UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM INFORMÁTICA

FABIANO BORGES RUY

SEMÂNTICA EM UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

VITÓRIA
2006

FABIANO BORGES RUY

SEMÂNTICA EM UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

Dissertação apresentada ao Mestrado em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Orientador: Prof. Dr. Ricardo de Almeida Falbo.

VITÓRIA
Junho, 2006

FABIANO BORGES RUY

SEMÂNTICA EM UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

COMISSÃO EXAMINADORA

_____

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

_____

Prof. Davidson Cury, D.Sc.
UFES

_____

Profª. Renata Silva Souza Guizzardi, Ph.D.
SRA-IRST-ITC, Trento, Itália

Vitória, 21 de Junho de 2006.

Ruy, Fabiano Borges, 1980

Semântica em um Ambiente de Desenvolvimento de Software. [Vitória] 2006

xiii, 122 p., 29,7 cm (UFES, M. Sc., Informática, 2006)

Dissertação, Universidade Federal do Espírito Santo, PPGI

I. Engenharia de Software

A meus pais, José Maria e Joana, pelo apoio.

A Kaylla, pelo companheirismo.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

SEMÂNTICA EM UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

Fabiano Borges Ruy

Junho de 2006

Orientador: Prof. Ricardo de Almeida Falbo

O desenvolvimento de software é uma tarefa de natureza complexa. Produzir software satisfazendo as restrições de prazo, custo e qualidade tem sido um dos maiores desafios da Engenharia de Software. Ambientes de Desenvolvimento de Software (ADSs) buscam fornecer um conjunto de ferramentas, métodos e técnicas para apoiar o engenheiro de software nessa tarefa. Nos últimos anos, para fornecer apoio mais efetivo, tornou-se imprescindível a introdução de gerência de conhecimento nesses ambientes. ADSs têm incorporado conhecimento de variados tipos como de engenharia de software, domínios de aplicação e organizacional.

Durante a evolução desses ambientes, a necessidade de aplicação de semântica torna-se cada vez mais evidente, dadas características como a natureza complexa da Engenharia de Software, a premissa de integração dos ADSs e a grande quantidade de informações que armazenam. Nesse contexto, ontologias e máquinas de inferência são tecnologias-chave para possibilitar uma gerência mais adequada do conhecimento envolvido e fornecer apoio mais amplo aos usuários.

Esta dissertação discute como ontologias estão sendo utilizadas no ambiente ODE (*Ontology-based software Development Environment*) com o objetivo de evoluí-lo para um ADS Semântico. Nesse intuito, foi desenvolvida uma Ontologia de Organizações de Software para que conhecimento organizacional seja incorporado ao ambiente. O trabalho também propõe uma infra-estrutura capaz de prover serviços semânticos baseados em conhecimento ontológico e que objetiva facilitar e disseminar o uso de semântica no ambiente ODE.

Palavras-chave: Ambientes de Desenvolvimento de Software, Sistemas Semânticos, Ontologias, Gerência de Conhecimento, Conhecimento Organizacional.

ABSTRACT

SEMANTICS IN A SOFTWARE ENGINEERING ENVIRONMENT

Fabiano Borges Ruy

June, 2006

Advisor: Prof. Ricardo de Almeida Falbo

Software development is a complex task. One of the biggest challenges of Software Engineering is to produce software satisfying restrictions as time, cost, and quality. Software Engineering Environments (SEEs) try to provide a collection of tools, methods, and techniques to support the software engineer in this task. To offer a more effective support, in the last years, the introduction and management of knowledge in SEEs became indispensable. SEEs have been incorporating several types of knowledge, including software engineering, application domains, and organizational knowledge.

During SEEs evolution, the need for the application of semantics becomes more and more evident, given characteristics such as the complex nature of Software Engineering, the integration premise of SEEs, and the great amount of information they store. In this context, ontologies and inference machines are key technologies to allow more adequate knowledge management and to provide a wider support to users.

This work presents how ontologies have been used in ODE (Ontology-based software Development Environment), to make it a Semantic SEE (SSEE). For that, we developed a Software Organization Ontology to incorporate organizational knowledge to the environment. The work also proposes an infrastructure to provide semantic services based on ontological knowledge, aiming to facilitate and disseminate the use of semantics in ODE environment.

Keywords: Software Engineering Environments, Semantic Systems, Ontologies, Knowledge Management, Organizational Knowledge.

Capítulo 1

Introdução

O desenvolvimento de software é uma tarefa de natureza complexa. Produzir software satisfazendo restrições de prazo, custo e qualidade tem sido um dos grandes desafios da Engenharia de Software. À medida que os sistemas requeridos pelos usuários crescem em complexidade, o processo de software também se torna mais complexo e passa a ser fundamental a utilização de ferramentas automatizadas para apoiar suas tarefas.

Dessa forma, é crescente a demanda por Ferramentas CASE (*Computer Aided Software Engineering*) e Ambientes de Desenvolvimento de Software (ADSs), que buscam combinar técnicas, métodos e ferramentas para apoiar o Engenheiro de Software na construção de produtos de software, abrangendo todas as atividades do processo de software ou ao menos porções significativas dele (FALBO, 1998), (HARRISON *et al.*, 2000).

Outra preocupação nesta área é garantir que esses sistemas provejam informações para apoiar a resolução de problemas no desenvolvimento de software. Cada vez mais, informações de vários tipos têm sido incorporadas a esses ambientes. Um tipo fundamental é o conhecimento sobre a própria área de Engenharia de Software, importante para apoiar os desenvolvedores em diversas de suas tarefas. Outros tipos de conhecimento também relevantes incluem o conhecimento sobre domínios de aplicação, conhecimento organizacional e, naturalmente, uma gama de informações produzidas no dia-a-dia dos projetos de software desenvolvidos.

Entretanto, como resultado desse esforço, nos últimos anos, um novo problema surgiu: o excesso de recursos de informação, associado à falta de semântica para guiar uma busca por recursos realmente relevantes para o contexto em mãos. Assim como a falta de informação constitui um problema grave, o excesso também o é. Esse fato pode ser claramente percebido na *Web*, onde, muitas vezes, encontrar informação relevante pode ser uma tarefa árdua e complexa, e está relacionado, sobretudo, à falta de semântica associada aos recursos de informação (FALBO *et al.*, 2004c).

Dessa forma, os ADSs têm buscado, de diversas maneiras, organizar o conhecimento com o qual lidam. Nesse contexto, muitas tecnologias e abordagens relevantes têm sido aplicadas a esses ambientes, dentre elas ontologias, meta-dados e gerência de conhecimento. Ontologias podem ser utilizadas para explicitar uma conceituação e, a partir de uma especificação formal dessa conceituação, padronizar os recursos de informação. Meta-dados são usados para ligar os itens de conhecimento ao seu contexto original. Abordagens de gerência de conhecimento podem ser utilizadas, dentre outros, para capturar, armazenar e disseminar itens de conhecimento organizacional disponíveis no ambiente. Além disso, agentes e técnicas de inteligência artificial também têm sido aplicados, todos com o intuito de possibilitar que os ADSs forneçam um apoio mais amplo e efetivo ao processo de software.

1.1. Contexto e Objetivo do Trabalho

Este trabalho está inserido no contexto do Projeto ODE (*Ontology-based software Development Environment*) (FALBO *et al.*, 2003), desenvolvido no Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo (LabES / UFES) e que tem sido foco de pesquisas de diversos alunos de graduação e mestrado. O principal objetivo desse projeto é o desenvolvimento de um ambiente integrado, centrado em processos, para apoiar organizações de software em seus esforços de desenvolvimento e manutenção de software.

A característica marcante desse ambiente é a sua base ontológica. Ela é composta por ontologias que descrevem conhecimento, principalmente sobre engenharia de software, e é utilizada para estruturar e contextualizar os principais modelos do ambiente. Além disso, as ontologias fornecem uma conceituação fundamental para que serviços, como os de gerência de conhecimento, sejam realizados.

Todavia, para tratar mais aspectos relacionados à engenharia de software, a utilização de mais um tipo de conhecimento se faz necessária: o conhecimento sobre organizações de software. Para que o ambiente possa atuar em uma organização e prover apoio mais amplo à prática de desenvolvimento, é fundamental tratar o conhecimento sobre a organização, seus membros e competências relacionadas, elementos essenciais ao aprendizado organizacional.

Mas, não basta apenas introduzir mais conhecimento ao ambiente. Este conhecimento deve ser adequadamente formalizado e estar associado a uma base conceitual robusta, como uma ontologia. Além disso, o ambiente deve ser capaz de prover mecanismos para que o

conhecimento seja acessado e utilizado de maneira a fornecer apoio real à prática de engenharia de software.

Pautado nessas considerações, o objetivo geral deste trabalho é melhorar ADSs incorporando informações semânticas. Esse objetivo geral pode ser mais bem compreendido por meio de um conjunto de objetivos específicos. O primeiro deles é prover meios para amarrar semanticamente os itens de informação, ou mais especificamente, os objetos do ambiente. Para isso, é necessário fortalecer a arquitetura conceitual do ambiente, remodelando-a, e apresentar algumas soluções para a aplicação de semântica às tecnologias empregadas no ambiente.

O segundo objetivo específico é incorporar conhecimento organizacional ao ambiente ODE. Para tal, uma ontologia de organizações de software, com foco na estrutura organizacional e em competências, é elaborada e uma infra-estrutura de objetos é derivada a partir dela, incorporando ao ambiente funcionalidades relacionadas a esses dois aspectos.

Por fim, este trabalho também tem a finalidade de realizar estudos práticos com uma linguagem de definição e manipulação de ontologias, trabalhando mais de perto a semântica desses modelos conceituais. O meio escolhido é a proposição de uma infra-estrutura capaz de oferecer serviços semânticos e a aplicação desses serviços em tarefas do ambiente.

