controle.

Uma primeira grande diferença a se notar entre o modelo de projeto e o de análise é que as associações que as classes do pacote *Ontologia* herdavam das superclasses do pacote *UML* foram trazidas para as subclasses. Esta alteração, que a princípio pode parecer uma

redundância, simplifica bastante a manipulação dos objetos, evitando verificações constantes e conversões de tipos. A tabela 6.1 resume as alterações feitas neste sentido:

Tabela 6.1 – Alterações nas Associações entre Objetos do Domínio do Problema.

Associação no Modelo de Análise	**Associação no Modelo de Projeto**
Modelo — Diagrama	Ontologia — DiagramaOntologia
Modelo — ElementoModelo	Ontologia — Conceito Ontologia — Relacao Ontologia — QuestaoCompetencia Ontologia — Axioma
Classificador — Extremidade — Associacao	Conceito — ExtremidadeRelacao — Relação
Classificador — Propriedade	Conceito — PropriedadeOntologia
Classificador — Instancia	Conceito — InstanciaOntologia
ElementoGeneralizavel — Generalização	Conceito — GeneralizacaoConceito
Associacao — Ligacao	Relacao — LigacaoOntologia
Instancia — PropriedadeInstancia	InstanciaOntologia — PropriedadeInstanciaOntologia
Instancia — ExtremidadeInstancia — Ligação	InstanciaOntologia — ExtermidadeInstanciaOntologia — LigacaoOntologia
PropriedadeEstrutural — Classificador	PropriedadeOntologia — TipoPropriedadeOntologia

Uma outra observação a ser feita é que algumas associações passam a ser computadas indiretamente, como é o caso da associação entre *Diagrama* e *ElementoModelo*. A Figura 6.4 mostra que a classe *DiagramaOntologia*, que herda de *Diagrama*, não possui nenhuma associação com nenhuma das classes que herdam de *ElementoModelo*. Os elementos presentes em um diagrama são calculados a partir dos desenhos de elemento de modelo que o desenho do diagrama contém (os desenhos são apresentados e explicados na seção 6.4.1). Este também é o caso da associação entre *Modelo* e *ElementoModelo*, pois a classe *Ontologia* (que é um *Modelo*) não materializa associações diretas com alguns elementos de modelo, como *ExtremidadeRelacao*, *InstanciaOntologia* e *LigacaoOntologia*, por exemplo. Estas associações são calculadas indiretamente através dos elementos de modelo que estão diretamente associados à *Ontologia* (como *Conceito* e *Relacao*).

Outras alterações que também se fazem notar:

- Na implementação do meta-modelo da UML, um objeto da classe *Instancia* pode possuir ligações com vários Classificadores. Em ODEd, *InstanciaOnotologia* só pode estar associada a um *Conceito*;
- Foi criada uma classe no modelo de ontologias para cada classe utilizada do meta-modelo da UML, a fim de facilitar a manipulação dos objetos;
- O tipo de uma propriedade é, no meta-modelo da UML, um classificador. No modelo de ODEd esta propriedade *tipo* pertence à classe *TipoPropriedadeOntologia*, que não herda de *Classificador* por motivos de simplificação. Os tipos de propriedade que a ontologia irá aceitar serão os tipos compatíveis com o mecanismo de inferência, daí as propriedades *nome* e *xmlSchemaDatatype*;
- Por fim, as características herdadas do meta-modelo da UML e que não fazem sentido num modelo de ontologias são simplesmente ignoradas pela aplicação, que não as utiliza em momento algum.

6.2.2. Camada de Gerência de Dados

A camada de gerência de dados é a responsável por recuperar e armazenar os dados sobre os objetos do domínio em bases de dados persistentes. Em ODEd, foi utilizado um banco de dados relacional, o Microsoft Access, e, portanto, faz-se necessário mapear os conceitos do mundo de objetos para o mundo relacional, a fim de guardar nas tabelas do banco de dados as informações sobre os objetos persistentes.

O processo consiste em traduzir as classes do modelo orientado a objetos para a terceira forma normal padrão. Para cada tabela em terceira forma normal, derivada deste processo de “normalização de objetos”, uma tabela na base de dados é definida (FALBO, 2000), os objetos são transformados em linhas das tabelas, as propriedades em colunas e os relacionamentos são mapeados através de transposição de chaves.

Este processo é documentado através de um diagrama relacional, que mostra as diferentes tabelas do banco, suas chaves, colunas e relacionamentos umas com as outras. O digrama relacional do pacote Ontologia pode ser visto na Figura 6.6.

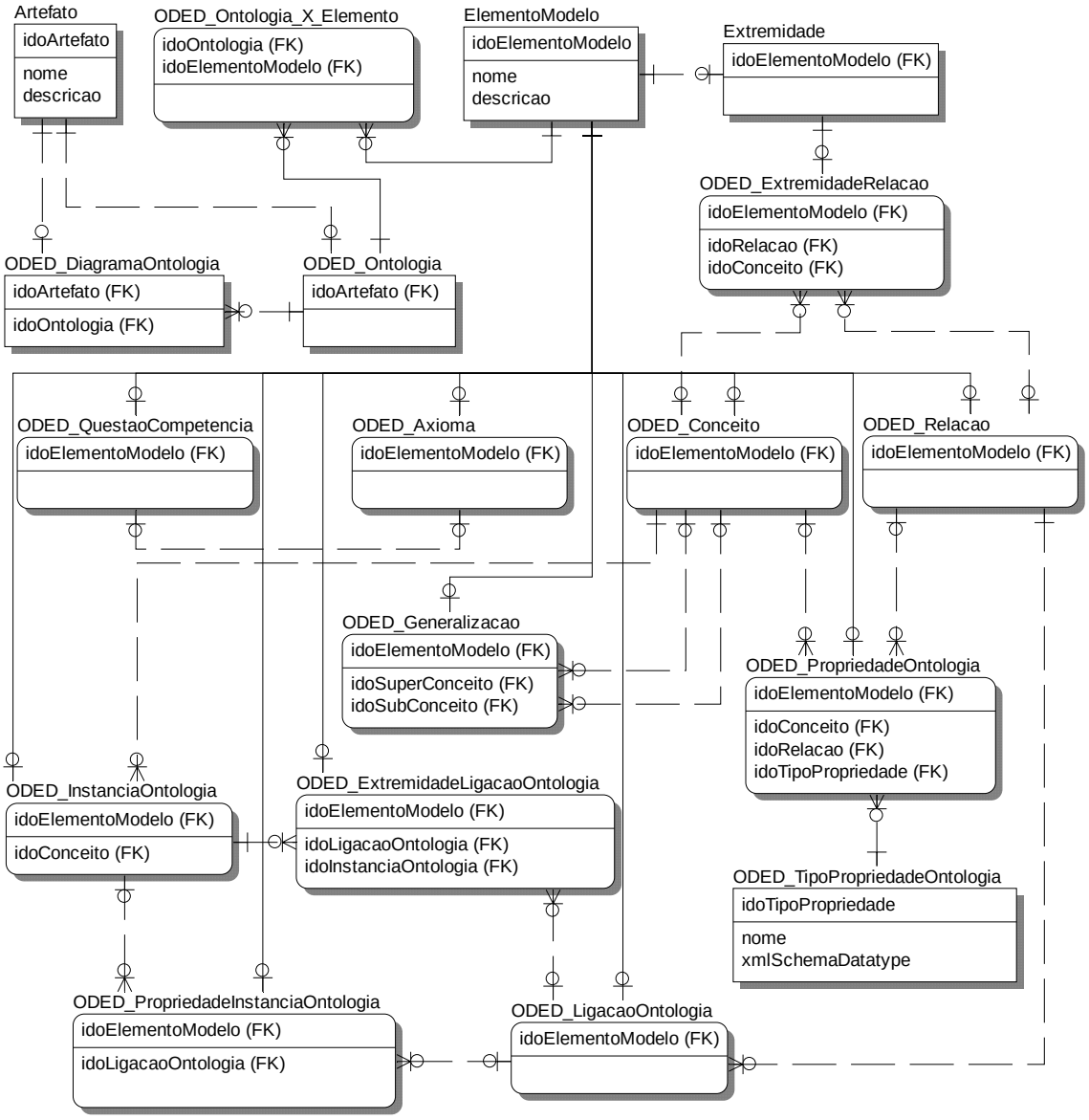

Figura 6.6. Diagrama Relacional do Pacote Ontologia.

Feita a tradução do mundo de objetos para o mundo relacional, é necessário que esta tradução seja implementada no sistema para que o mesmo saiba como recuperar e armazenar objetos no banco de dados relacional. Este, aliás, é o objetivo desta camada de gerência de dados (CGD), que serve de intermediária entre os dois mundos. ODEd utiliza a abordagem definida em (RUY, 2003) para a construção da CGD, que consiste em prover uma *"classe sombra"* para cada classe persistente da CDP.

A classe sombra encapsula as operações do mecanismo de persistência e é responsável pela leitura e gravação de dados dos objetos pertencentes à classe

correspondente, ou seja, ela deve ter implementado em seu código o mapeamento objeto-relacional definido pelo processo de "normalização" dos objetos. RUY apresentou, como resultado de seu trabalho, o pacote utilitário *CP2* que possui grande parte desse mapeamento já codificado, bastando que as classes persistentes herdem de *ObjetoPersistente*, que possui um identificador de objetos para ser usado como chave primária, e as classes sombra sejam subclasses de *ObjetoSombra*. Feito isso, as classes sombra precisam somente implementar os detalhes específicos da persistência da classe correspondente.

A camada de gerência de dados do pacote ontologia consiste, portanto, das classes sombra correspondentes às classes da camada de domínio do problema, como mostra a Figura 6.7.

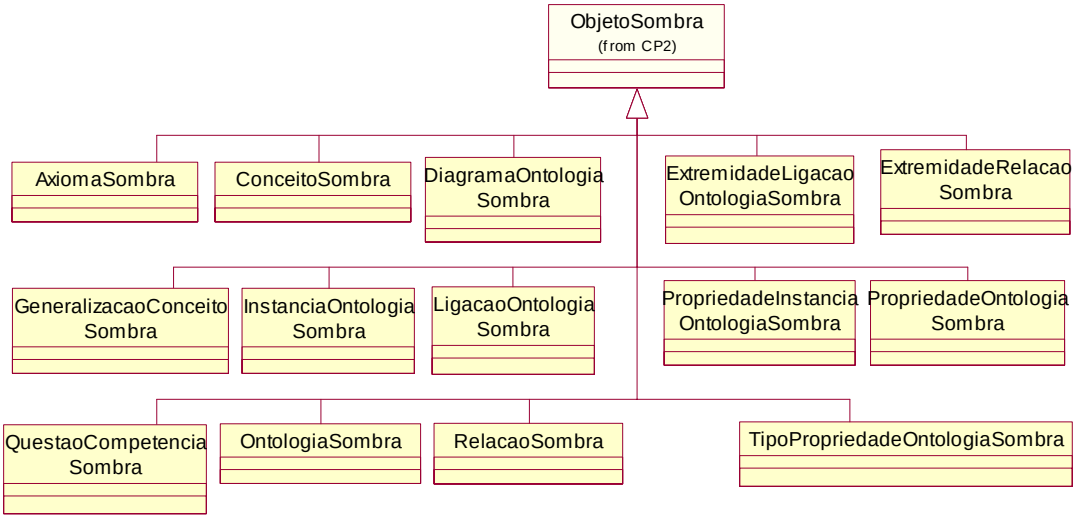

Figura 6.7. Camada de Gerência de Dados do Pacote Ontologia.

6.2.3. Camada de Gerência de Tarefas

A abordagem escolhida para a camada de gerência de tarefas em ODEd é a criação de uma classe de controle para cada caso de uso. Tais classes, também chamadas de Classes de Aplicação, possuem um método para cada cenário do caso de uso que implementa.

No pacote *Ontologia* são reunidas classes de aplicação para todos os casos de uso do pacote *Edição de Ontologias*, definido na especificação de requisitos (Capítulo 4), inclusive os que pertencem aos agrupamentos *Editar Ontologia* e *Avaliar Ontologia*. A Figura 6.8 mostra as classes da camada de gerência de tarefas do pacote *Ontologia*.