Acredita-se que o atendimento a esses objetivos fornecerá a ODE formas mais estruturadas e adequadas de utilizar a informação semântica envolvida no desenvolvimento de software.

1.2. Metodologia Utilizada

O começo deste trabalho, a contar pelos primeiros estudos sobre ADSs e o ambiente ODE, ocorreu em um projeto de iniciação científica realizado entre 2001 e 2002. Nesse projeto, o foco principal da pesquisa foi voltado a trabalhos sobre ADSs e integração de ferramentas, e um de seus principais frutos foi um artigo intitulado "*ODE – Um Ambiente de Desenvolvimento de Software Baseado em Ontologias*" (BERTOLLO *et al.*, 2002), que apresentava os primeiros passos de ODE como um ambiente integrado. Naquele momento, o Projeto ODE também realizava seus primeiros experimentos relacionados à semântica (FALBO *et al.*, 2002b).

A pesquisa teve continuidade com o projeto final de graduação que tratava de integração de dados e de conhecimento em ODE. A partir desse trabalho, os estudos com

ontologias e capacidades de inferência foram ampliados e geraram como resultado outros dois artigos: "*Uma Ferramenta Baseada em Conhecimento para Apoiar a Definição de Processos de Software em Níveis*" (RUY *et al.*, 2003), e "*Knowledge-based Support to Process Integration in ODE*" (RUY *et al.*, 2004). Até então, o suporte baseado em conhecimento era apoiado por uma máquina de inferência Prolog.

Com o início do mestrado, áreas relacionadas a ontologias e gerência de conhecimento, principalmente organizacional, passaram a receber mais ênfase na pesquisa e foram tema de um outro artigo: "*Learning How to Manage Risks Using Organizational Knowledge*" (FALBO *et al.*, 2004b).

Nesse momento, os estudos de meios para integrar mais efetivamente as ontologias em ODE foram aprofundados. Paralelamente a isso, ocorria a evolução do ambiente ODE, em que algumas tecnologias foram sendo incorporadas pelos membros do grupo de pesquisa, entre elas, tecnologias tradicionais de suporte à Gerência de Conhecimento e Agentes. Nesse período, foi proposta uma divisão conceitual mais elaborada do ambiente, integrando melhor as ontologias e provendo formas mais adequadas para amarrar semanticamente os objetos e classes. As novas propostas foram discutidas em: "*Ontologias e Ambientes de Desenvolvimento de Software Semânticos*" (FALBO *et al.*, 2004c) e "*Using Ontologies to Add Semantics to Software Engineering Environments*" (FALBO *et al.*, 2005b). O segundo artigo, publicado no *17th International Conference on Software Engineering and Knowledge Engineering* (SEKE'2005), em Taipei, China, contém parte fundamental deste trabalho de mestrado.

A partir da nova arquitetura conceitual, dois outros objetivos deste trabalho foram focados: conhecimento organizacional e linguagens com suporte semântico. Para lidar com eles, inicialmente foi levantado um histórico sobre ADSs e verificada a evolução de aspectos semânticos presentes nesses ambientes.

Em seguida, a pesquisa foi voltada a conhecimento organizacional e ontologias que o formalizem, como os trabalhos de (LIMA, 2004), (COTA, 2002) e (LEÃO *et al.*, 2004). Como resultado, foi desenvolvida uma ontologia de organizações de software, baseada nos trabalhos pesquisados.

O passo seguinte foi estudar a respeito de sistemas semânticos e linguagens e bibliotecas utilizadas para representar ontologias e que possuíssem capacidades de inferência. Com a popularização da *Web* Semântica e o crescimento de trabalhos com ontologias, surgiram novas linguagens para trabalhar aspectos semânticos mais adequadas que o, até então usado, Prolog, muitas delas baseadas em XML.

O conteúdo desta dissertação foi sendo elaborado enquanto o trabalho avançava. Ele reflete os estudos realizados e os resultados alcançados durante a trajetória de pesquisa.

1.3. Organização do Trabalho

Esta dissertação encontra-se organizada na presente *Introdução* e em mais quatro outros capítulos.

O Capítulo 2 – *Ambientes de Desenvolvimento de Software* – apresenta a evolução de ferramentas de apoio à Engenharia de Software, partindo das primeiras ferramentas CASE e seguindo até os ADSs mais atuais. O texto aborda as tecnologias que foram sendo empregadas nesses ambientes, com ênfase nas relacionadas à utilização de conhecimento. Além disso, é apresentado ODE, o ambiente foco de pesquisas deste trabalho.

No Capítulo 3 – *ODE: Em Direção a um ADS Semântico* – sistemas semânticos são brevemente discutidos e é feita uma associação com ADSs, enfatizando as características que fazem desses ambientes, sistemas indicados à aplicação de semântica. A arquitetura conceitual de ODE é apresentada, bem como as amarrações semânticas entre seus níveis.

O Capítulo 4 – *Tratamento Semântico ao Conhecimento Organizacional em ODE* – apresenta uma Ontologia de Organizações de Software e como ela é derivada em modelos de objetos que compõem o ambiente. Além disso, é proposta uma infra-estrutura capaz de prover serviços de natureza semântica utilizando uma linguagem de representação de ontologias e uma biblioteca com capacidades de inferência.

Finalmente, o Capítulo 5 – *Considerações Finais* – contém as conclusões sobre o trabalho desenvolvido, evidenciando suas contribuições e perspectivas de trabalhos futuros.

Capítulo 2

Ambientes de Desenvolvimento de Software

Desenvolver software de qualidade assegurada, com elevada produtividade, dentro do prazo estabelecido e sem necessitar de mais recursos do que os alocados tem sido o grande desafio da Engenharia de Software. Diante dessa necessidade, cresce a demanda por ferramentas para suportar o desenvolvimento de software de maneira eficiente e que possam dar um apoio efetivo aos engenheiros de software na realização de suas tarefas.

Ambientes de Desenvolvimento de Software são sistemas que objetivam fornecer apoio ao processo de software, provendo um conjunto de ferramentas e facilidades integradas que sejam capazes de apoiar cada uma das atividades desse complexo processo.

Claramente, construir um ambiente que atenda a esses objetivos não é algo trivial. Há anos, pesquisadores, universidades e até mesmo a indústria de software vêm investindo nesse intento, com algumas tentativas frustradas (CHEN *et al.*, 1992) e outras que fizeram com que a pesquisa progredisse, incorporando novas características e evoluindo esses ambientes (BROWN *et al.*, 1992), (THOMAS *et al.*, 1992), (AMBRIOLA *et al.*, 1997).

Nesse caminho evolutivo, que se inicia com a criação das primeiras ferramentas isoladas e segue até os complexos ambientes que existem hoje, surgiram diversas linhas de pesquisa, criando novos tipos de ferramentas e ambientes e agregando novas funcionalidades aos existentes. Houve também a incorporação de inúmeras tecnologias e disciplinas que não faziam parte da Engenharia de Software, mas de áreas como Inteligência Artificial, Bancos de Dados e até Administração. Durante a história dos ambientes, diversos exemplares foram sendo criados e são citados ao longo deste capítulo, que busca contextualizar os ambientes de desenvolvimento de software e, para isso, está organizado da seguinte forma: a seção 2.1 – *Evolução de ADSs* – percorre a principal linha evolutiva desses ambientes, apresentando seus tipos, características e exemplos; a seção 2.2 – *Áreas de Pesquisa Relacionadas* – discute algumas das áreas de pesquisa mais importantes que contribuíram para o crescimento desses ambientes; a seção 2.3 – *O Ambiente ODE* – apresenta o ambiente ODE que é o foco de

aplicação da pesquisa desenvolvida neste trabalho; por fim, a seção 2.4 apresenta as considerações finais e conclusões do capítulo.

2.1. Evolução de ADSs

Ferramentas para apoiar desenvolvedores na produção de software têm existido, de uma forma ou de outra, desde os primeiros dias da programação em computadores (HARRISON *et al.*, 2000), (GRUHN, 2002). Elas estão se tornando mais necessárias à medida que a demanda por software aumenta, o tempo de desenvolvimento reduz e a diversidade e a complexidade crescem além do imaginado há poucas décadas atrás (HARRISON *et al.*, 2000).

Para atender a essa crescente demanda por ferramentas de apoio ao desenvolvimento de software, muito esforço tem sido feito no sentido de criar ferramentas que ofereçam apoio às mais diversas atividades envolvidas no processo de software e de fazer com que elas evoluam, trabalhem em conjunto e sejam realmente efetivas no propósito para o qual foram construídas (BROWN *et al.*, 1992), (AMBRIOLA *et al.*, 1997), (HARRISON *et al.*, 2000), (HOLZ *et al.*, 2001), (FISCHER *et al.*, 2001), (GRUHN, 2002), (OLIVEIRA *et al.*, 2004), (LIMA, 2004).

A primeira geração de ferramentas de apoio ao desenvolvimento de software foi concebida por volta dos anos 1970. Essas ferramentas proviam suporte isolado a tarefas individuais como edição de código, compilação, depuração etc. (GRUHN, 2002). Desde então, houve uma evolução consideravelmente rápida das ferramentas de apoio. Nesse histórico evolutivo é importante destacar algumas classes de ferramentas, tais como ferramentas CASE, Ambientes de Desenvolvimento de Software (ADSs), ADSs Centrados em Processo, ADSs Orientados a Domínio e ADSs utilizando Gerência de Conhecimento em Engenharia de Software; assim como os fatores que motivaram essa evolução e as técnicas que foram sendo empregadas ao longo do anos.

2.1.1 – Ferramentas CASE

Todo engenheiro de software utiliza ferramentas e elas têm sido criadas desde os dias dos primeiros montadores (HARRISON *et al.*, 2000). Através do tempo, o número e a variedade de ferramentas têm crescido tremendamente. Elas abrangem desde ferramentas tradicionais, como editores, compiladores e depuradores, até ferramentas que apóiam a coleta

de requisitos, análise e projeto de sistemas, construção de interfaces gráficas, geração de consultas, arquitetura de sistemas e conexão de componentes, testes, gerência de configuração, administração de bancos de dados, reegenharia, engenharia reversa, visualização de programas e coleta de métricas (HARRISON *et al.*, 2000).

Essas ferramentas, conhecidas como ferramentas CASE (*Computer Aided Software Engineering*), tornaram-se extremamente úteis, fornecendo apoio às tarefas para as quais foram construídas.

Atualmente, as ferramentas CASE são amplamente utilizadas pela indústria de software. Ferramentas comerciais apóiam a realização de grande parte das atividades que compõem o processo de software, principalmente as relacionadas à construção.

Entretanto, cada uma dessas ferramentas geralmente atende a uma, ou a poucas atividades, resolvendo apenas problemas pontuais. Isso implica na utilização de várias ferramentas para que se possa atender às diversas atividades do processo. Além disso, são raras as vezes em que conseguem comunicar-se entre si, trocando informações ou serviços.

Segundo Gruhn (GRUHN, 2002), quanto mais essas ferramentas isoladas foram sendo construídas e utilizadas, mais urgente se tornou a necessidade de integrá-las de alguma maneira. Resultados produzidos por uma ferramenta devem ser passíveis de processamento por outra ferramenta. Assim, ao menos algum tipo de integração de dados tornou-se necessária.

Durante as décadas de 1980 e 1990, um dos conceitos mais populares entre os pesquisadores de ferramentas de apoio ao desenvolvimento de software era "integração". As ferramentas de apoio a tarefas pontuais, que já ocupavam um lugar no mercado, começaram a perder espaço entre os pesquisadores. Havia um crescente interesse na pesquisa dos chamados Ambientes CASE Integrados (CHEN *et al.*, 1992).

Acreditava-se que esses ambientes tomariam o lugar de ferramentas CASE individuais, pois seriam capazes de abrigar várias ferramentas integradas em um ambiente coeso que oferecesse suporte a todo o ciclo de vida do software, permitindo um aumento de produtividade e qualidade e redução de custos (CHEN *et al.*, 1992), (THOMAS *et al.*, 1992).

Diversas outras previsões, muitas vezes utópicas, foram feitas quanto a esses ambientes. Entretanto, os ambientes integrados ainda não foram capazes de tomar o lugar das ferramentas CASE, que são largamente utilizadas pelo mercado de software atualmente.

2.1.2 – Ambientes de Desenvolvimento de Software

Com o passar do tempo, o conceito que se consolidou quanto à integração de ferramentas foi o de Ambiente de Desenvolvimento de Software (ADS) (ou, em inglês, *Software Engineering Environment* - SEE). Embora a pesquisa tenha buscado metas mais adequadas à realidade, a finalidade dos ambientes não se alterou muito (HARRISON *et al.*, 2000). Assim, basicamente, o objetivo de um ADS é prover um ambiente capaz de apoiar todo o processo de desenvolvimento de software, integrando diversas ferramentas que possam trabalhar em conjunto.

Entretanto, a simples união das ferramentas em um mesmo sistema não é o suficiente. A integração em ADSs toma um significado muito mais amplo. Cada ferramenta passa a fazer parte de um todo. Deve haver um compartilhamento em vários níveis, englobando conceitos, funcionalidades, estruturas, representações e informações. Dessa forma, ADSs 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 as de gerência, desenvolvimento e controle da qualidade (FALBO, 1998).

A integração vem sendo considerada um dos mais desafiantes tópicos na pesquisa em ambientes de desenvolvimento de software. A integração demanda representação consistente da informação, interfaces padronizadas entre ferramentas, significados homogêneos da comunicação entre engenheiros de software e ferramentas, e uma efetiva abordagem que possibilite aos ADSs funcionar em várias plataformas (PRESSMAN, 2004).

Por ser um aspecto bastante amplo, a questão da integração em ADSs comumente é dividida em dimensões. Muitas dimensões de integração são tratadas na literatura, mas as que recebem maior destaque são (PFLEEGER, 2001), (FALBO *et al.*, 2004c):

- Integração de Dados: essa dimensão lida com a maneira como o ambiente e suas ferramentas compartilham dados. Essa foi, possivelmente, a primeira dimensão em que houve necessidade de integração. Não seria possível integrar ferramentas sem que estas pudessem trocar os dados que produzissem.
- Integração de Controle: permite que o ambiente perceba quando e quais funcionalidades são executadas e também que seja capaz de realizá-las quando necessário. Tem por objetivo suportar uma combinação flexível das funções do ambiente e de suas ferramentas, por meio do compartilhamento de suas funcionalidades ou serviços.

- Integração de Processo: estabelece uma ligação explícita entre as ferramentas e o processo de software seguido pelo ambiente. Permite que as ferramentas e os recursos definidos no processo se relacionem adequadamente. Tem a intenção de garantir que as ferramentas interajam efetivamente no apoio ao processo definido.
- Integração de Apresentação: mantém as interfaces do ambiente homogêneas e consistentes, permitindo que os usuários alternem entre várias ferramentas sem mudanças substanciais de estilo e aparência. Objetiva melhorar a efetividade da interação com o usuário, considerando desde ferramentas individuais até o ambiente como um todo.
- Integração de Conhecimento: permite que o conhecimento armazenado no ambiente esteja disponível às ferramentas ou usuários que precisem acessá-lo. Refere-se ao gerenciamento do conhecimento capturado durante projetos de software e à oferta de apoio baseado em conhecimento aos engenheiros de software.

Segundo (BROWN *et al.*, 1992), o nível de detalhes compartilhados pelas ferramentas é uma importante medida de integração. Na medida em que o nível de integração dessas dimensões aumenta, o ambiente se torna mais homogêneo, consistente, robusto e, por conseqüência, capaz de apoiar mais efetivamente o processo de software. Vale ressaltar que as dimensões de integração não são tratadas em ferramentas individuais ou porções do ambiente, mas no ADS como um todo, ao passo que este cresce e evolui.

Para que as dimensões possam ser trabalhadas em ADSs, é comum o uso de diversas tecnologias de apoio. Dessa forma, para integrar dados, pode-se utilizar, por exemplo, um sistema gerenciador de banco de dados ou arquivos XML; a integração de controle pode usar orientação a objetos, orientação a aspectos, uma estrutura Modelo-Visão-Controle ou agentes; linguagens de modelagem de processos são úteis à integração de processos; e a integração de conhecimento utiliza-se de sistemas baseados em conhecimento, técnicas de apoio à gerência de conhecimento, ontologias, agentes e máquinas de inferências. Dado que algumas dessas tecnologias são de grande importância para os ADSs, elas são discutidas em mais detalhes posteriormente.

Vale ainda comentar que ao longo da história dos ADSs, alguns ambientes deram maior ênfase a alguma das dimensões apresentadas, originando novos tipos de ambientes, como é o caso dos ADSs Centrados em Processo, que privilegiaram a integração de processos, e dos ADSs Orientados a Domínio e ADSs com Gerência de Conhecimento, que

focaram na integração de conhecimento. Esses tipos de ambientes são tratados nas seções seguintes.

2.1.3 – Ambientes de Desenvolvimento de Software Centrados em Processo

Produtos de software não devem ser desenvolvidos de forma *ad hoc*, mas sim seguindo um conjunto de atividades bem definido e ordenado. Esse conjunto de atividades, mais os recursos utilizados e produzidos, forma um processo de software, que envolve, ainda, um conjunto de ferramentas e técnicas para apoio à realização das atividades (PFLEEGER, 2001). Ou seja, para que o desenvolvimento de software ocorra com qualidade, é necessário que um processo de software bem definido seja seguido.

Com o objetivo de fazer com que os ADSs apóiem as atividades, segundo um processo de software estabelecido, surge uma linha de pesquisa com maior foco na integração de processos, a de ADSs Centrados em Processos (ADSCP) (ou, em inglês, *Process-centered Software Engineering Environment* - PSEE).

Os ADSCPs integram ferramentas para dar suporte ao desenvolvimento do produto de software e para apoiar a modelagem e a execução do processo de software que desenvolve esse produto (HARRISON *et al.*, 2000). Seu objetivo é oferecer apoio a diferentes tipos de processos de software, sendo parametrizável por um modelo de processo, o qual determina como o ambiente deve se comportar. Modelos de processo, usados dessa forma, podem definir quais atividades são executadas, quando e por quem, podem identificar ferramentas a serem usadas e o formato de documentos a serem criados e manipulados (GRUHN, 2002).

ADSCPs podem ser vistos como a automatização de um processo de software e têm a responsabilidade de (CHRISTIE, 1995): (i) guiar uma seqüência de atividades definida; (ii) gerenciar os produtos que estão sendo desenvolvidos; (iii) executar ferramentas necessárias para a realização das atividades; (iv) permitir comunicação entre as pessoas; (v) colher dados de métricas automaticamente; (vi) reduzir erros humanos; e (vii) prover controle do projeto à medida que este vai sendo executado.

Segundo AMBRIOLA *et al.* (1997), durante a execução do modelo de processo, um ADSCP deve prover uma variedade de serviços relacionados ao processo, tais como assistência aos desenvolvedores de software, automatização de tarefas de rotina, invocação e controle de ferramentas e garantia da realização de tarefas e práticas obrigatórias.