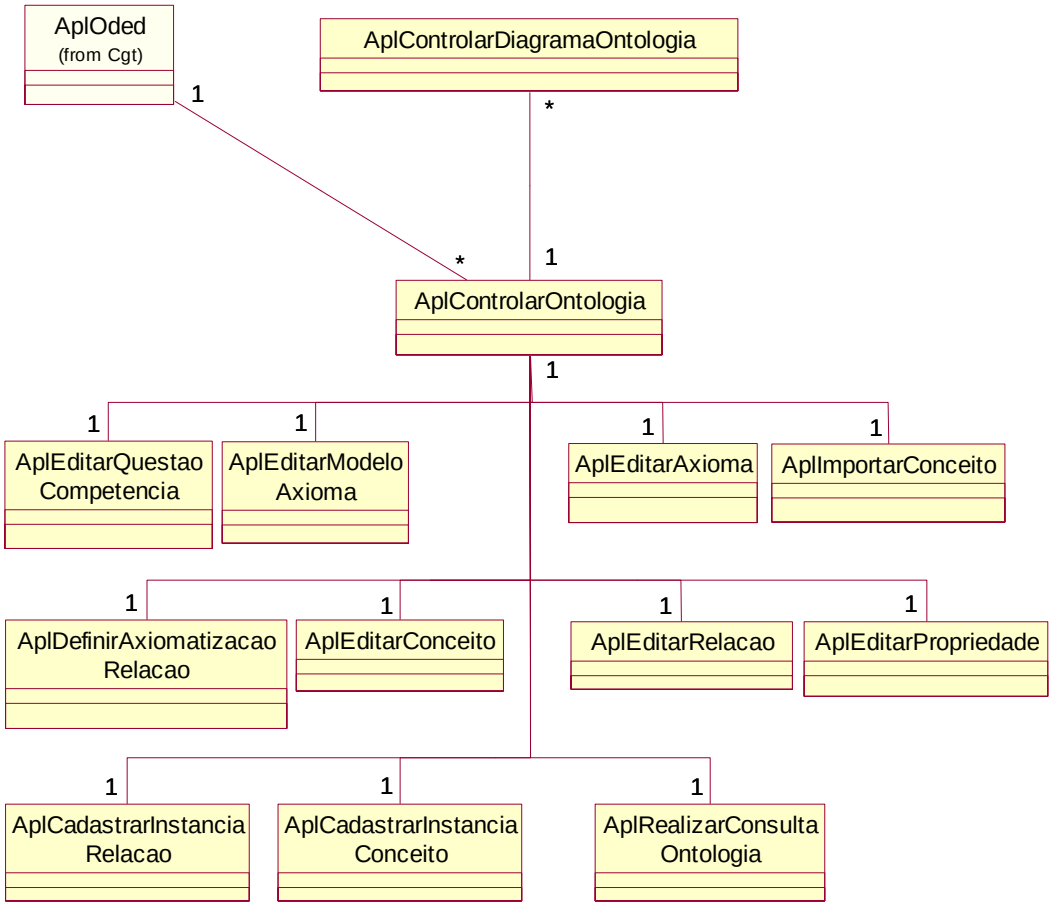

Figura 6.8. Camada de Gerência de Tarefas do Pacote Ontologia.

A classe *AplOded*, do pacote *ControleODEd* (seção 6.3) é a classe principal de ODEd, ou seja, aquela que é executada para abrir o editor. Ela possui associação com várias aplicações *AplControlarOntologia*, uma para cada ontologia correntemente aberta, que, por sua vez, podem possuir ligações com várias *AplControlarDiagramaOntologia*, uma para cada diagrama aberto. Estas duas classes de aplicação registram, respectivamente, a ontologia e o diagrama abertos e, além de realizar os cenários dos casos de uso nestes objetos, ainda controlam as alterações neles feitas para serem desfeitas, caso o usuário não as salve.

As demais classes de aplicação possuem um relacionamento um-para-um com *AplControlarOntologia* e são acessíveis somente através desta última.

Para simplificar o diagrama desta camada, os métodos e propriedades das classes de

aplicação não foram mostrados. Como já foi dito, estas classes possuem métodos que representam os cenários dos casos de uso que representam. A Figura 6.9 mostra, como exemplo, a classe *AplEditarQuestaoCompetencia* detalhada. Os métodos "executar" são responsáveis por ativar a interface com o usuário para a execução de um cenário, enquanto que os demais realizam-nos de fato, efetuando alterações nos objetos do domínio.

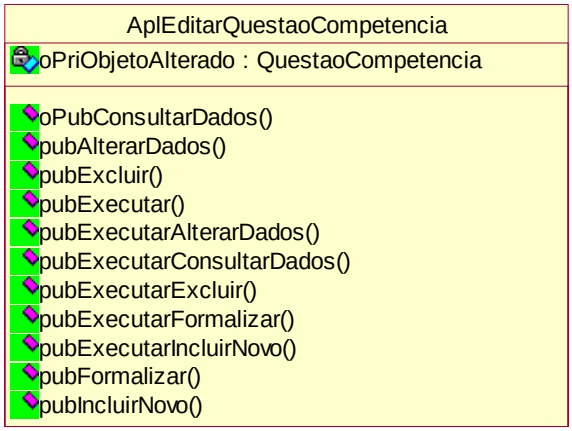

Figura 6.9. AplEditarQuestaoCompetencia em detalhe.

6.2.4. Camada de Interação Humana

Esta camada congrega as janelas, caixas de diálogos, menus etc necessários à interação do sistema com o usuário. São os elementos gráficos que dão acesso às funcionalidades do sistema para o usuário. Mostrar todas as classes envolvidas nesta camada é desnecessário e as figuras abaixo mostram algumas classes que merecem destaque.

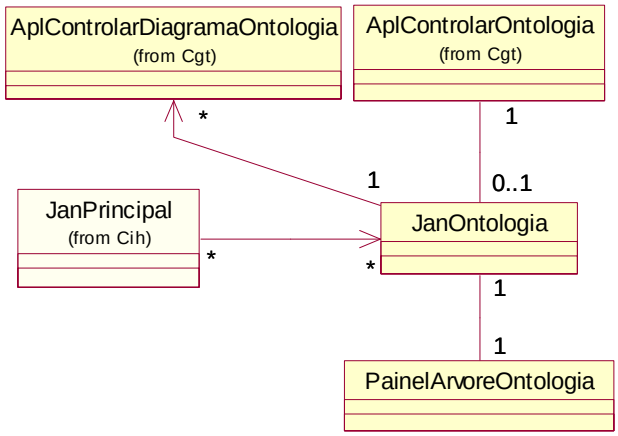

Figura 6.10. Janela da Ontologia

A Figura 6.10 mostra a classe *JanOntologia* e suas relações. Ela implementa a janela que dá acesso a todas as funcionalidades relacionadas à edição de ontologias. Como vemos na figura, a janela principal de ODEd (ver pacote *Controle*, seção 6.3) pode conter várias janelas de ontologia abertas, simbolizando que podemos abrir várias ontologias simultaneamente. A janela de ontologias possui acesso à aplicação de controle de ontologias, para que possa chamar os métodos relativos aos casos de uso de todas as aplicações. Possui também ligação com várias aplicações de controle de diagrama, o que indica que podemos ter vários diagramas de ontologia abertos. A classe *PainelArvoreOntologia* foi incluída também neste diagrama por ser um importante elemento de interface, que mostra os elementos da ontologia hierarquicamente distribuídos em uma árvore e dá acesso a uma série de funcionalidades relacionadas a estes elementos (tais como excluí-los ou consultar seus dados).

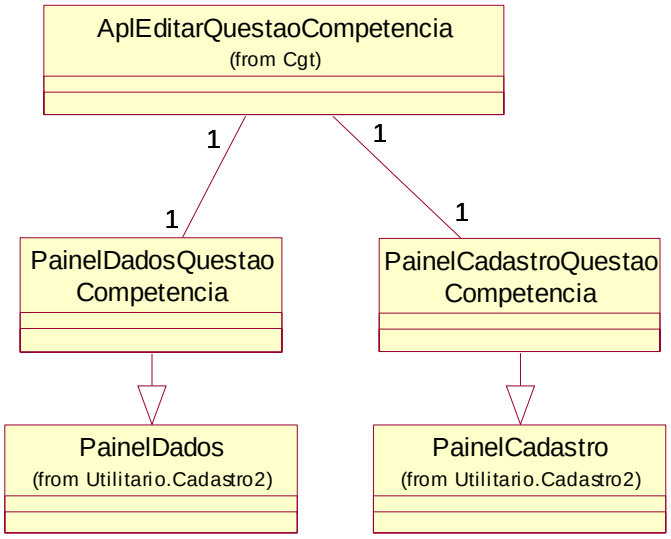

Figura 6.11. Painéis de cadastro de Questão de Competência.

A Figura 6.11 mostra os painéis de cadastro de questões de competência, exemplificando como são implementadas as interfaces gráficas dos casos de uso que permitem o cadastro de objetos de uma forma geral. *PainelCadastroQuestaoCompetencia* lista todas as questões de competência que pertencem à ontologia, enquanto que *PainelDadosQuestaoCompetencia* permite que o engenheiro de ontologias crie, consulte ou altere os dados das questões de competência. Ambas as classes só precisam implementar o que for específico do objeto que tratam, pois o utilitário *Cadastro2* (seção 6.7.3) realiza o

trabalho que é comum a todas as interfaces desse tipo.

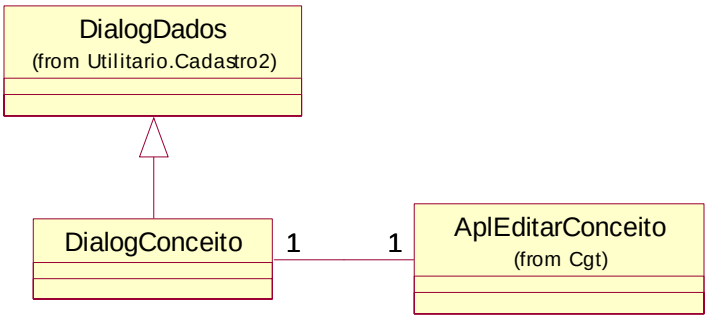

Figura 6.12. Caixa de Diálogo de Propriedades de Conceito

Questões de Competência não são representadas por elementos diagramáticos e, portanto, precisam de toda a interface provida pelos painéis de cadastro e de dados. Já os conceitos e as relações, por exemplo, são criados e excluídos nos diagramas, necessitando somente de uma caixa de diálogo para que o engenheiro de ontologias possa ver e alterar suas propriedades. Tal interface, *DialogConceito*, é mostrada na Figura 6.12 e também se utiliza de uma classe do utilitário *Cadastro2*.

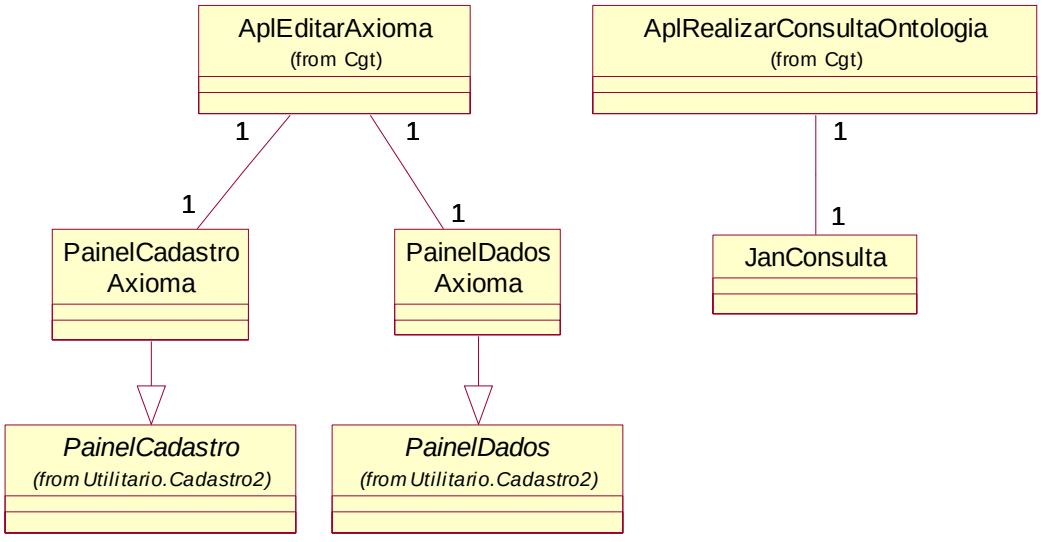

Figura 6.13. Interfaces gráficas relacionadas ao suporte a axiomas.

O suporte à edição de axiomas e a consulta à ontologia são implementados, respectivamente, por *AplEditarAxioma* e *AplRealizarConsultaOntologia*, mostradas na Figura 6.13. A primeira possui interfaces gráficas de cadastro, idênticas às exibidas na Figura 6.11. A segunda possui uma janela que permite que o engenheiro de ontologias faça

uma consulta e receba seus resultados.

6.3. Pacote ControleODEd

O pacote *ControleODEd* tem por único objetivo reunir as classes necessárias para a execução de ODEd, que são, tal como mostra a Figura 6.14, *AplOded* (na camada de gerência de tarefas) e *JanPrincipal* (na camada de interação humana). A primeira é uma classe de aplicação, executável pela máquina virtual Java, que centraliza o acesso a todas as outras classes de aplicação, enquanto a segunda é a janela principal do editor, que se abre quando o mesmo é executado.

Esta janela é do tipo MDI, ou *Multiple Document Interface*, ou seja, ela permite que se abram diversas janelas dentro dela. Isso é possível graças à utilização do utilitário para janelas MDI (seção 6.7.4).

Figura 6.14. Diagrama de Classes do Pacote ControleODEd.

6.4. Pacote DiagramaODEd

Este pacote contém as classes que implementam os elementos diagramáticos, ou seja, os desenhos de diagramas, conceitos, relações, etc. que são usados pelo editor de ontologias para construir, de forma gráfica, a ontologia. Este pacote, portanto, só possui as camadas de interação humana (contendo os desenhos) e de gerência de dados (para persistência dos desenhos).