Além disso, ARBAOUI *et al.* (2002) propõem um conjunto de requisitos para esses ambientes, considerando os recentes avanços e necessidades. Segundo eles, um ADSCP deve:

- fornecer uma linguagem de modelagem de processo com sintaxe e semântica definidas, bem como apoio à execução do modelo de processo;
- apoiar a ordenação dinâmica das atividades do processo de software;
- apoiar processos de software distribuídos, o que implica em apoiar a integração interpessoal formal e a interoperabilidade entre ferramentas;
- apoiar a evolução do processo de software.

Vale comentar, ainda, a necessidade de apoio à evolução do processo para que o estado de execução do modelo esteja consistente com o estado de execução do processo, uma vez que uma distância expressiva entre esses estados significa que o modelo de processo não é mais capaz de influenciar a execução do processo. No entanto, é necessária a tolerância a desvios, uma vez que situações não previstas ocorrem. Nesse caso, o ambiente tem uma visão parcial do processo real, dependendo fortemente do *feedback* fornecido pelas pessoas que estão executando o processo para tornar-se consciente do desvio.

Por fim, FUGGETTA (2000) recomenda que os ADSCPs:

- não sejam intrusivos, isto é, suavemente se integrem e complementem o ambiente de desenvolvimento tradicional, automatizando de maneira efetiva apenas os fragmentos de processo que são razoáveis automatizar;
- sejam capazes de tolerar e gerenciar inconsistências e desvios, de forma a refletir a natureza criativa da atividade de desenvolvimento de software;
- informem claramente aos desenvolvedores o estado do processo de desenvolvimento de software de muitos pontos de vista diferentes;
- sejam implantados de forma incremental, de modo que a transição para a nova tecnologia seja facilitada e os riscos sejam reduzidos.

Alguns importantes exemplos de ADSCPs ao longo da história incluem: *Adele, Argo, PCTE e SPADE* (citados em (HARRISON *et al.*, 2000)), *OIKOS* e *EPOS* (citados em (AMBRIOLA *et al.*, 1997)) e *Merlin* e *MELMAC* (citados em (GRUHN, 2002)).

2.1.4 – Ambientes de Desenvolvimento de Software Orientados a Domínio

Um dos desafios do desenvolvimento de software é a correta compreensão daquilo que o sistema necessita realizar, ou seja, os requisitos do sistema. A falta de compreensão do problema pode levar ao desenvolvimento correto do produto errado, o que pode ser agravado

pelo fato da solução estar dispersa no conhecimento de vários especialistas. Ou seja, uma das grandes dificuldades no desenvolvimento de software é que, muitas vezes, os desenvolvedores não estão familiarizados com o domínio para o qual o software está sendo desenvolvido (OLIVEIRA, 1999).

Para tratar esse problema, vários grupos de pesquisa propuseram evoluir ADSs para apoiar o desenvolvimento de software considerando características peculiares do domínio (FISCHER, 1996), (FISCHER *et al.*, 2001), (OLIVEIRA, 1999), (OLIVEIRA *et al.*, 2004). Constatou-se que ADSs apóiam melhor o desenvolvimento e a manutenção de um produto de software se forem capazes de fornecer conhecimento do domínio aos desenvolvedores (LIMA *et al.*, 2002). A partir dessa constatação e das limitações dos ADSs convencionais em apoiar o aprendizado sobre um domínio de aplicações, definiram-se os Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSODs) (OLIVEIRA *et al.*, 2000). Esses ambientes propõem um apoio ao entendimento do domínio para os desenvolvedores, principalmente aqueles que não têm familiaridade ou experiência em realizar trabalhos no domínio considerado. ADSODs são definidos tendo como base os tradicionais ADSs, mas incorporando um novo fator: o conhecimento de um domínio específico (OLIVEIRA *et al.*, 2004).

Um ADSOD é um ambiente que apóia o desenvolvimento de sistemas de software em um domínio específico, considerando seu conhecimento para guiar o desenvolvedor em várias tarefas do processo de software. Esta nova classe de ADSs requer duas características essenciais:

1. O conhecimento do domínio deve ser capturado, modelado e armazenado para uso no ambiente;
2. O ambiente deve suportar a disseminação e uso do conhecimento do domínio.

Para atender à primeira característica, é importante que o conhecimento de domínio seja bem definido e formalizado. E isso tem sido alcançado através do uso de ontologias. Ontologias de domínio definem os conceitos, relações, propriedades e restrições válidos para o domínio em questão. Assim, o conhecimento de domínio pode ser definido sobre uma base conceitual robusta, que facilita a sua manipulação.

A segunda característica essencial pode ser contemplada por meio do uso de algumas facilidades de gerência de conhecimento. Uma vez formalizado e introduzido no ambiente, o conhecimento de domínio pode ser gerenciado, fazendo com que sejam promovidos seu uso e sua disseminação.

Podem-se citar alguns exemplos de ADSODs desenvolvidos. No contexto da estação TABA (ROCHA *et al.*, 1990), foram construídos os ambientes CORDIS (OLIVEIRA, 1999), para domínio de cardiologia; NETUNO (GALOTA, 2000), para o domínio de acústica submarina; SIDER (CARVALHO, 2002), para o domínio siderúrgico; e INSECTA (FOURO, 2002), para o domínio de entomologia. Além desses, (FISCHER *et al.*, 2001) apresentam EDC, um ADSOD aplicado ao domínio de planejamento urbano, que provê suporte a decisões, principalmente em planejamento de transportes e desenvolvimento comunitário.

2.1.5 – Ambientes de Desenvolvimento de Software com Gerência de Conhecimento em Engenharia de Software

Desenvolvedores de software lidam de forma intensa com diferentes tipos de conhecimento ao longo dos processos de software. O conhecimento do domínio da aplicação é uma parcela do conhecimento necessário. Uma outra parcela é constituída pelo conhecimento acumulado pela organização e relevante para o contexto específico. Conhecimento sobre diretrizes e melhores práticas organizacionais, técnicas e métodos de desenvolvimento de software, além de experiências anteriores com o uso dessas técnicas e métodos e com o processo de software são exemplos de conhecimento relevante nesse contexto. No entanto, identificação, organização, armazenamento e uso de conhecimento não são tarefas triviais.

Para atender a essa necessidade, cada vez mais a gerência de conhecimento tem se tornado presente em ADSs. Um primeiro passo foi utilizar algumas de suas técnicas para gerenciar o conhecimento de domínios de aplicação (ADSOD).

Entretanto, como o foco principal dos ambientes é apoiar o desenvolvimento de software, nada mais natural que gerenciar o conhecimento de seu próprio domínio, o de Engenharia de Software. Assim, alguns ambientes passaram a incorporar conhecimento a respeito de processos, atividades, recursos, artefatos, métodos, técnicas, paradigmas, tecnologias, entre outros (FALBO, 1998), (MAURER *et al.*, 2002), (LIMA, 2004).

Nesse contexto, intimamente relacionado ao conhecimento de engenharia de software, está o conhecimento sobre a própria organização, que também abrange os recursos, processos e atividades organizacionais, além de outros tipos de conhecimento específicos de uma organização, tais como sua estrutura, seus objetivos, suas normas etc (TIWANA, 2000), (DIERKES *et al.*, 2001), (LIMA, 2004).

Três pesquisas importantes que focam em ADSs com Gerência de Conhecimento em Engenharia de Software são os ADSOrg (LIMA, 2004), originários da Estação TABA (ROCHA *et al.*, 1990), o Projeto MILOS (HOLZ *et al.*, 2001) e a Gerência de Conhecimento em ODE (NATALI *et al.*, 2003). Esse último será explorado em detalhes na seção 2.3 deste capítulo.

O conceito de ADSOrg (Ambiente de Desenvolvimento de Software Orientado a Organização) (LIMA, 2004) surgiu a partir da necessidade de uma organização gerenciar o seu próprio conhecimento e o de seu domínio. Um ADSOrg apóia a atividade de Engenharia de Software em uma organização, fornecendo conhecimento acumulado pela organização e relevante para esta atividade, ao mesmo tempo em que apóia, a partir dos projetos específicos, o aprendizado organizacional em Engenharia de Software.

Os ADSOrg representam uma evolução dos ADSODs, tendo como objetivo evitar que o conhecimento de software fique disperso ao longo da estrutura organizacional e, conseqüentemente, sujeito a dificuldades de acesso e mesmo a perdas (LIMA *et al.*, 2000). Esses ambientes pretendem apoiar o desenvolvimento e a manutenção de software tanto em organizações em que essas são as atividades principais de negócio, quanto em organizações que possuem outro tipo de negócio e nas quais o desenvolvimento e a manutenção de software são atividades de suporte.

Para apoiar o gerenciamento do conhecimento em organizações, devem-se considerar alguns requisitos para ADSOrg, dentre eles (LIMA *et al.*, 2000):

i) ter uma representação da estrutura organizacional;

ii) reter conhecimento especializado sobre desenvolvimento e manutenção de software;

iii) permitir a utilização deste conhecimento em projetos;

iv) apoiar a atualização constante do conhecimento armazenado no ambiente; e

v) facilitar a localização de especialistas da organização que podem ser úteis em um projeto.

Os requisitos (i) e (v) estão fortemente relacionados, sendo que (i) refere-se ao modelo do conteúdo e (v) à utilização desse conteúdo. Uma equipe de projeto que tem acesso à estrutura organizacional na qual está inserida é capaz de localizar mais facilmente os especialistas que podem trazer contribuições ao projeto. Isso é muito importante quando o conhecimento necessário não está disponível no ADSOrg ou quando, apesar de disponível, sua aplicação requer entendimento mais profundo.

Para atender ao requisito (ii), um ADSOrg precisa dispor de conhecimento sobre as atividades de desenvolvimento e manutenção de software que são sempre realizadas na organização independente de um projeto específico, permitindo a elaboração de um processo padrão para a organização. Além disso, o ADSOrg precisa armazenar a experiência dessa organização em Engenharia de Software. Sempre que pertinente, cada item de experiência deve ser acompanhado da identificação de especialistas internos e materiais de referência que podem fornecer algum tipo de orientação adicional.

Com relação à utilização do conhecimento disponível sobre as atividades de desenvolvimento e manutenção de software – requisito (iii) –, ela é fundamental, principalmente, nas atividades iniciais de um projeto de desenvolvimento de software, quando são elaborados a proposta de desenvolvimento e o plano de projeto. Essas atividades são pouco apoiadas por ADSs e são centradas no conhecimento e na experiência do gerente do projeto. Os ADSOrg pretendem apoiar gerentes de projeto, transformando as atividades iniciais dos projetos de desenvolvimento de software em atividades centradas no conhecimento e na experiência de seus vários gerentes de projeto ao longo do tempo.

Uma questão importante é que o conhecimento de uma organização está em constante evolução, diferente do conhecimento sobre o domínio que tende a evoluir lentamente. Experiências relevantes no desenvolvimento de software podem ser adquiridas a cada projeto, podendo conduzir ao refinamento do repositório de conhecimento. Além disso, a infra-estrutura organizacional e o processo padrão da organização também podem ser alterados. Assim, é fundamental apoiar a atualização do conhecimento armazenado em um ADSOrg – requisito (iv).

Outra pesquisa em ADS com gerência de conhecimento organizacional é o Projeto MILOS (MAURER *et al.*, 2002), que é uma parceria da Universidade de Calgary, Canadá com a Universidade de Kaiserlautern na Alemanha, e tem a finalidade de oferecer uma infra-estrutura que integre os conceitos de ADSCP e de Gerência de Conhecimento. O ambiente construído é composto por um Ambiente de Modelagem Estendida de Processos, um Ambiente de Planejamento de Projeto, um Ambiente de Controle de *Workflow* e um Assistente de Informação que armazenam e manipulam modelos genéricos de processos, planos de projeto e dados de projetos.

2.2. Áreas de Pesquisa Relacionadas

Diversas áreas de pesquisa têm sido fonte de contribuições para a evolução dos ADSs ao longo de sua história, visando a torná-los mais úteis de alguma maneira. O uso de várias tecnologias, métodos e técnicas é útil como forma de incrementar as funcionalidades dos ambientes ou trazer novas características que, por diversas vezes, apóiam sua evolução. Assim, é possível notar, em diversos ADSs, a utilização de resultados de pesquisas tanto da área de engenharia de software como de áreas afins, como inteligência artificial, banco de dados e administração, entre outras.

Esta seção discute algumas das áreas de pesquisa relacionadas a ADS, focando, principalmente, naquelas associadas à manipulação de conhecimento. Quando se trata de manipulação de conhecimento, Gerência de Conhecimento é um conceito chave e merece destaque. Mas, aliadas à Gerência de Conhecimento, muitas outras áreas são úteis ao uso de conhecimento e apoio a atividades em ADSs, tais como Ontologias, Agentes e Máquinas de Inferência.

2.2.1 – Gerência de Conhecimento

Obter sucesso em um mercado cada vez mais competitivo depende criticamente da qualidade do conhecimento que uma organização utiliza em seus processos. Em resposta a esta necessidade, a gerência de conhecimento tem sido utilizada. Segundo Benjamins (BENJAMINS *et al.*, 1998), a gerência de conhecimento não é um produto nem uma solução que organizações possam comprar prontos. É um processo que precisa ser implementado durante um período de tempo, que envolve tanto relações humanas quanto práticas de negócio e tecnologia de informação. Desta forma, a gerência de conhecimento combina ferramentas e tecnologias para prover apoio à administração de conhecimento, gerando benefícios para a organização e para seus membros.

Uma gerência de conhecimento eficiente deve ser capaz de apoiar a criação, captura e utilização dos vários tipos de conhecimento com os quais lida. As atividades básicas de gerência de conhecimento incluem: identificação, captura, adaptação, integração, disseminação, uso e manutenção do conhecimento. No núcleo dessas atividades, encontra-se uma memória corporativa ou organizacional, apoiando o reúso e o compartilhamento do conhecimento organizacional (MARKKULA, 1999).

Os objetivos de uma organização determinam o tipo de conhecimento que ela deve capturar. Inúmeros tipos de conhecimento podem ser utilizados para evitar retrabalho e melhorar a qualidade. Processos, modelos de qualidade, artefatos desenvolvidos, experiências e lições aprendidas são alguns exemplos de tipos de conhecimento reutilizáveis. Porém, para que o reúso de conhecimento seja eficiente, é necessário um armazenamento adequado desse conhecimento. Por exemplo, os itens de conhecimento gerados em um projeto devem ser adaptados a futuras necessidades de outros projetos, agregando informações que auxiliem seu reúso (BROMMÉ *et al.*, 1999). Esses objetivos podem ser alcançados através de um Sistema de Gerência de Conhecimento efetivo.

Gerência de Conhecimento em Organizações de Software

Embora muitas organizações não percebam, conhecimento sempre foi administrado por elas em algum nível e de alguma maneira (MEEHAN *et al.*, 2002). No entanto, a gerência explícita e efetiva do conhecimento tem sido apresentada como um fator chave para o sucesso das organizações nos ambientes de negócio atuais, não sendo diferente quando o negócio é o desenvolvimento de software. Segundo (KUCZA *et al.*, 2001), muitas organizações de desenvolvimento de software têm reconhecido que, para terem sucesso no futuro, precisam administrar e utilizar conhecimento de forma mais efetiva, produtiva e inovadora, envolvendo desde indivíduos e equipes de projeto, até a organização como um todo. Assim, as metas gerais da Gerência de Conhecimento podem ser formuladas como: tornar os indivíduos de uma organização conscientes dos processos de conhecimento, melhorar esses processos onde problemas tenham sido identificados e apoiá-los com os meios técnicos sempre que possível e razoável (LIMA, 2004).

PRAHALAD *et al.* (1990) argumentam que a Gerência de Conhecimento:

(i) apóia a tomada de decisão eficaz e a criação de soluções inovadoras e criativas, a redução da perda de conhecimento por saída de especialistas e o combate à repetição de erros através da exploração de experiências adquiridas em projetos anteriores,

(ii) permite que não especialistas obtenham conselho especializado quando necessário,

(iii) reduz esforço duplicado por eficientemente possibilitar a construção sobre trabalho prévio, além de apoiar a consolidação de conhecimentos em

> competências e a diminuição da curva de aprendizado de novas tecnologias, tornando a organização capaz de se adaptar rapidamente a novas oportunidades.

JURISTO *et al.* (2002) relembram, ainda, que, na verdade, o desenvolvimento de software ainda tem que atingir o nível de engenharia e que, para isso, é necessário identificar relações de causa e efeito que permitam explorar a eficácia das tecnologias de forma quantitativa e repetitiva. Segundo PFLEEGER (1999), se observarmos por tempo suficiente e com atenção suficiente, vamos achar regras racionais que nos mostrem as melhores formas de construir melhor um software. Nesse sentido, a Gerência de Conhecimento, que busca identificar o que funciona (e o que não funciona) no contexto da organização, é capaz de fornecer valioso material para a derivação de conclusões no nível de indústria e do domínio de conhecimento.

Aprendizado Organizacional

Para sobreviver, uma organização deve, ainda, estar comprometida com aprendizado. O sucesso depende fortemente de sua flexibilidade e dinamicidade, e isso somente pode ser efetivamente alcançado através do aprendizado. Contudo, aprendizado não pode ser pensado como um processo individual apenas. As organizações devem aprender com suas próprias experiências, lições aprendidas devem ser registradas e compartilhadas e o conhecimento relevante deve ser institucionalizado e reusado, prevenindo a repetição de erros (GARVIN, 2000).

Isso também vale para organizações de software, isto é, organizações de software devem se conduzir como organizações que aprendem continuamente. O desenvolvimento de software é um esforço coletivo, complexo e criativo e, para desenvolver produtos de software de qualidade, as organizações de software devem utilizar seu conhecimento organizacional de engenharia de software. Diversas atividades do processo de software podem ser melhoradas por meio da oferta de facilidades de gerência de conhecimento para apoiar sua realização. Algumas dessas atividades são: alocação de recursos, planejamento da qualidade, especificação de requisitos, definição de processos e gerência de riscos (FALBO *et al.*, 2004b).

2.2.2 – Ontologias

Ontologia é um termo usado para se referenciar uma compreensão compartilhada de algum domínio de interesse, que pode ser usada como uma estrutura unificada para solucionar problemas nesse domínio. Uma ontologia tenta resolver problemas como falta de comunicação dentro das organizações, o que pode gerar, por exemplo, dificuldades na identificação dos requisitos de um sistema. Ontologias consistem de conceitos e relações – o vocabulário – e suas definições, propriedades e restrições descritas na forma de axiomas (FALBO, 1998). Por meio de ontologias, é possível conseguir uma uniformidade de vocabulário, de forma a evitar ambigüidades e inconsistências. Mais precisamente, não é o vocabulário que especifica uma ontologia, mas as conceituações que os termos do vocabulário pretendem capturar (CHANDRASEKARAN *et al.*, 1999).

Acesso comum à informação é essencial para melhorar a comunicação entre desenvolvedores e ferramentas em um ADS, evitando problemas de interpretação. Especialmente em ADSs, ontologias podem ser usadas para reduzir confusões terminológicas e conceituais, facilitando o entendimento compartilhado e a comunicação entre pessoas com diferentes necessidades e pontos de vista. Além disso, a padronização de conceitos provida por uma ontologia permite que a comunicação entre as ferramentas que compõem um ambiente seja aprimorada.