6.4.1. Camada de Interação Humana

A camada de interação humana do pacote *DiagramaODEd* reúne os elementos diagramáticos, ou seja, os desenhos que representam os elementos da ontologia em diagramas. Para uma melhor visualização, o diagrama de classes dessa camada foi dividido em duas figuras: Figura 6.15 e Figura 6.16.

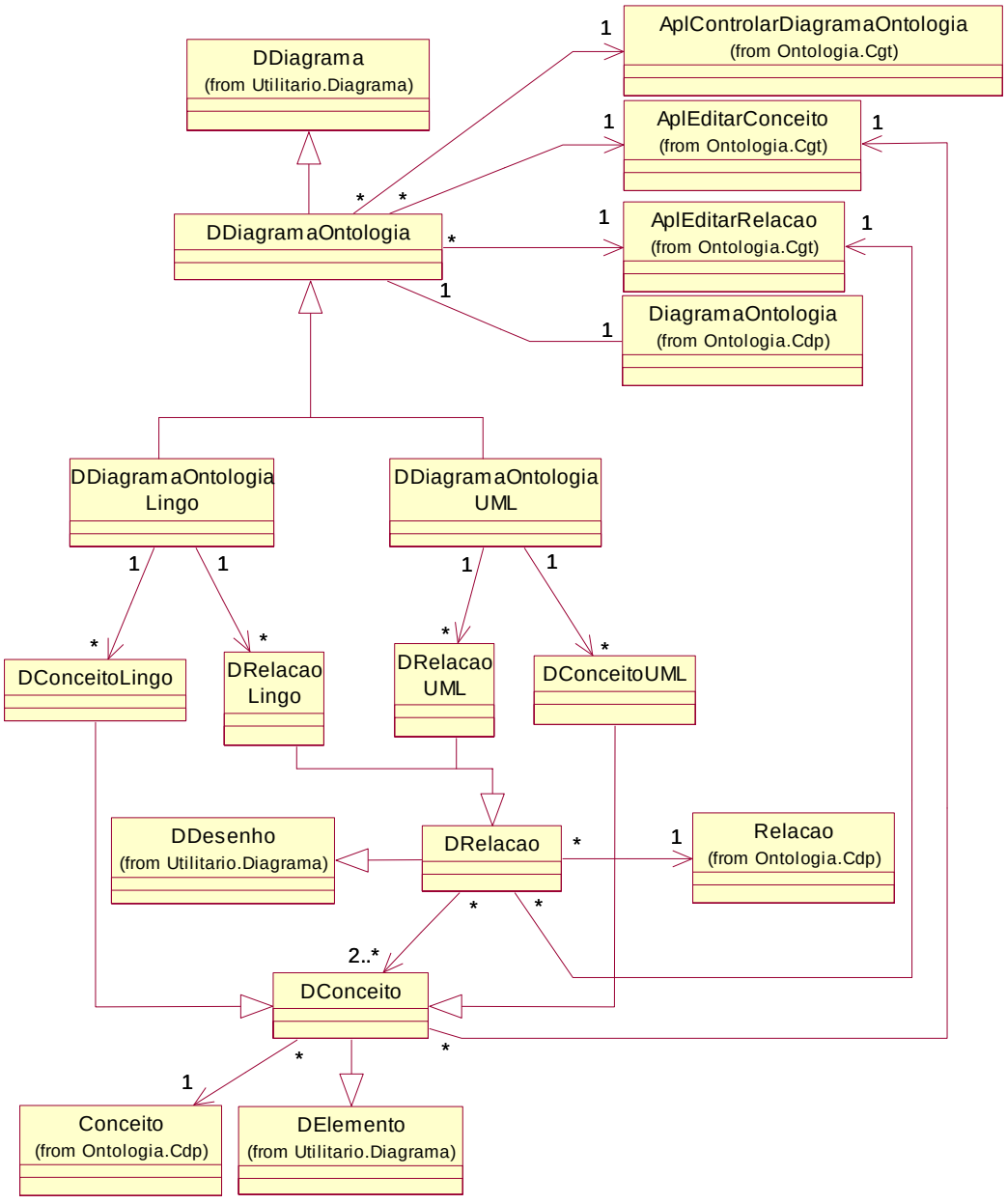

Figura 6.15. Camada de Interação Humana do Pacote DiagramaODEd (parte 1).

Nesta primeira figura, podemos ver a relação entre os elementos diagramáticos e seus correspondentes na camada de domínio do problema, além das relações com as classes de aplicação responsáveis pela manipulação destes objetos de negócio. *DConceito*, por exemplo, possui uma ligação com o *Conceito* que representa e também com *AplEditarConceito*, para que possa acessar métodos relativos ao caso de uso *Editar*

Conceito, como, por exemplo, *Alterar Dados de Conceito*, caso o engenheiro de ontologias faça isso através do desenho no diagrama.

Vemos também nessa figura que esta camada utiliza o pacote de utilitário *Diagrama* (seção 6.7.2) e que há dois tipos de diagrama para representação das ontologias: LINGO (FALBO, 1998) e UML.

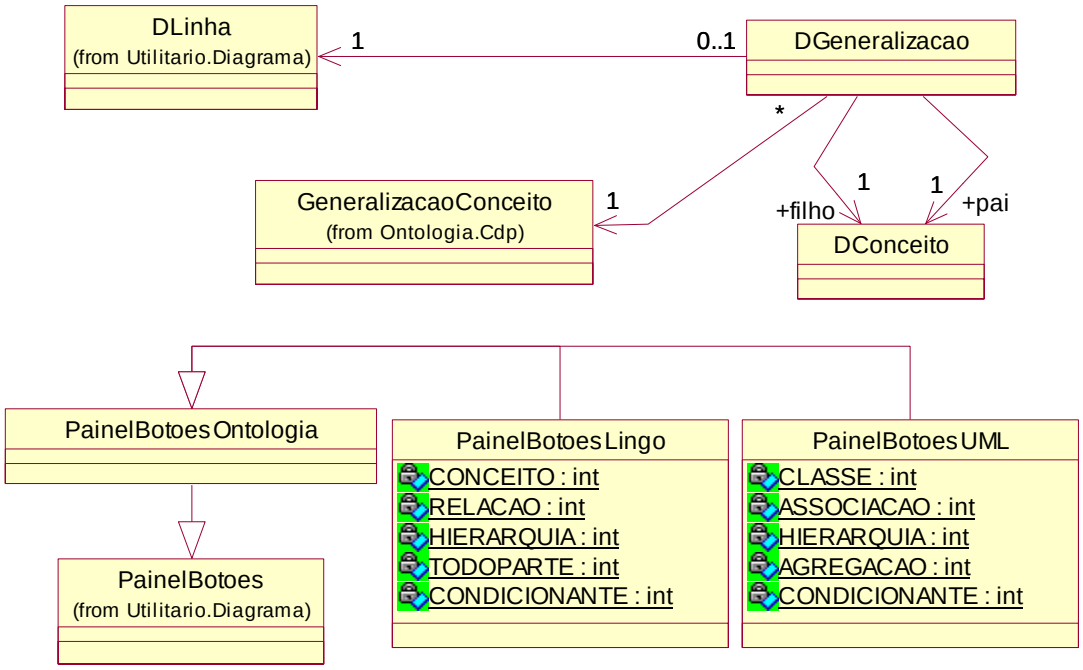

Figura 6.16. Camada de Interação Humana do Pacote DiagramaODEd (parte 2).

Na segunda parte do diagrama, vemos dois elementos diagramáticos que são utilizados tanto pelos diagramas em LINGO quanto pelos escritos em UML, pois possuem a mesma representação diagramática: *DLinha* e *DGeneralizacao*. Vemos, também, os painéis de botões para os diferentes diagramas e suas ferramentas. Os atributos estáticos dos painéis Lingo e UML mostrados no diagrama representam as ferramentas que estão disponíveis para o Engenheiro de Ontologias durante a edição do diagrama.

6.4.2. Camada de Gerência de Dados

Elementos diagramáticos também têm de ser persistidos no banco de dados, pois quando o engenheiro de ontologias abre um diagrama já existente, precisamos recuperar os

desenhos que este diagrama contém, suas posições e relacionamentos uns com os outros. A Figura 6.17 mostra o diagrama relacional desta camada e a Figura 6.18 mostra as classes sombra que compõem o diagrama de classes da CGD.

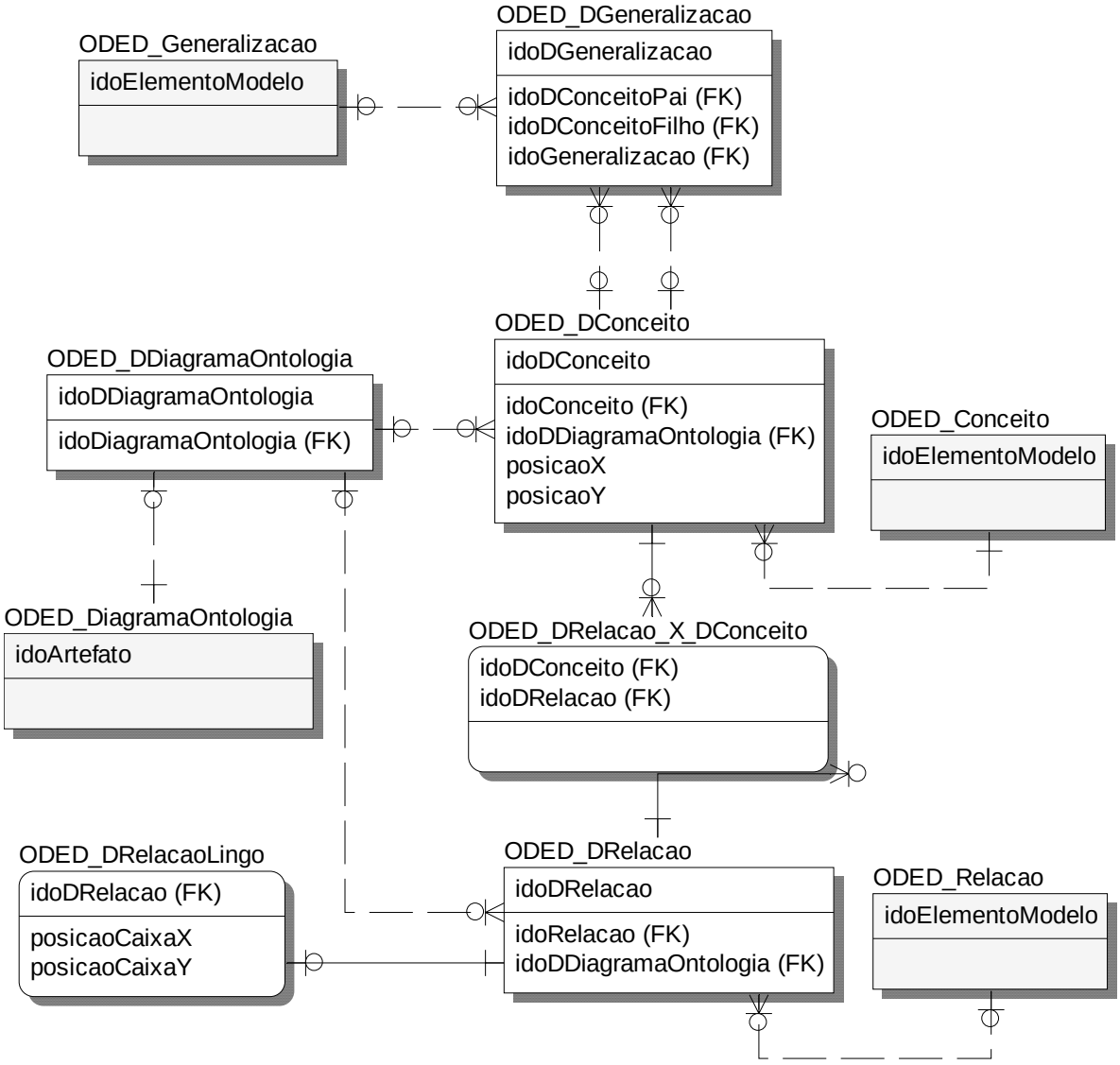

Figura 6.17. Diagrama Relacional do Pacote Diagrama.