No contexto da gerência de conhecimento, as ontologias podem ser vistas como a "cola" que mantém ligadas as atividades de gerência de conhecimento (STAAB, 2001). As ontologias definem um vocabulário comum utilizado pelo sistema de gerência de conhecimento e facilitam a comunicação, integração, busca, armazenamento e representação do conhecimento (O'LEARY, 1998).

2.2.3 – Agentes

Segundo (WOOLDRIDGE *et al.*, 2000), um agente é um sistema de computador que está situado em algum *ambiente* e que é capaz de executar *ações autônomas* de forma *flexível* neste ambiente, a fim de satisfazer seus objetivos de projeto. Estar situado em um ambiente significa que o agente é capaz de perceber o ambiente onde está inserido e executar ações que mudam esse ambiente de alguma forma.

No contexto da gerência de conhecimento, sistemas multi-agentes podem ser utilizados para disponibilizar o conhecimento adquirido ao longo de vários projetos para os engenheiros de software (PEZZIN, 2004).

Agentes de software podem ser utilizados para ligar os membros de uma organização ao conhecimento disponível (O'LEARY, 1998). Eles podem apoiar não só a busca e filtro, mas também, a disseminação do conhecimento. Se um processo de software tiver sido definido, agentes podem agir de forma pró-ativa, buscando e oferecendo itens de conhecimento que podem ser relevantes para a tarefa que o usuário está executando (NATALI, 2003).

Agentes podem não só recuperar os produtos de atividades executadas em projetos similares, mas, também recuperar o conhecimento informal existente no sistema sob a forma de lições aprendidas. Essas lições podem indicar pontos positivos de se realizar determinada ação e oportunidades de melhoria. Desta forma, com o apoio de agentes disponibilizando conhecimento já adquirido, as tarefas executadas por um engenheiro de software têm sua complexidade reduzida.

A seção 2.3 aborda a questão da incorporação de agentes em um ambiente de desenvolvimento de software, o ambiente ODE.

2.2.4 – Máquinas de Inferência

Uma tecnologia interessante capaz de fazer com que os sistemas executem suas funcionalidades, ou forneçam algum tipo de apoio, de forma mais inteligente são as máquinas de inferência. Através de capacidades de inferência, um sistema pode utilizar deduções lógicas para chegar a conclusões sobre as tarefas realizadas e, então, fornecer apoio mais efetivo ao usuário. Em ADSs, capacidades de inferência são especialmente úteis para permitir uma manipulação mais inteligente de conhecimento (RUY *et al.*, 2004).

Através de regras definidas e de um conhecimento prévio, um sistema pode utilizar uma máquina de inferência integrada para derivar novo conhecimento, bastante útil a tarefas complexas (RASMUS, 1995), como as envolvidas no desenvolvimento de software e em ADSs.

A definição de regras para inferência é especialmente interessante se essas regras forem definidas a partir de uma base conceitual robusta, como as ontologias. Os axiomas de uma ontologia podem dar origem a regras que manipulam o conhecimento definido com base na mesma ontologia (RUY *et al.*, 2004). Dessa forma, o novo conhecimento gerado a partir de dedução lógica possui força conceitual suficiente para servir de apoio ao desenvolvedor na realização de suas tarefas ao longo do processo de software.

A utilização de máquinas de inferência torna-se ainda mais interessante, quando aliada a agentes ou à gerência de conhecimento, conforme discutido no exemplo da próxima seção.

2.3. O Ambiente ODE

Um ADS especialmente importante no contexto deste trabalho é o Ambiente ODE, uma vez que este é utilizado como foco principal de pesquisa e experimentação dos estudos realizados.

ODE (*Ontology-based software Development Environment*) (FALBO *et al.*, 2003) é um ADS Centrado em Processo, que realiza Gerência de Conhecimento em Engenharia de Software, fundamentando-se especialmente em sua base ontológica.

As pesquisas relacionadas a ODE tiveram início em 1999, mas o ambiente só passou a existir como um ADS integrado a partir de 2002. Ao longo de sua trajetória, ODE sofreu várias evoluções e, em meados de 2004, em uma parceria universidade-empresa, o ambiente foi implantado em uma organização de software, visando a apontar oportunidades de melhoria nas ferramentas do ambiente, tomando por base situações reais dessa organização.

O ambiente é desenvolvido no Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo (LabES / UFES) e utiliza em sua construção produtos de software livres, incluindo a linguagem Java e o sistema gerenciador de bancos de dados PostgreSQL, rodando no sistema operacional Linux.

ODE possui várias ferramentas, dentre elas as de apoio a: definição de processos de software (BERTOLLO *et al.*, 2006), acompanhamento de projetos (ControlPro) (DAL MORO *et al.*, 2005), gerência de recursos humanos (GerênciaRH), realização de estimativas (EstimaODE) (CARVALHO *et al.*, 2006), gerência de riscos (GeRis) (FALBO *et al.*, 2004b), documentação (XMLDoc) (NUNES *et al.*, 2004), modelagem orientada a objetos (OODE), realização de inferências (RUY *et al.*, 2004) e edição de ontologias (ODEd) (MIAN *et al.*, 2003).

Além disso, há outras pesquisas relacionadas ao ambiente ODE que merecem destaque no contexto deste trabalho. São elas:

i) sua base ontológica;

ii) sua infra-estrutura de gerência de conhecimento;

iii) sua infra-estrutura de construção de agentes; e

iv) sua infra-estrutura de inferência.

Base Ontológica

Em ODE, parte-se do pressuposto que, se as ferramentas de um ADS são construídas baseadas em ontologias, a integração delas pode ser facilitada, pois os conceitos envolvidos são bem definidos pelas ontologias (FALBO *et al.*, 2003).

Dentre as ontologias que compõem a base ontológica de ODE, têm-se as ontologias de processo de software (FALBO, 1998), (BERTOLLO, 2006), de qualidade de software (DUARTE, 2001), de artefatos de software (NUNES, 2005), de gerência de configuração de software (NUNES, 2005), de riscos de software (FALBO *et al.*, 2004b) e de requisitos (NARDI *et al.*, 2006).

A instanciação dessas ontologias dá origem a uma parte importante do conhecimento do ambiente. Esse conhecimento é usado para apoiar o usuário em diversas tarefas, tais como definição de processos, alocação de recursos, avaliação de qualidade, gerenciamento de riscos, levantamento de requisitos etc. Além disso, as ontologias são utilizadas para estruturar o ambiente e sua infra-estrutura de gerência de conhecimento, para estabelecer uma forma padrão de comunicação entre os agentes que atuam no ambiente e como base para a realização de inferências.

Infra-Estrutura de Gerência de Conhecimento

Um trabalho importante desenvolvido no contexto de ODE é a sua Infra-estrutura de Gerência de Conhecimento (NATALI, 2003). Essa infra-estrutura integra um Sistema de Gerência de Conhecimento a ODE, com o objetivo de apoiar a administração de parte do conhecimento gerado durante o processo de software.

Essa infra-estrutura foi desenvolvida de modo que haja uma memória organizacional ao centro, cercada por um conjunto de serviços de gerência de conhecimento, como mostra a Figura 2.1, que incluem:

- Criação e Captura de Conhecimento: responsável por oferecer mecanismos para obtenção e armazenamento do conhecimento;
- Recuperação e Acesso ao Conhecimento: responsável por oferecer mecanismos de busca dos itens de conhecimento armazenados na memória organizacional;

- Disseminação de Conhecimento: serviço pró-ativo realizado por agentes de software com o intuito de disponibilizar aos usuários itens de conhecimento potencialmente úteis a uma dada tarefa.
- Uso do Conhecimento: responsável por apoiar o reúso, por parte do usuário, do conhecimento existente e oferecer mecanismos de realimentação sobre a utilidade do conhecimento apresentado; e
- Manutenção do Conhecimento: responsável pelo gerenciamento dos repositórios de conhecimento, tomando por base o *feedback* dos usuários.

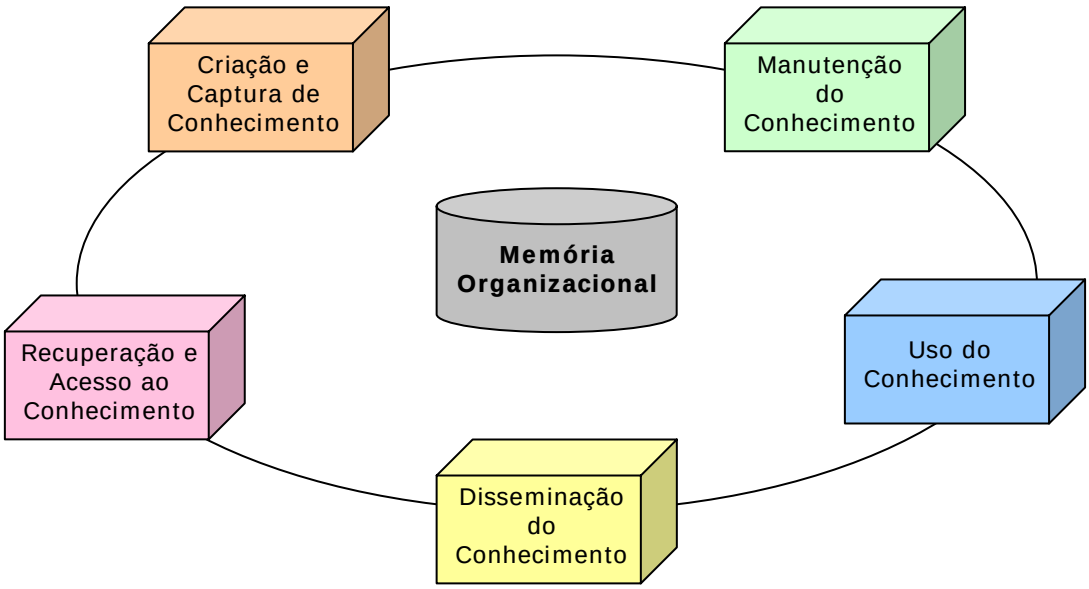


Figura 2.1 – Infra-Estrutura de Gerência de Conhecimento de ODE (NATALI, 2003)

Conforme apontado anteriormente neste capítulo, ontologias definem um vocabulário comum a ser utilizado pelo sistema de gerência de conhecimento e, por conseguinte, facilitam a comunicação, integração, busca, armazenamento e representação do conhecimento. Com base nessa premissa, a estrutura da memória organizacional de ODE é definida, também, fortemente apoiada em ontologias.