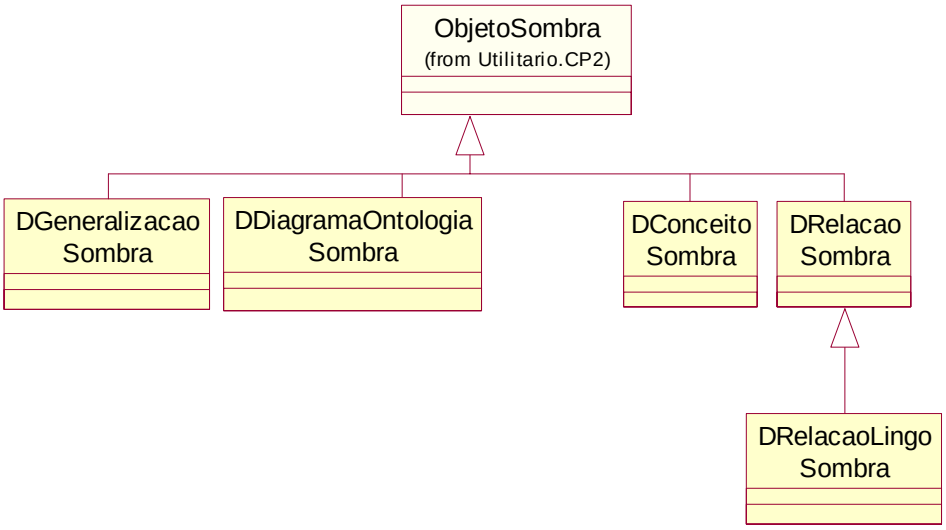

Figura 6.18. Camada de Gerência de Dados do Pacote Diagrama.

6.5. Pacote Axioma

O pacote *Axioma*, responsável pela modelagem dos elementos que compõem os axiomas, sofreu poucas alterações em relação ao correspondente modelo de análise.

6.5.1. Camada de Domínio do Problema

As classes do pacote *Axioma* sofreram somente uma alteração em relação ao modelo de análise. Como podemos ver na Figura 6.19, a classe *Quantificacao*, que era uma classe associativa, se tornou uma classe concreta. Além disso, foram inseridas as navegabilidades e tipos de dados no modelo.

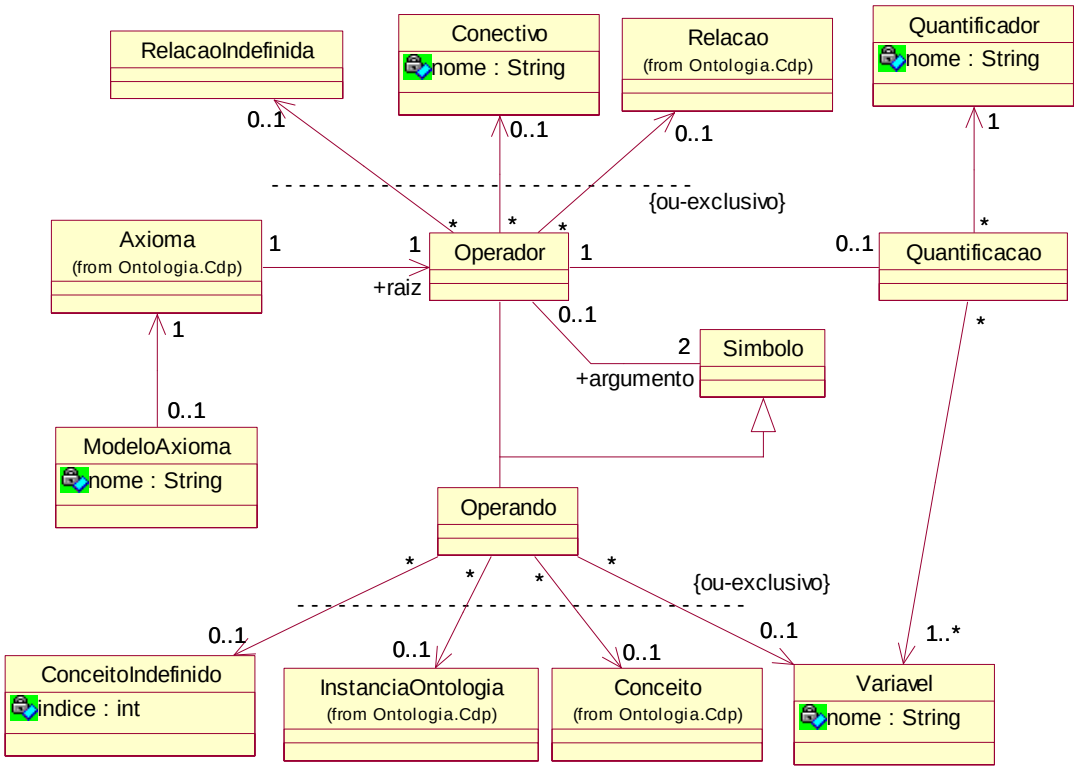

Figura 6.19. Camada de Domínio do Problema do Pacote Axioma.

6.5.2. Camada de Gerência de Dados

A camada de gerência de dados do pacote *Axioma* é similar à do pacote *Ontologia*, provendo uma classe Sombra para cada classe do domínio. A Figura 6.20 mostra estas classes, todas filhas de *ObjetoSombra*, do utilitário de persistência. Já a Figura 6.21 mostra o diagrama relacional do pacote *Axioma*.

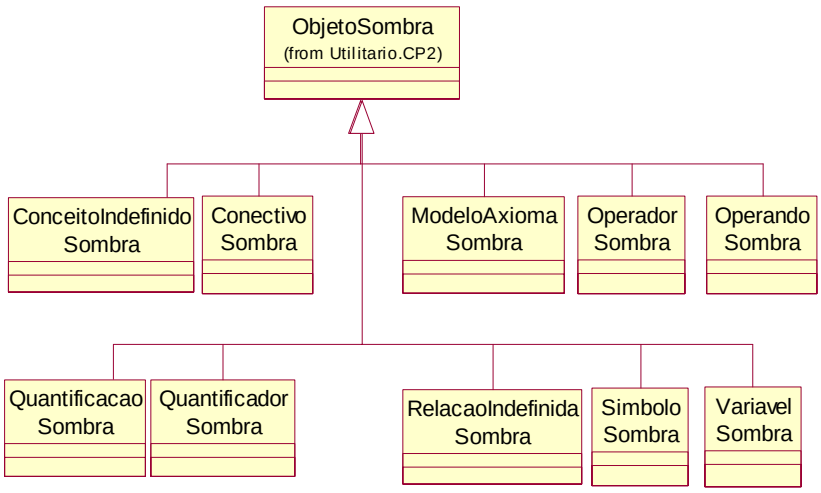

Figura 6.20. Camada de Gerência de Dados do Pacote Axioma.

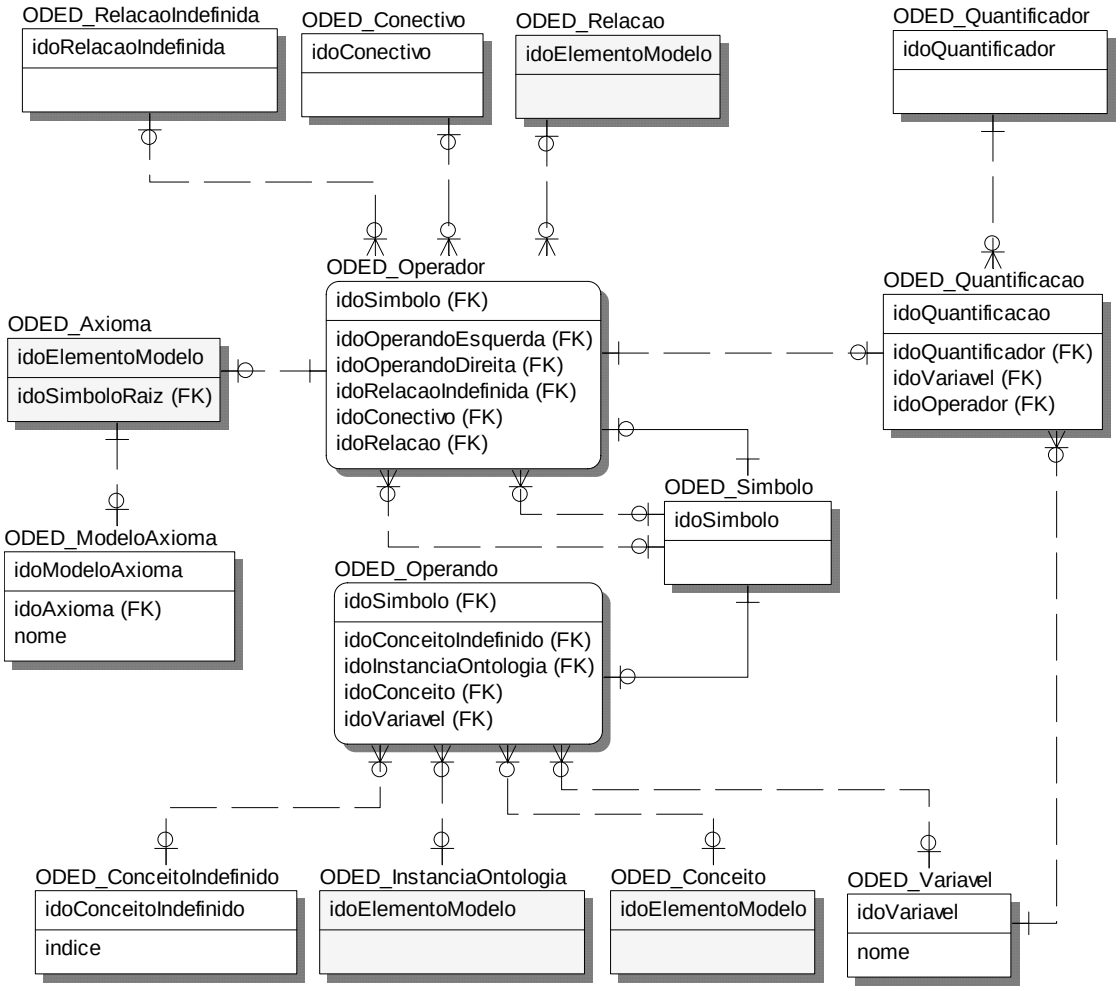

Figura 6.21. Diagrama Relacional do Pacote Axioma.

6.5.3. Integração da Máquina de Inferência

Como foi explicado na seção 2.4, para apoiar a atividade de avaliação de ontologia e para inclusão de conhecimento expressado na forma de axiomas na infra-estrutura de objetos gerada faz-se necessária a integração de um mecanismo de inferência a ODEd, além da escolha de uma linguagem para formalização de ontologias e de axiomas.

Como máquina de inferência foi escolhido JTP (*Java Theorem Prover*) (FIKES et al., 2003); para formalização de ontologias foi escolhida DAML+OIL (*DARPA Agent Markup Language + Ontology Inference Layer*) (CONNOLLY et al., 2001); e para representar axiomas foi escolhida a linguagem KIF (*Knowledge Interchange Format*) (GENESRETH, 1992). Tais escolhas foram justificadas também na seção 2.4.

A máquina de inferência é utilizada em ODEd em dois momentos: durante a avaliação da ontologia (respondendo às consultas do engenheiro de ontologias) e durante a utilização da infra-estrutura de objetos gerada.

Para a avaliação da ontologia, ODEd executa os seguintes passos:

1. Toda a ontologia é convertida para DAML+OIL, utilizando a biblioteca de manipulação de arquivos XML JDOM;
2. O mecanismo de inferência de JTP é ativado e a ontologia é carregada através do método *loadDamlKB(), da classe jtp.context.daml.DamlReasoningContext*. Este método lê a ontologia em formato DAML+OIL a partir do arquivo XML;
3. Os axiomas são, um a um, adicionados ao mecanismo de inferência através do método *tellKifString()*, também da classe *jtp.context.daml.DamlReasoningContext*;
4. O engenheiro de ontologias escreve sua consulta na forma de um axioma KIF com variáveis abertas;
5. A consulta é enviada ao mecanismo de inferência através do método *ask()*, novamente pertencente à classe *jtp.context.daml.DamlReasoningContext*;
6. JTP retorna uma coleção de objetos *jtp.ReasoningStep*, que contém em sua estrutura um mapa (*java.util.Map*) que associa as variáveis abertas da consulta do usuário a nomes de elementos da ontologia.

Já a integração do mecanismo de inferência com a infra-estrutura de objetos gerada se

dá através de chamadas a JTP inseridas na implementação dos métodos correspondentes às relações da ontologia. Cada vez que um método desses é chamado, ele deve verificar se a relação à qual ele corresponde não possui restrições, na forma de axiomas, na ontologia. Caso possua, deve ser realizada uma consulta à ontologia e o resultado dessa consulta deve ser adicionado ao resultado normal do método.

Por exemplo, suponha a ontologia da Figura 6.27, onde temos três conceitos: *Usuario*, *Tipo* e *Conhecimento*. Temos as relações *Ser*, entre *Usuario* e *Tipo*; *Saber*, entre *Usuario* e *Conhecimento*; e *Requerer*, entre *Tipo* e *Conhecimento*. Ao ser gerada a infra-estrutura de objetos, a classe *Usuario* possuirá um método *obterSer()*, que irá retornar todos os conhecimentos que aquele usuário sabe. Suponhamos, agora, que exista um axioma que diga: "Se um usuário U é de um tipo T e o tipo T requer o conhecimento C, então U automaticamente sabe C". Ao executar o método *obterSer()*, além de retornar os conhecimentos diretamente ligados ao usuário através da relação *Saber*, a infra-estrutura deve retornar também aqueles que estão associados ao *Tipo* do *Usuario*, através da relação *Requerer*.

6.6. Pacote Geracao

O pacote *Geracao* contém as classes de aplicação e interface com o usuário que implementam as funcionalidades do pacote *Geração de Artefatos*, identificado na fase de especificação de requisitos.

6.6.1. Camada de Gerência de Tarefas

Assim como no pacote *Ontologia*, esta camada de gerência de tarefas possui uma classe de aplicação para cada caso de uso identificado, como mostra a Figura 6.22.

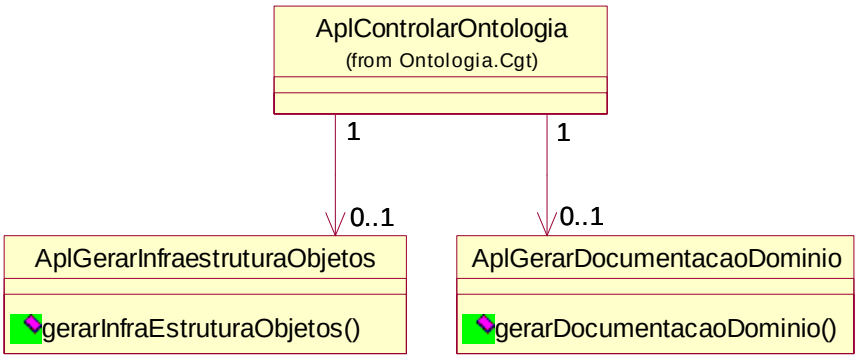

Figura 6.22. Camada de Gerência de Tarefas do pacote Geracao.

6.6.2. Camada de Interação Humana

Os elementos gráficos necessários para efetuar os casos de uso de *Geração de Artefatos* são simples e são utilizados objetos da própria API Java, como itens de menu (*JMenu, JMenuItem*) e caixas de diálogo de seleção de arquivos (*JFileChooser*). Nenhuma classe de interação humana foi criada para o pacote *Geracao*.

6.7. Pacote Utilitario

Para construção do protótipo de ODEd foram utilizados vários pacotes de utilitários. Alguns destes pacotes já estavam prontos, como resultados de outros trabalhos do projeto ODE, e alguns foram elaborados durante este trabalho e ficarão como contribuição para que outras pessoas façam reuso de seu código. As seções abaixo descrevem sucintamente cada um dos pacotes utilizados.

6.7.1. Camada de Persistência II (CP2)

A camada de persistência II, desenvolvida em (RUY, 2003), foi utilizada neste trabalho para persistência dos objetos. Sua utilização já foi tratada na seção 6.2.2, no entanto, novas classes foram criadas e é uma proposta deste trabalho incorporar estas classes na camada de persistência oficial de ODE. A Figura 6.23 mostra as duas classes que foram adicionadas: *JLayeredPanePersistente* e *ControleExclusoes*.

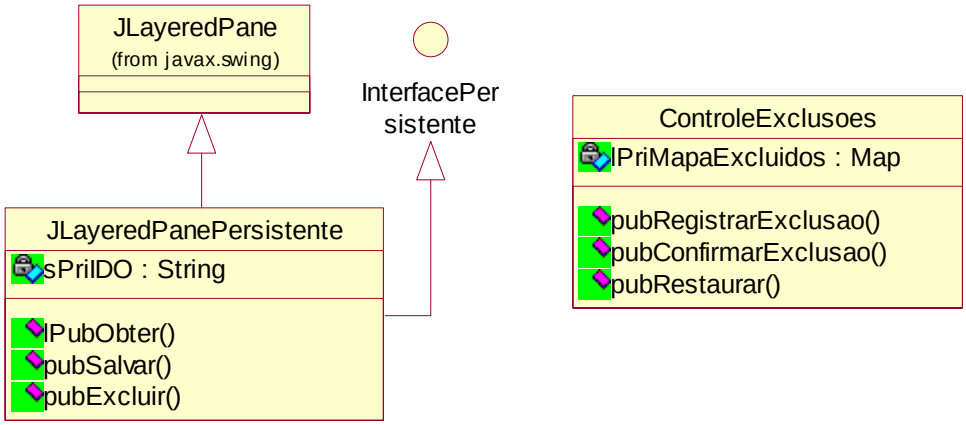

Figura 6.23. Classes adicionadas por este trabalho à Camada de Persistência.

JLayeredPanePersistente foi criada para permitir que os diagramas de ODE, que são *JLayeredPane*s, pudessem ser integrados ao mecanismo de persistência. Essa abordagem já havia sido utilizada por outras ferramentas para persistência de diagramas que estendem *JPanel*, criando a classe *PainelPersistente*, que pertence ao utilitário *CP2*. A criação de uma nova classe se deu porque os diagramas de ODEd precisavam estender *JLayeredPane*, e não *JPanel*.

Já a classe *ControleExclusoes* foi criada a partir de uma necessidade de ODEd de registrar quais elementos da ontologia foram excluídos para, caso o usuário não salvasse a ontologia, os objetos pudessem ser restaurados. Como essa abordagem não se aplica somente a ontologias, mas a qualquer objeto que precisa ser salvo para que as alterações sejam persistidas, é proposta uma classe de utilitário para auxiliar neste caso.

6.7.2. Framework para Diagramas

Ao iniciar o trabalho com os desenhos de conceitos, relações e demais elementos de modelo, percebeu-se que seria interessante criar classes genéricas que implementassem funcionalidades relacionadas à criação de diagramas. Dessa percepção nasceu o *framework* para diagramas, mostrado na Figura 6.24.

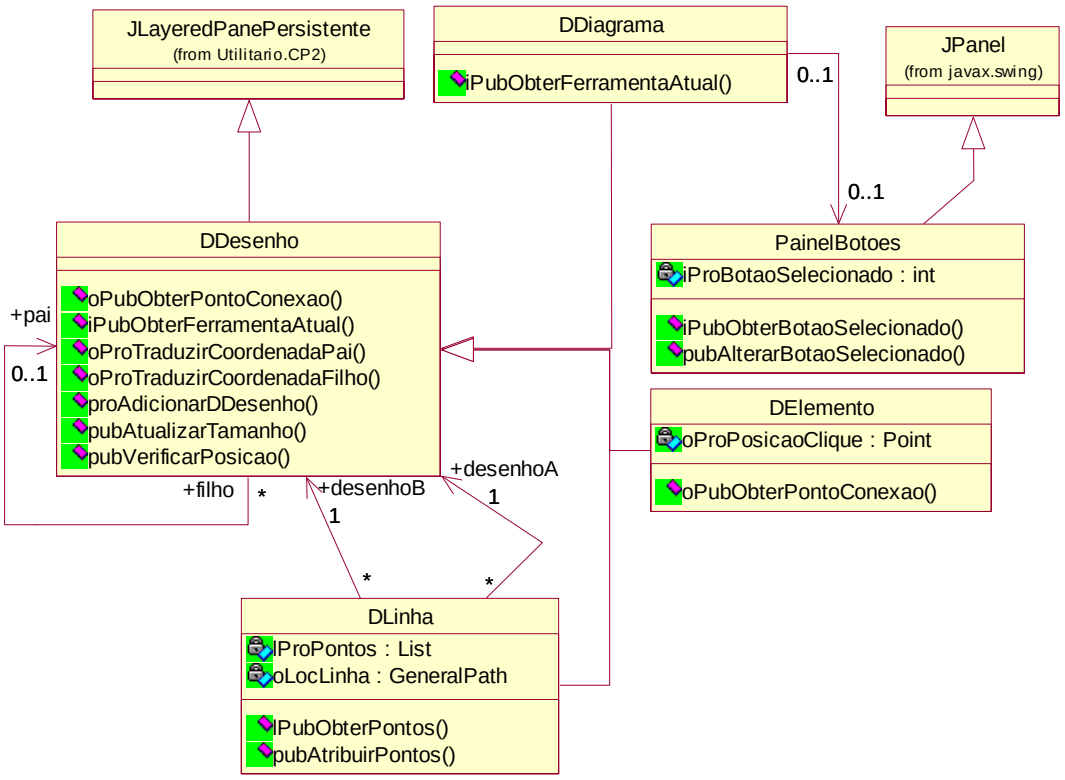

Figura 6.24. Diagrama de Classes do Utilitário Diagrama.

Este *framework* é baseado em *Swing* e no utilitário *CP2* (seção 6.7.1). A classe *DDesenho* implementa a funcionalidade comum a todos os desenhos, inclusive ao diagrama. Desenhos podem ser organizados hierarquicamente, de forma que um desenho possa ser composto de outros desenhos. A classe ainda traz métodos para se redesenhar ao mudar de tamanho ou posição, para traduzir coordenadas relativas ao seu pai ou aos seus filhos e para obtenção do ponto de conexão, que é de onde partem as linhas que conectam este desenho a outros. *DDesenho* é um elemento gráfico persistente, por isso estende *JLayeredPanePersistente*.

Como subclasses de *DDesenho* estão algumas classes de desenho específicas, como *DDiagrama*, que implementa um diagrama, ou seja, um desenho feito para conter outros desenhos e que possui um painel de botões, que são as ferramentas do diagrama; *DElemento*, que representa um desenho que pode ser movido; e *DLinha*, que representa uma linha que conecta dois desenhos.

Uma das características importantes dessas classes é o tratamento aos eventos de

mouse gerados pelo *framework Swing*. Eventos como o clique do mouse ou a simples passagem do mesmo por cima dos desenhos gera chamadas a métodos de tratamento de eventos de *mouse*. Duas coisas são importantes em relação a este assunto: primeiro, alguns desenhos (como por exemplo, *DLinha*) são compostos por grande quantidade de espaço "vazio" que faz parte do elemento gráfico. Portanto, o desenho deve especificar, através do método *contains()*, quais *pixels* realmente pertencem ao desenho e quais não pertencem, para que o *Swing* realize corretamente as chamadas de métodos. Segundo, os desenhos, via de regra, só devem tratar eventos de *mouse* se a ferramenta escolhida for a seta. Caso contrário, devem repassar o evento para o seu ancestral direto, até que chegue ao *DDiagrama*, que é quem normalmente trata os eventos gerados por outras ferramentas.

6.7.3. Framework para Cadastros II (Cadastro2)

Ao implementarmos sistemas, muitas vezes nos deparamos com casos de uso muito similares de cadastro de objetos. Muitos deles são implementados através de uma interface gráfica que lista os objetos existentes e outra que permite que sejam consultados ou editados os dados relativos ao objeto a ser cadastrado. Para promover o reuso de interfaces gráficas, foi criado um *framework* para cadastros, que se encontra em sua versão II, por já ter sido construído anteriormente dentro do projeto ODE, mas ter sofrido algumas alterações neste trabalho. A Figura 6.25 mostra as classes que compõem este pacote utilitário que, como podemos ver, são baseadas no *framework* gráfico *Swing* da linguagem Java.

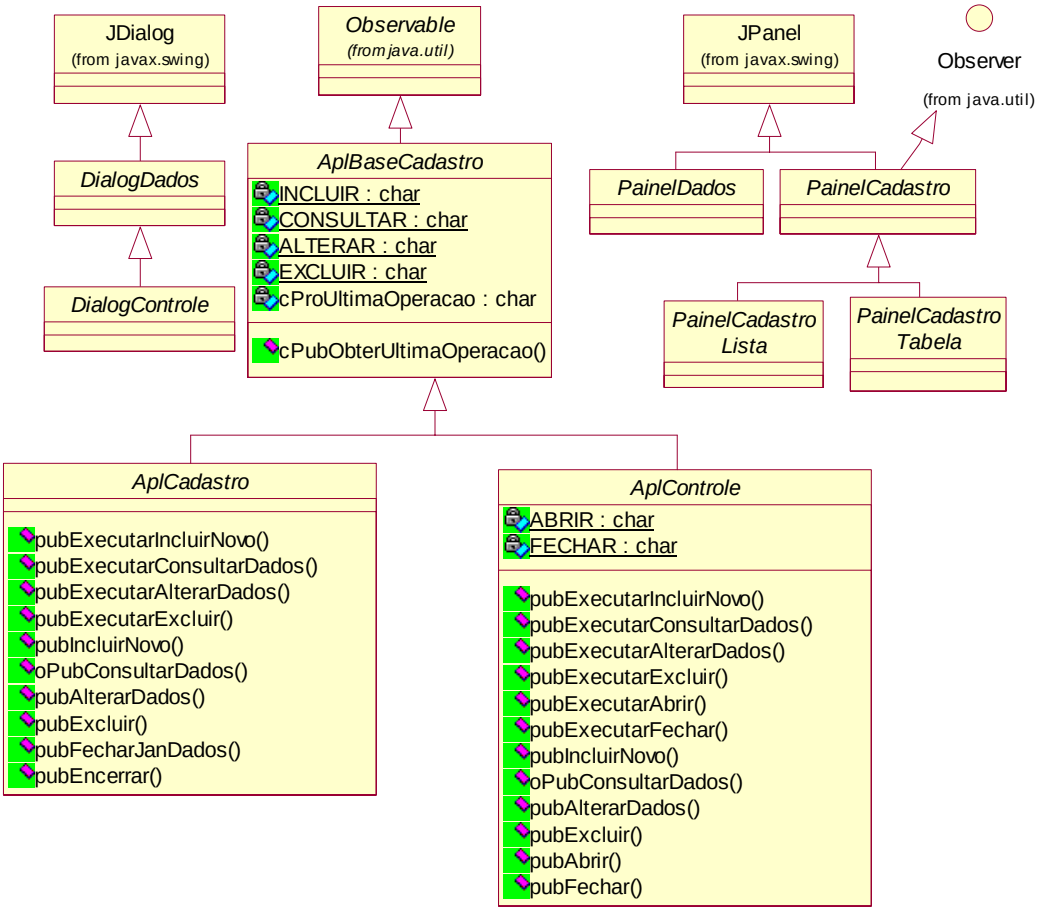

Figura 6.25. Diagrama de Classes do Utilitário Cadastro2.