Na memória organizacional, os itens de conhecimento podem ser classificados em itens de conhecimento formais e informais. Os itens de conhecimento formais são os diversos tipos de artefatos gerados pelas ferramentas do ambiente, tais como planos de riscos, estimativas, processos e alocações de recursos. Já os itens de conhecimento informais compreendem, atualmente, lições aprendidas e pacotes de mensagens.

Infra-Estrutura de Construção de Agentes

Por serem sistemas de software complexos, ADSs são potenciais beneficiários da tecnologia de agentes. Como um exemplo da utilidade dessa tecnologia, pode-se citar a disseminação de conhecimento na gerência de conhecimento. Assim, em ODE, agentes são utilizados para aperfeiçoar algumas das funcionalidades do ambiente e uma infra-estrutura para apoiar a construção de agentes para atuarem em ODE, denominada AgeODE (PEZZIN *et al.*, 2004), foi desenvolvida.

Agentes devem ter a habilidade de se comunicar. Essa habilidade é parte percepção (o recebimento de mensagens) e parte ação (o envio de mensagens). A comunicação entre os agentes de ODE é feita usando KQML (*Knowledge Query and Manipulation Language*) (FININ *et al.* 1995). As primitivas de KQML definem as ações admissíveis que os agentes podem tentar na comunicação com outros agentes.

Novamente, ontologias são bastante úteis, agora para apoiar a comunicação entre os agentes. O projeto de um sistema multiagente requer a definição de um modelo do ambiente no qual o agente atua, para que este possa conversar sobre ele. Esse modelo pode ser exatamente uma ontologia. Assim, para que um agente consiga se comunicar com outro agente, ambos devem conhecer a(s) mesma(s) ontologia(s) (PEZZIN *et al.*, 2004).

Dessa forma, AgeODE dotou o ambiente de uma base sobre a qual novos agentes podem ser construídos seguindo o padrão criado e podendo comunicar-se com os agentes já existentes. Esses agentes são capazes de aumentar o potencial do ambiente quando atuam para apoiar a gerência de conhecimento, alocação de recursos, definição de processos, configuração automática do ambiente, identificação e avaliação de riscos etc.

Infra-estrutura de Inferência

Outra pesquisa interessante relacionada a ODE é a sua Infra-estrutura de Inferência (RUY, 2003), desenvolvida para que o ambiente possa manipular conhecimento de uma maneira alternativa, segundo o paradigma lógico.

O funcionamento da infra-estrutura consiste de três passos principais que são:

i) criação de modelos de base de conhecimento, que utiliza um editor para montar os conceitos e regras das bases de conhecimento;

ii) instanciação das bases de conhecimento por meio de um processo de extração de dados do ambiente para criar instâncias dos conceitos modelados; e

iii) realização das inferências, utilizando as bases de conhecimento em uma máquina Prolog para derivar as conclusões lógicas desejadas a respeito do conhecimento que está sendo manipulado. Por fim, os resultados são transformados em objetos para servir de apoio a diversas tarefas do ambiente.

Uma maior efetividade é alcançada se os conceitos e regras modelados são espelhados em ontologias do ambiente, uma vez que o conhecimento manipulado possui uma ligação semântica mais forte com os conceitos nos quais o ambiente se baseia.

Assim como as pesquisas mencionadas anteriormente, esta também pode ser associada às demais, oferecendo um suporte mais inteligente aos usuários de ODE.

Como é possível perceber, ODE está estruturado sobre uma robusta base ontológica. Nessa base, apóiam-se também diversas pesquisas e trabalhos que podem funcionar isoladamente, oferecendo facilidades de gerência de conhecimento, agentes, ou capacidades de inferências às ferramentas do ambiente, ou podem complementar-se, provendo um conjunto de facilidades mais amplo e efetivo.

2.4. Conclusões do Capítulo

Este capítulo discute algumas das crescentes necessidades de automatização e suporte inteligente existentes na área de engenharia de software. Contempla também algumas das soluções pesquisadas e desenvolvidas pela própria área de engenharia de software, abordando as ferramentas de apoio, seguindo pela linha de evolução dos Ambientes de Desenvolvimento de Software e instrumentos de suporte utilizados.

O desejo de criar sistemas inteligentes, capazes de realizar tarefas que não sejam meramente mecânicas, é bastante antigo. No entanto, a realização deste ato ocorre de forma lenta e gradativa. Mas, com o passar dos anos e o avanço das pesquisas, é possível aproximar-se deste intento.

Na engenharia de software, uma área em que conhecimento e experiência são características fundamentais, o foco é a automatização de tarefas complexas. Essas tarefas necessitam de conhecimento, seja da própria área de engenharia de software, de domínios de aplicação, conhecimento organizacional, ou mesmo experiência de engenheiros de software.

No contexto de ADSs, essa automatização baseada em conhecimento é caracterizada desde as primeiras tentativas de integração em que o compartilhamento de dados entre as

ferramentas tornava-se algo necessário para reduzir o esforço dos engenheiros de software. A partir de então, outros avanços podem ser considerados, como a troca de dados com algum significado embutido; a incorporação de conhecimento de diversos tipos aos ambientes; e a manipulação deste conhecimento, seja de forma direta, ou apoiada por modelos conceituais robustos. Enfim, um desejo dos ambientes atuais é estarem aptos a adquirir conhecimento, entender seu significado, manipulá-lo de forma inteligente e prover ao usuário apoio efetivo às suas tarefas.

Um trabalho que merece ser lembrado nesse campo é (BROWN *et al.*, 1992), em que, no início da década de 1990, os autores já discutiam um "nível semântico". Segundo eles, "na integração em nível semântico, as ferramentas concordam com as definições das estruturas de dados, assim como com os significados das operações sobre essas estruturas".

A parte mais interessante é que, mesmo em um momento em que os ADSs ainda davam os primeiros passos, em que a integração ocorria de forma primitiva e julgava-se ser mal entendida por muitos, o trabalho já visava um nível de integração em que as estruturas e operações tinham seu significado entendido pelo ambiente.

Desde aquela época, muito se avançou em ADSs. E, em um momento em que é possível unir em um único ambiente tecnologias como as de apoio à gerência de conhecimento, ontologias, meta-dados, técnicas avançadas de inteligência artificial e várias outras, o entendimento por parte do ambiente pode assumir proporções muito maiores, aproximando-se do que (BROWN *et al.*, 1992) apontavam como o "nível semântico".

Capítulo 3

ODE: Em Direção a um ADS Semântico

Os estudos na área de semântica relacionados ao desenvolvimento de sistemas vêm crescendo nos últimos anos. Algumas áreas de pesquisa e tecnologias mais promissoras que atualmente buscam a resolução de problemas de desenvolvimento de aplicações (EAI[1], XML, Gerência de Conhecimento, *Web Services* e, é claro, a *Web* Semântica), compartilham uma confiança fundamental em semântica (MCCOMB, 2004).

Os Ambientes de Desenvolvimento de Software (ADSs), ao longo de sua história, têm mostrado por meio de várias de suas características (grande quantidade de informações, complexidade das tarefas, premissa de integração, apoio ao trabalho em equipe, manipulação de conhecimento etc.) uma forte necessidade de tratar semântica. À medida que a complexidade dos processos de software aumenta, os ADSs têm que evoluir para oferecer um apoio mais amplo aos desenvolvedores de software (FALBO *et al.*, 2005b). Este trabalho defende que o próximo passo nessa evolução é a adoção de semântica em ADSs. Além disso, apresenta ao longo deste capítulo o que este trabalho tem realizado para materializar esse objetivo no Ambiente ODE.

Um ADS Semântico pode ser visto como um ADS em que parte da informação manipulada possui um significado formal (semântico), aumentando a habilidade das ferramentas para trabalhar em cooperação umas com as outras e com os desenvolvedores de software (FALBO *et al.*, 2005b). O ambiente ODE foi remodelado para dispor de uma arquitetura conceitual, com divisão em níveis, que apóia a realização de tarefas com base em uma conceituação robusta e que é amparada por algumas tecnologias que buscam fortalecer a aplicação de semântica no ambiente.

Este capítulo apresenta as necessidades de incorporação de semântica em ADSs e discute como ontologias estão sendo utilizadas no ambiente ODE para evoluí-lo para um ADS

[1] EAI: *Enterprise Application Integration*

Semântico. A organização em seções se dá da seguinte maneira: a seção 3.1 – *Sistemas Semânticos* – apresenta alguns conceitos, necessidades e vantagens da semântica em sistemas de software; a seção 3.2 – *Semântica em ADSs* – se volta para as necessidades de semântica em ADSs e apresenta o objetivo de evolução de ODE; a seção 3.3 – *Arquitetura Conceitual de ODE* – apresenta a estrutura em níveis do ambiente e como as ontologias são utilizadas para estruturar o ambiente e prover a conceituação necessária para a utilização de semântica; por fim, a seção 3.4 apresenta as considerações finais e conclusões do capítulo.

3.1. Sistemas Semânticos

Semântica é geralmente definida como o estudo do significado das palavras. Porém, em uma definição mais ampla, busca-se que esse significado esteja em algo mais do que apenas palavras. Ultimamente, a importância e o sucesso de um sistema estão em "o quê" os símbolos manipulados pela máquina realmente significam no mundo real. Isso não está relacionado apenas ao que eles significam, mas também ao grau em que as pessoas e outros sistemas que lidam com a mesma informação entendem e concordam com o significado atribuído (MCCOMB, 2004).