Como vemos na figura, o *framework* contém um conjunto de classes abstratas a serem estendidas tanto por classes de gerência de tarefa (aplicações), quanto por classes de interação humana (componentes gráficos).

As classes de aplicação foram divididas em dois tipos: aplicações de cadastro e aplicações de controle. As primeiras reúnem as funcionalidades de inserção, consulta, alteração e remoção, enquanto as últimas possuem também funcionalidades de abertura e fechamento (por exemplo, de diagramas). Ambas as classes implementam a classe *Observable*, que pertence ao padrão de projeto Observador (GAMMA et al., 1995). A idéia é que, ao serem cadastrados os objetos, os observadores (geralmente interfaces gráficas) são notificados da "novidade" e podem se alterar de acordo com o que foi realizado. Para citar um exemplo em ODEd, *JanOntologia* é observadora de *AplEditarConceito*, sendo que se

um conceito é criado, alterado ou excluído, essa modificação é refletida automaticamente nos diagramas e na árvore da ontologia (*PainelArvoreOntologia*).

As interfaces gráficas do *framework* também são divididas em dois grupos: caixas de diálogo e painéis. As caixas de diálogo são mais simples: *DialogDados* permite que os dados de um objeto sejam mostrados e alterados, e que um novo objeto seja criado; *DialogControle* permite, além disso, que objetos sejam abertos a partir de uma lista. Já os painéis são mais flexíveis, podendo ser acoplados em qualquer tipo de *Container* Swing (*JFrame*, *JInternalFrame* e até *JDialog* também). O *PainelCadastro* lista os objetos existentes (há uma versão que usa lista e outra que usa tabela) e provê botões de acesso às funcionalidades de cadastro, enquanto *PainelDados* mostra um formulário para preenchimento ou alteração dos dados do objeto a ser cadastrado. Como já foi explicado anteriormente sobre o padrão Observador, *PainelCadastro* implementa a interface *Observer* e se atualiza automaticamente a cada cadastro de objeto.

As classes de interface gráfica são acopladas às classes de aplicação dentro do *framework*, dado que um clique em um botão na interface gráfica automaticamente chama o método correspondente na classe de aplicação. Resta ao desenvolvedor codificar somente os aspectos específicos do caso de uso que está implementando.

6.7.4. Janela MDI (*Multiple Document Interface*)

Aplicações que utilizam janelas MDI, ou seja, janelas que comportam outras janelas abertas em seu interior, são muito comuns. Tais janelas podem ser vistas hoje em editores de texto, ferramentas diagramáticas, editores de imagem e em vários outros tipos de aplicação. É interessante, portanto, construir classes de utilitário que implementem o comportamento desse tipo de janela para que outros sistemas em ODE possam utilizá-las.

A Figura 6.26 mostra as classes que compõem esse utilitário. Como podemos ver, este pacote também é baseado no *framework* gráfico *Swing*, da linguagem Java, que, por sua vez, já possui quase toda a implementação necessária para construção de janelas MDI, pois *JDesktopPane* permite que sejam inseridas janelas do tipo *JInternalFrame* em seu interior.

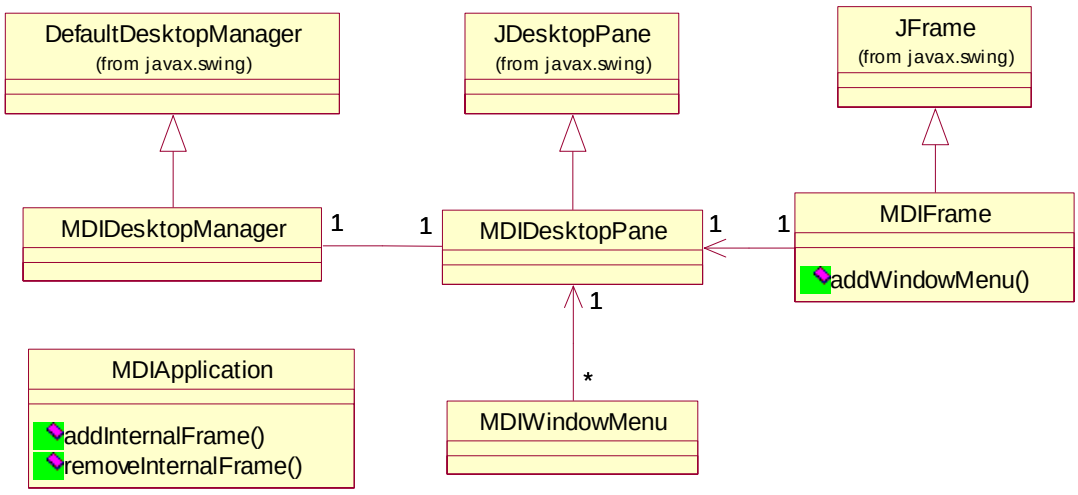

Figura 6.26. Diagrama de Classe do Utilitário MDI.

As classes do pacote MDI adicionam uma funcionalidade a mais: a criação de um menu "Janela" (implementado pela classe *MDIWindowMenu*), que automaticamente lista todas as janelas internas abertas e permite que o usuário alterne entre elas ou as disponha lado a lado ou em cascata. Para ter este menu à disposição do usuário, basta que o desenvolvedor chame o método *addWindowMenu()* durante a criação da janela. O pacote também provê a classe *MDIApplication* que é uma sugestão de interface a ser implementada por toda aplicação que tem como janela principal uma janela MDI.

6.8. Implementação e Testes

A fase de implementação toma por base o modelo de projeto para programação das classes que compõem o sistema. O sistema é concebido através desta implementação, produzindo-se os arquivos de código fonte que resultarão em um sistema executável através de compilação ou interpretação. A implementação é tão mais fácil quanto o projeto tenha sido elaborado corretamente e com detalhes suficientes. Poucos diagramas adicionais são criados na fase de implementação. Na verdade, os diagramas criados na fase de projeto são detalhados ou corrigidos quando percebemos a necessidade (FURLAN, 1998).

Como já dito anteriormente, foram usadas a plataforma de desenvolvimento Java e o banco de dados Microsoft Access para implementação de ODEd, pois esta é a plataforma de desenvolvimento padrão do projeto ODE. Apesar de ser desenvolvido como uma

ferramenta de ODE, ODEd foi implementado em separado e deverá ser integrado assim que atingir sua versão final.

Além da implementação, precisamos garantir que o software criado é de qualidade e funciona corretamente, ou seja, produz o resultado esperado pelo usuário. Para este fim, deve-se realizar a atividade de testes, que neste trabalho ocorreu em dois momentos: primeiro, durante a implementação, cada caso de uso era testado assim que ia sendo implementado. Posteriormente, com o protótipo pronto, todo o sistema foi testado, sendo que o testador se colocou no papel do usuário, anotando as dificuldades e erros encontrados.

Durante a fase de testes, principalmente durante os testes isolados dos casos de uso, muitos erros foram encontrados. Isso é encarado como um sucesso, e não um fracasso, da atividade de testes, já que os erros foram reportados e corrigidos a tempo. Com certeza muitos erros e dificuldades de uso ainda serão encontrados, pois o que foi implementado é apenas um protótipo do que deve ser ODEd.

6.9. Protótipo Implementado

Esta seção procura ilustrar o funcionamento do protótipo implementado, exibindo suas principais funcionalidades.

Ao executarmos a aplicação principal de ODEd, *AplOded*, a janela principal da ferramenta é aberta. A Figura 6.27, além de mostrar esta janela, mostra várias outras características do protótipo. Primeiro, podemos observar que há uma janela interna para cada ontologia aberta, pois, como explicado anteriormente, a janela principal de ODEd é uma janela MDI (*Multiple Document Interface*).

Em segundo lugar, a figura mostra também a composição da janela da ontologia: barra de menu para acesso às funções na parte de cima; lista dos elementos da ontologia agrupados por tipo à esquerda; no canto inferior esquerdo, a descrição do item que está selecionado na listagem; e à direita, os diagramas abertos. Para cada diagrama aberto, temos uma aba com seu nome, permitindo ao usuário que alterne de um diagrama para outro.

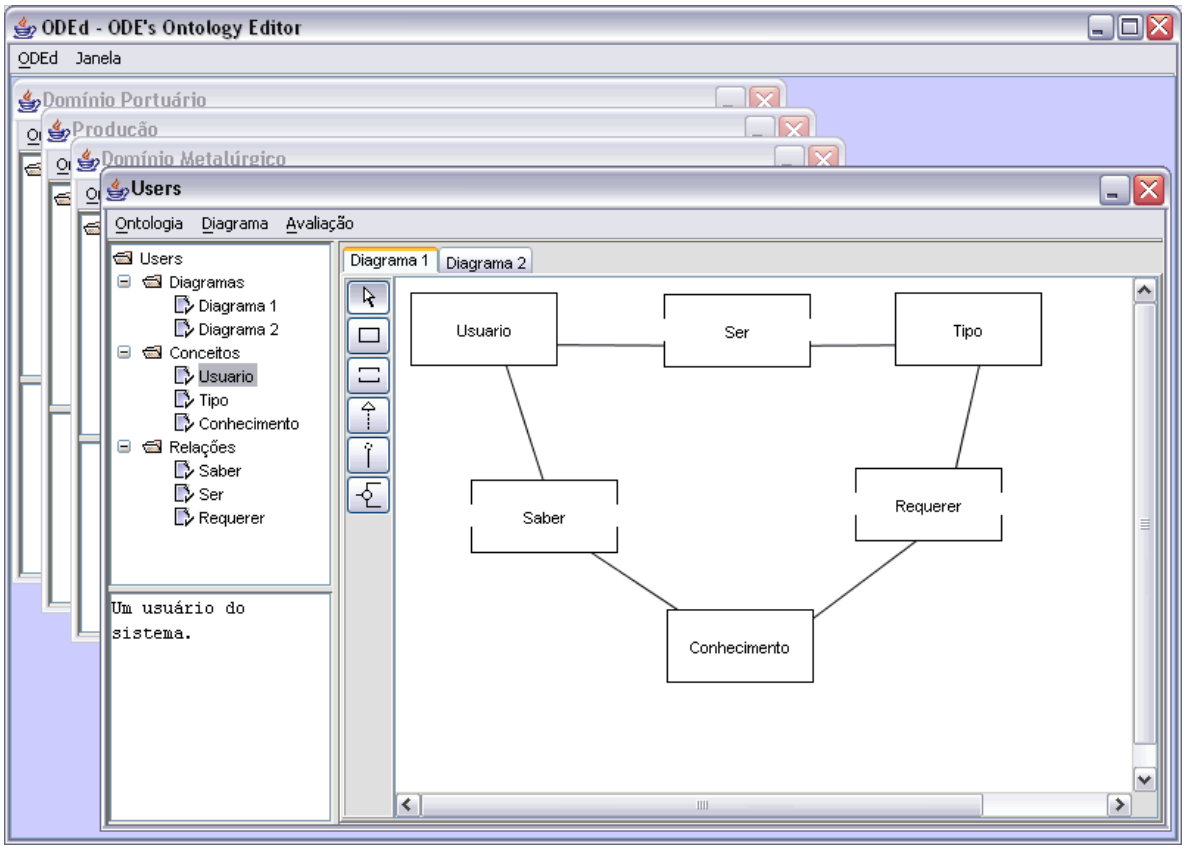

Figura 6.27. Janela principal de ODEd com várias Ontologias abertas.

Algumas funcionalidades de ODEd estão disponíveis através da listagem dos elementos (como excluir, consultar e alterar dados) e dos diagramas (criar e excluir). No entanto, a maioria das funcionalidades de ODEd são acessíveis através dos menus de funções que ficam na parte superior das janelas. A Figura 6.28 mostra os menus da janela principal de ODEd.

Vemos pela figura que estão disponíveis ao usuário somente os cenários *Criar Nova Ontologia* e *Abrir Ontologia* (do caso de uso *Controlar Ontologia*), pois todos os outros cenários e casos de uso são referentes a uma ontologia específica e, portanto, seu acesso se faz via janela da ontologia. Além do acesso a estes dois cenários, o engenheiro de ontologias pode, também, sair da aplicação e manipular as janelas internas abertas a partir dos menus da janela principal.

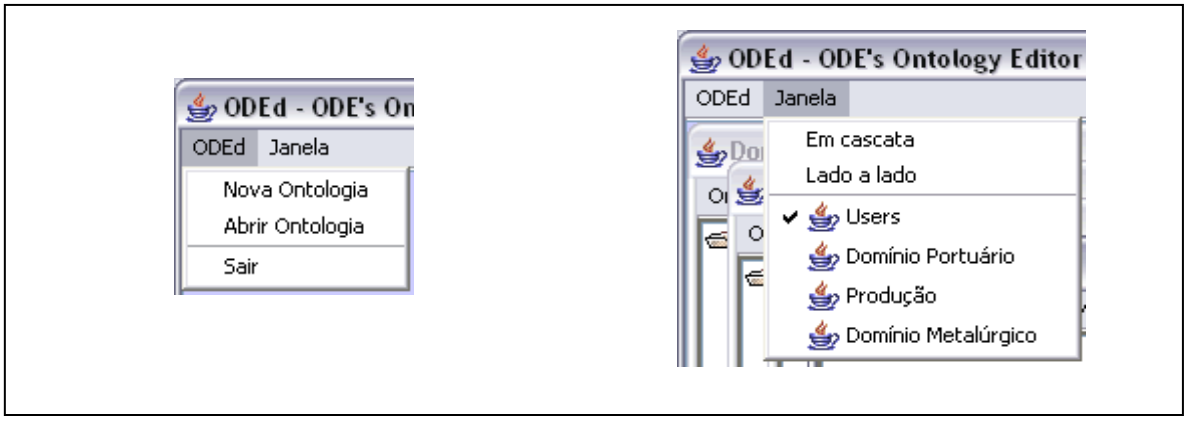

Figura 6.28. Menus de função na janela principal.

É nos menus da janela de ontologia que está a grande maioria das funcionalidades. A Figura 6.29 mostra os menus da janela da ontologia *Users*. No menu *Ontologia* o usuário tem acesso ao cadastro de questões de competência e axiomas, além da geração da infra-estrutura de objetos e da documentação sobre o domínio. Além disso, alguns cenários do caso de uso *Controlar Ontologia* estão também disponíveis, tais como excluir, salvar, fechar, e, através do item *Propriedades...*, é possível consultar e alterar os dados da ontologia.

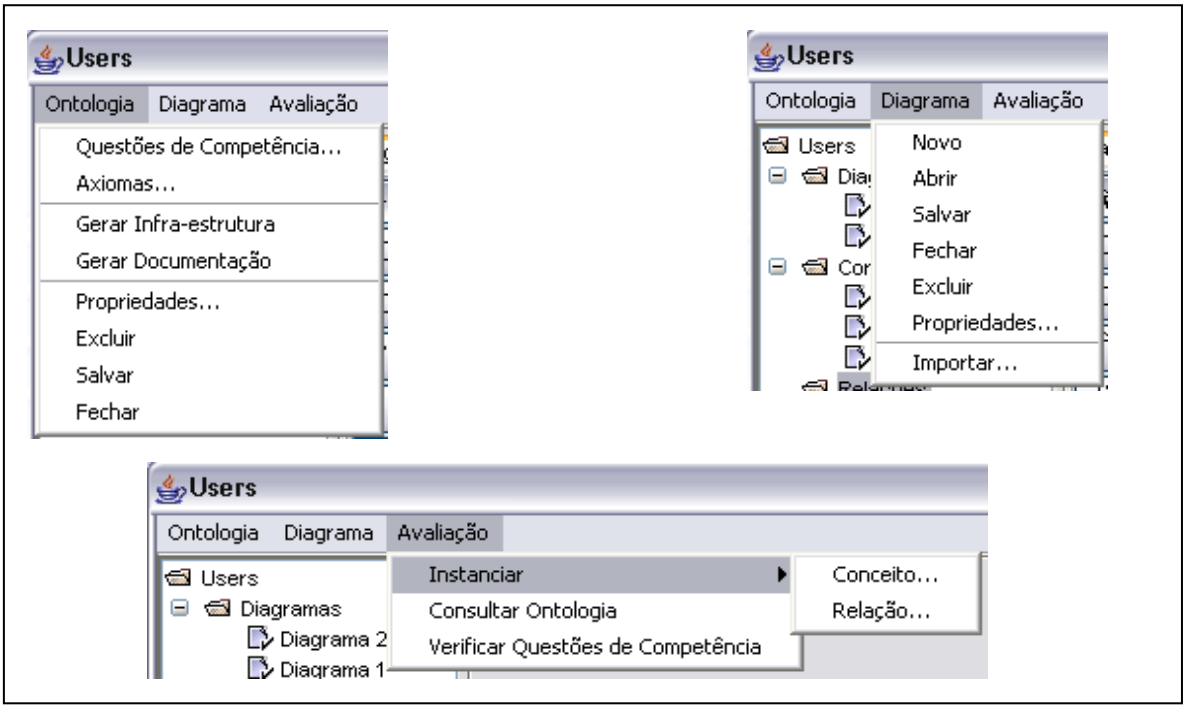

Figura 6.29. Menus de função na janela da ontologia.

O menu *Diagrama* provê as funcionalidades do caso de uso *Controlar Diagrama*,

através dos itens de menu *Novo*, *Abrir*, *Salvar*, *Fechar*, *Excluir* e *Propriedades*.... O último item, *Importar*..., dá acesso ao caso de uso *Importar Conceito*, permitindo que o engenheiro de ontologias traga para o diagrama, e conseqüentemente também para a ontologia, conceitos de outras ontologias.

Por fim, através do menu *Avaliação*, o engenheiro de ontologias pode abrir o cadastro de instâncias e enviar consultas à ontologia, seja através de uma consulta, seja verificando as questões de competência que estão formalizadas.

Como acaba de ser mostrado, ODEd possui muitas funcionalidades diferentes. Nem todas as funcionalidades terão suas interfaces humano x máquina mostradas aqui. No entanto, foram selecionadas algumas delas, que serão exibidas abaixo.

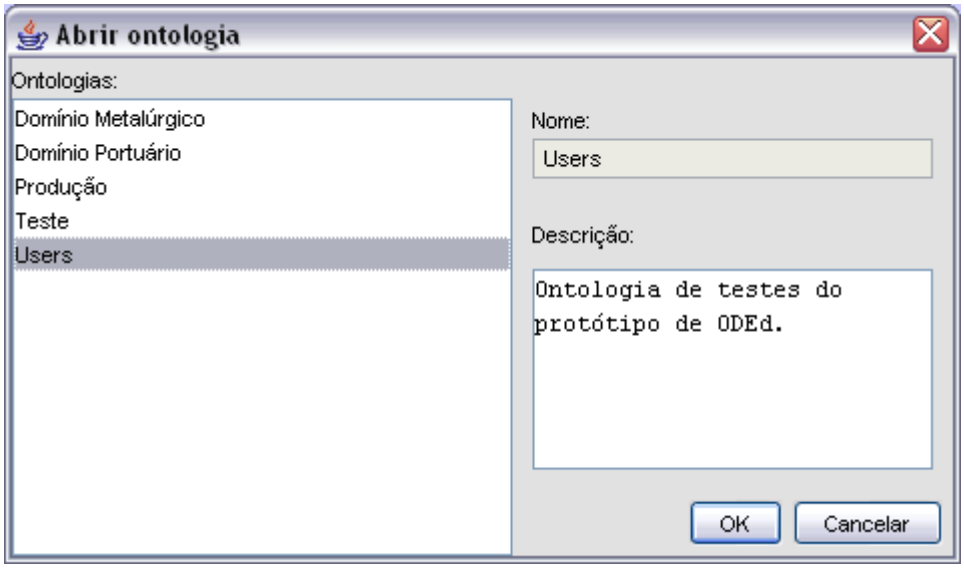

Figura 6.30. Caixa de diálogo de Ontologia.

A Figura 6.30 mostra a caixa de diálogo de ontologia, que é utilizada para os cenários *Abrir Ontologia*, *Consultar Dados de Ontologia*, *Alterar Dados de Ontologia* e *Incluir Nova Ontologia*, do caso de uso *Controlar Ontologia*. Esta mesma classe de interface automaticamente adequa-se ao cenário em questão, por exemplo, não exibindo a lista de ontologias existentes para o cenário de alteração de dados de ontologia.

Caixas de diálogo similares a esta são utilizadas para os casos de uso *Controlar Diagrama*, *Editar Conceito*, *Editar Relação* e *Editar Propriedade*.

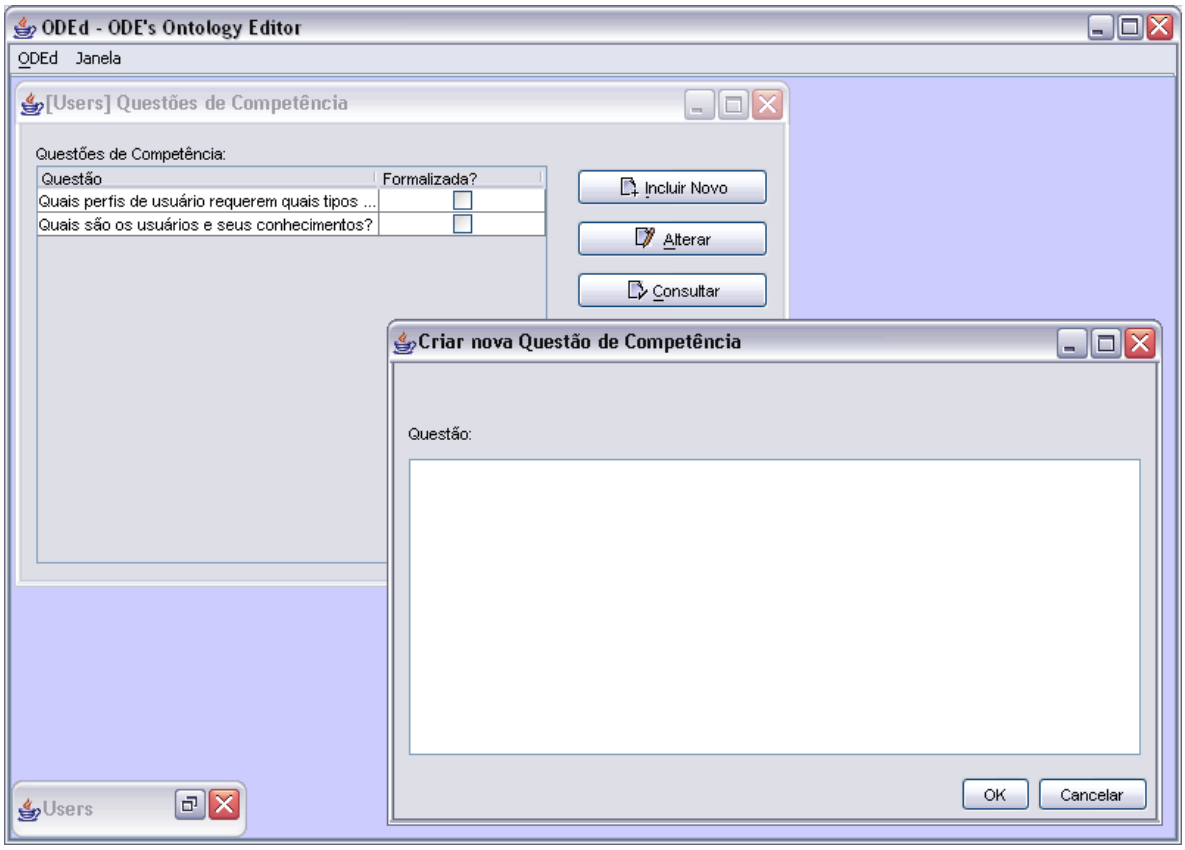

Figura 6.31. Cadastro de questões de competência.

A Figura 6.31 mostra as janelas de cadastro de questões de competência. A janela posicionada no canto superior esquerdo é uma janela de cadastro, que lista as questões de competência que já foram cadastradas e provê botões para incluir, alterar, consultar e excluir questões de competência. A outra é a janela de dados, onde o usuário informa os dados da questão de competência para os cenários de inclusão e alteração de dados. Esta mesma interface é utilizada também para os cadastros de axiomas e instâncias, com a ressalva de que a janela de dados de axioma é diferente. Esta é mostrada na Figura 6.32.

A interface de criação e edição de axiomas foi feita de forma a permitir que tanto usuários experientes quanto novatos em KIF pudessem escrever axiomas com facilidade. Na parte da esquerda são listados os itens que podem compor um axioma: conceitos, relações, instâncias e variáveis. Na parte de baixo estão os conectivos e operadores que a linguagem permite. Na parte superior temos o axioma sendo construído, sua descrição e se ele está sendo escrito de forma in-fixa ou pré-fixa.

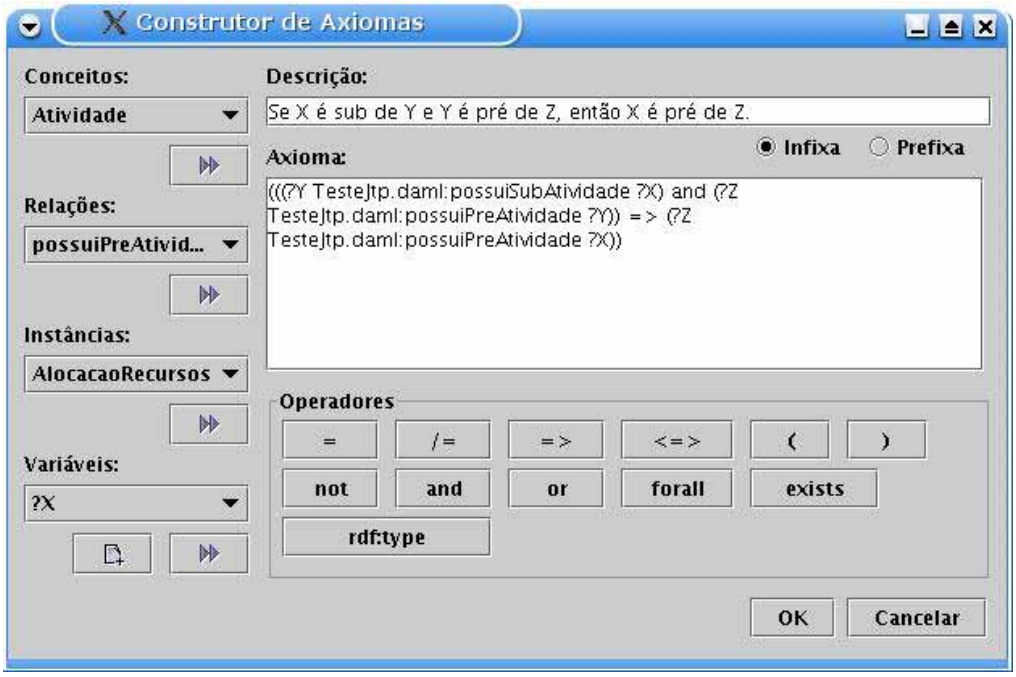

Figura 6.32. Janela de dados de axioma.

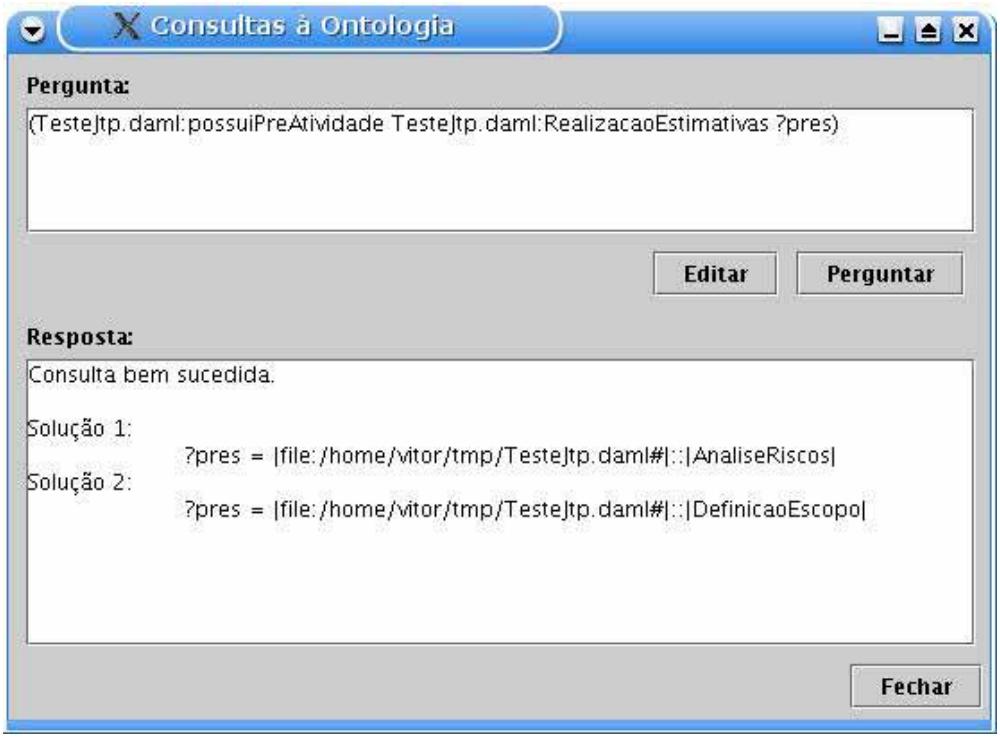

Figura 6.33. Janela de consulta à ontologia.

Para realizar consultas à ontologia o engenheiro de ontologias se utiliza da interface mostrada na Figura 6.33. Na parte de cima ele deve escrever a consulta que quer fazer, na

forma de axioma. Através do botão *Editar*, poderá usar o editor de axiomas para montar sua consulta. Após clicar no botão *Perguntar*, sua consulta é enviada para o mecanismo de inferência, cuja resposta é exibida na parte inferior da janela.

Capítulo 7

Considerações Finais

Neste capítulo são apresentadas as conclusões a respeito do projeto desenvolvido ao longo deste trabalho (seção 7.1) e as perspectivas de trabalhos futuros (seção 7.2).

7.1. Conclusões

Ontologias, quando utilizadas no contexto de Engenharia de Domínio, são bastante úteis no processo de desenvolvimento de software, pois provêem uma melhor documentação do domínio e promovem o reuso, o que leva a uma maior produtividade e qualidade no desenvolvimento.

Porém, para uma aplicação realmente eficiente dos conceitos da Engenharia de Domínio é preciso ter disponível uma ferramenta que dê suporte automatizado a esta atividade, permitindo a criação de ontologias e geração dos artefatos que poderão ser reutilizados em atividades posteriores do processo de software.

Foi objetivo deste trabalho a construção de uma ferramenta com essas características, ODEd, com base na ferramenta idealizada e parcialmente implementada em (MIAN, 2003), com algumas adequações aos novos padrões de desenvolvimento e adicionando algumas novas funcionalidades para suporte à utilização de axiomas na ontologia.

O desenvolvimento deste trabalho permitiu um maior contato com um projeto real de engenharia de software, desde o levantamento de requisitos até a implementação. Esta última promoveu um grande aprendizado de vários aspectos da linguagem Java que ainda nos eram desconhecidos. O trabalho também proporcionou o conhecimento da área de ontologias e sobre Ambientes de Desenvolvimento de Software.

Algumas dificuldades foram encontradas no caminho, como a falta de documentação, na forma de casos de uso e diagramas de classe, de todas as funcionalidades de ODEd, e a dificuldade encontrada em reutilizar o código já existente, o que forçou uma re-

implementação de toda a ferramenta. O resultado é que o protótipo apresentado não possui algumas das funcionalidades do protótipo anterior, por falta de tempo hábil para implementação.

No entanto, é importante ressaltar que a adequação do código ao novo padrão, a utilização de pacotes de utilitário e padrões de projeto, a criação de interfaces gráficas mais organizadas, a documentação do processo de desenvolvimento (dos requisitos ao projeto) e a melhoria na documentação de código são muito importantes e facilitarão bastante a tarefa de integração do editor ao ambiente ODE, além de futuras evoluções do software.

Esta foi também uma oportunidade para consolidar os conceitos vistos durante o curso de graduação de Ciência da Computação oferecido pelo Departamento de Informática da Universidade Federal do Espírito Santo, entidade à qual este trabalho foi apresentado como exigência parcial para obtenção do título de Bacharel em Ciência da Computação. A consolidação desses conceitos promoveu o crescimento na formação acadêmica e profissional do graduando.

7.2. Trabalhos Futuros

Diversas melhorias e evoluções podem, ainda, ser feitas na ferramenta:

- Melhorias no protótipo: a implementação de ODEd pode ainda sofrer diversas melhorias em relação às interfaces gráficas, facilitando ainda mais o trabalho do Engenheiro de Ontologias;
- Adequação a UML 2.0: a adequação à nova versão da UML é uma proposta de trabalho não só para ODEd, mas para outras ferramentas do ambiente ODE;
- Adequação a OWL: novas versões do mecanismo de inferência JTP já suportam OWL, portanto, ODEd poderia passar a trabalhar também com essa linguagem padrão da W3C;
- Maior integração com JTP: as respostas do mecanismo de inferência são atualmente simplesmente exibidas como texto na janela de resposta às consultas do engenheiro de ontologias. Para obter maiores benefícios da inferência, devemos converter estas respostas em objetos de domínio;

- JTP em ODE: outras ferramentas do ambiente ODE e o próprio ambiente podem se beneficiar de inferência. Uma máquina Prolog já foi integrada a ODE (RUY. 2003), e JTP poderia ser integrado de forma semelhante, compondo uma camada com diferentes mecanismos de inferência em ODE;
- Interface Web: está sendo cogitada no LabES a criação de um portal de aprendizado de Engenharia de Domínio através do uso de Ontologias. Tal portal utilizaria ODEd através de uma interface Web e permitiria uma maior colaboração entre vários usuários.

REFERÊNCIAS BIBLIOGRÁFICAS

BROEKSTRA, J.; Klein, M.; Decker, S.; Fensel, D.; van Harmelen, F.; Horrocks, I. *Enabling knowledge representation on the Web by Extending RDF Schema*. In Proceedings of the Tenth International World Wide Web Conference (WWW10), Hong Kong, May 2001.

COAD, P.; Yourdon, E.. Projeto Baseado em Objetos. Editora Campus: 1993.

CONNOLLY D.; van Harmelen, F.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; Stein, L. A. *"DAML+OIL (March 2001) Reference Description"*, December. 2001.

COSTA, P.D. *Um Framework para Persistência de Objetos em Bancos de Dados Relacionais*. Projeto de Graduação, Curso de Ciência da Computação, UFES, Outubro/2000.

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

FALBO, R.A. *A Experiência na Definição de um Processo Padrão Baseado no Processo Unificado*, Anais do II Simpósio Internacional de Melhoria de Processo de Software, SIMPROS'200. São Paulo, SP, Setembro/2000.

FALBO, R.A.; Guizzardi, G.; Duarte, K.C.; Natali, A.C.C. *"Developing Software for and with Reuse: An Ontological Approach"*, ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA-02), Foz do Iguaçú, RS, 2002.

FALBO, R.A.; Natali, A. C. C.; Mian, P. G.; Bertollo, G.; Ruy, F. B. *ODE: Ontology-based software Development Environment*, IX Congreso Argentino de Ciencias de la Computación, p. 1124-1135, La Plata, Argentina, Outubro 2003.

FIKES, R.; Jenkins, J.; Frank, G. *"JTP: A System Architecture and Component Library for Hybrid Reasoning."* Proceedings of the Seventh World Multiconference on Systemics, Cybernetics, and Informatics. Orlando, Florida, USA. July 27 - 30, 2003.

FURLAN, José Davi. Modelagem de Objetos através da UML. São Paulo: Makron Books, 1998.

GAMMA, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns - Elements of Reusable Object-oriented Software*. Professional Computing Series, Addison-Wesley, 1995.

GENESRETH, M. E.; Fikes, R. E. *Knowledge Interchange Format, Version 3.0 Reference Manual*. Technical Report Logic-921, Computer Science Department, Stanford University, 1992.

GRUBER, T.R. *A translation approach to portable ontology specifications*, Knowledge Aquisition, 5(2), 1993.

GUIZZARDI, G., FALBO, R.A., PEREIRA FILHO, J.G., *Using Objects and Patterns to Implement Domain Ontologies*, Anais do XV Simpósio Brasileiro de Engenharia de Software, Outubro/2001.

HARRISON, W.; Ossher, H.; Tarr, P. *Software Engineering Tools and Environments: A Roadmap*, In Proceedings of the 22th International Conference on Software Engineering (ICSE-00), ACM Press, 2000.

MIAN, Paula Gomes. *"Integração de Controle em um Ambiente de Desenvolvimento de Software"*, Projeto de Graduação, Curso de Ciência da Computação. Espírito Santo: UFES, Março 2001.

MIAN, P.G.; Falbo, R.A. *"Supporting Ontology Development with ODEd"*, Proceedings of the 2nd Ibero American Symposium on Sotware Engineering and Knowledge Engineering. Salvador, BA, Outubro/2002.

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

OLIVEIRA, K.M. *Modelo para Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio*, Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, RJ, Outubro/1999.

OMG – Object Management Group. "Unified Modeling Language Specification", versão 1.4, Setembro de 2001.

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

PRESSMAN, R.S. *Software Engineering: A Practitioner's Approach*. 5th edition. New York, USA, McGraw Hill, 2001.

RUY, F.B. *Infra-Estruturas de Apoio à Integração de Dados e Conhecimento em Ode*. Projeto de Graduação, Curso de Ciência da Computação, UFES, Abril/2003.

SILVA, B.C.C. *Adequação ao Meta-modelo da UML em OODE: Apoio à Elaboração de Diagramas de Classe e Casos de Uso*. Projeto de Graduação, Curso de Ciência da Computação, UFES, Outubro/2003.

SOUZA, V.E.S.; Falbo, R.A. *Construindo Axiomas e Avaliando Ontologias em ODEd*. Sessão de Ferramentas do XVII Simpósio Brasileiro de Engenharia de Software, Manaus, AM, Outubro/2003.

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

YOURDON, E. *Object-Oriented Systems Design: an Integrated Approach*. Yourdon Press Computing Series, Prentice Hall, 1994.