A maior parte do que se havia pensado como difíceis problemas da computação e do desenvolvimento de sistemas de negócios sofreu avanços significativos nos últimos anos. Já se sabe como escrever algoritmos mais eficientes. Já se conhecem meios bastante efetivos de processar e armazenar dados. Foram resolvidos diversos problemas de interoperabilidade de diferentes plataformas. Rotineiramente são armazenados *terabytes* de dados em *data warehouses*. Um computador doméstico possui maior capacidade de processamento do que uma organização típica de apenas uma geração atrás. Cerca de um bilhão de dispositivos estão conectados à Internet hoje (MCCOMB, 2004).

A preocupação agora, e que tende a se manter na próxima década, são alguns problemas que não podem ser resolvidos com soluções, em sua maior parte, de natureza mecânica. É necessário determinar quais os sistemas que realmente se deseja construir. É preciso saber que partes de um sistema precisam ser feitas flexíveis a mudanças futuras e quais podem permanecer estáveis por um longo tempo. É necessário compreender os sistemas que já se tem, antes de tentar mudá-los. E, principalmente, é preciso permitir que os computadores ofereçam apoio a processos nos quais, até o momento, pouca ênfase foi dada, mantendo-se como de domínio quase que exclusivamente humano, tais como interpretação,

negociação e raciocínio (MCCOMB, 2004). Em qualquer um desses itens, a semântica é um fator de grande importância. É preciso que os sistemas e as pessoas compreendam as informações com as quais estão lidando. Para a maioria das situações acima, a maior porção das dificuldades está vinculada a questões semânticas, que, uma vez resolvidas, podem reduzir significativamente os problemas.

Em nenhum período de tempo, desenvolvem-se pesquisas com foco em somente um tipo de problema, mas há períodos em que certas questões recebem maior ênfase. Segundo MCCOMB (2004), dentre as pesquisas na área de Engenharia de Software, nos anos 1980, focava-se no desenvolvimento de aplicações; no início dos anos 1990, era dada maior atenção às interfaces com o usuário e, mais tarde, às interconexões. Nesta década, iniciou-se falando de segurança e agora está-se na era dos dispositivos móveis. Porém, algumas tendências indicam que cada vez aumenta a necessidade e o foco em semântica, questão que pode perdurar por um bom tempo (MCCOMB, 2004).

A semântica está presente, mesmo que implicitamente, em várias etapas ao longo do desenvolvimento e uso dos sistemas. Os requisitos levantados com o usuário, os modelos lógicos, físicos e abstrações criados, as trocas de dados entre aplicações, as mensagens apresentadas aos usuários e os estímulos feitos pelos usuários para que o sistema execute o que as pessoas desejam, são todos exemplos em que é necessário um entendimento dos símbolos manipulados pelos sistemas ou pelas pessoas. Entretanto, o grau no qual os sistemas preocupam-se com a semântica varia bastante.

Diversas tarefas, comumente realizadas pelos humanos, têm passado a ser executadas pelos sistemas à medida que estes têm acesso a informações sobre os dados que manipulam. Assim, alguns processos passam a depender menos dos usuários, uma vez que vários passos podem ser executados pelo sistema com base em seu “conhecimento”. Exemplos disso são buscas com resultados mais rápidos e objetivos, verificação da coerência de algumas ações dos usuários, sugestões oportunas por parte dos sistemas, resultados mais exatos e validados, dentre outros.

A próxima seção apresenta como vários desses conceitos se relacionam com o tipo de sistema explorado neste trabalho, os ambientes de desenvolvimento de software.

3.2. Semântica em ADSs

Conforme discutido no Capítulo 2, ADSs têm uma história de cerca de duas décadas, iniciando com o apoio a pequenos fragmentos do processo de software, passando pela noção de ADSs Centrados em Processos (ADSCPs) e avançando para a incorporação de conhecimento sobre domínios de aplicação (ADSODs), sobre engenharia de software e a introdução de serviços de gerência de conhecimento.

Ao longo dessa história, é possível perceber diversos motivos pelos quais um maior apelo semântico é necessário para os ADSs. Nas ferramentas CASE, buscava-se integrá-las, fazendo com que trocassem, ao menos, dados. E esperava-se que os dados de uma ferramenta fossem compreendidos pelas outras com as quais interagia. Os ADSs têm como essência a integração e, por conseguinte, a necessidade de um entendimento compartilhado dos conceitos utilizados. ADSCPs, por sua vez, precisam que as ferramentas compreendam os processos de software manipulados. Por fim, os ADSs que incorporam conhecimento, seja de domínio, de engenharia de software ou organizacional, têm como fundamento compreender, mesmo que parcialmente, o conhecimento manipulado, para que possam utilizá-lo efetivamente.

Vale ainda citar as diversas áreas de pesquisa, técnicas e tecnologias envolvidas no desenvolvimento de ADSs. A contar pelos exemplos comentados no Capítulo 2 (gerência de conhecimento, ontologias, agentes, máquinas de inferência), todas estão relacionadas, de alguma forma, ao aprimoramento do que o ambiente é capaz de compreender e, então, utilizar como forma de apoio ao engenheiro de software.

Retornando à essência dos ADSs, pode-se dizer que a necessidade por semântica se deve a alguns aspectos fortemente presentes nesses ambientes e nas atividades que estes se propõem a apoiar:

- O desenvolvimento de software é um esforço criativo, complexo, coletivo e também uma tarefa de conhecimento intenso e que, muitas vezes, requer certo grau de experiência para que possa ser realizada adequadamente. A prática exige uma interação (e um entendimento comum) não só humano-máquina, mas também entre as pessoas que executam atividades do processo de software.

- ADSs, por sua própria natureza, integram ferramentas, que devem compartilhar conceitos, informações e funcionalidades com o objetivo de oferecer apoio o mais especializado possível aos usuários. A integração sempre foi apontada como um dos maiores desafios na área. E, a cada passo na evolução dos ambientes, o problema da integração aparenta ser mais complexo.

Focando nos aspectos de complexidade apresentados, dois conceitos centrais merecem maior atenção quando a questão é a incorporação de semântica aos ADSs: integração e conhecimento.

3.2.1 – Integração

Em qualquer momento da história dos ADSs, a noção de integração tem sido considerada essencial. Integração de ferramentas implica no nível de concordância entre as ferramentas e envolve algumas dimensões, comentadas no capítulo 2, tais como (THOMAS et al, 1992), (TRAVASSOS, 1994), (FALBO, 1998), (FALBO *et al.*, 2003): apresentação, que se refere à melhoria da eficiência da interação com o usuário; dados, que lida com a forma pela qual as ferramentas e o ambiente compartilham informações; controle, que objetiva suportar a combinação flexível de funções das ferramentas e do ambiente; processo, que busca garantir que as ferramentas interajam efetivamente, apoiando o processo definido; e conhecimento, que se refere ao gerenciamento do conhecimento nos ambientes.

Em qualquer uma dessas dimensões, pode-se perceber que, para que as ferramentas estejam realmente integradas, precisam compartilhar um entendimento do significado dos itens que manipulam, ou seja, as ferramentas precisam de semântica. Observando a semântica como o significado entendido por ferramentas e pessoas, pode-se ver que ela permeia todas as dimensões de integração mencionadas:

(i) A integração de apresentação está diretamente relacionada ao grau com que pessoas e sistemas concordam com o significado das interfaces com o usuário. A semântica está presente no entendimento tanto das informações apresentadas pelo sistema aos usuários, quanto nos comandos que os usuários solicitam que o sistema execute.

(ii) Os tipos de integração de dados e de controle também são extremamente dependentes de semântica, uma vez que as ferramentas devem possuir o mesmo entendimento a respeito das estruturas lógicas e de dados e dos serviços providos por outras ferramentas e pelo ambiente.

(iii) A integração de processos depende de semântica no sentido de que todas as ferramentas do ambiente precisam compartilhar um entendimento comum sobre o que é um processo de software e qual o processo de software empregado.

(iv) Finalmente, a semântica é fundamental para a integração de conhecimento, uma vez que a essência dessa dimensão é manipular o conhecimento como este é entendido pelas pessoas. Embora os sistemas ainda não possuam a capacidade de compreender completamente o conhecimento manipulado, os sistemas de gerência de conhecimento já são capazes de identificar informações relevantes que possam prover apoio útil aos usuários na apresentação desse conhecimento.

3.2.2 – Conhecimento

Atualmente, o grau com o qual um sistema preocupa-se com a semântica varia significativamente. Se um sistema possui um alto grau de precisão semântica (ou seja, a informação é semanticamente amarrada a um nível específico de discernimento) e um alto grau de veracidade semântica (ou seja, o sistema implementa procedimentos para garantir que a informação é válida), então, pode-se dizer que esse sistema é altamente preocupado com semântica (MCCOMB, 2004).

De fato, a precisão e a veracidade semântica são partes de um tópico mais abrangente que procura responder a questões como (MCCOMB, 2004):

- como dar nome às coisas;
- como formar categorias;
- como garantir algumas restrições;
- como fazer com que esses aspectos afetem os sistemas a serem construídos.

E essas questões são exatamente as que as ontologias procuram responder.

Uma ontologia é uma teoria lógica que compreende o significado pretendido de um vocabulário formal, ou seja, o seu compromisso ontológico para uma particular conceituação do mundo (GUARINO *et al.*, 1998). Uma ontologia consiste de conceitos, relações, propriedades e restrições expressas como axiomas (FALBO *et al.*, 1998).

Ontologias possuem um conjunto rico de relacionamentos, restrições e regras sobre as quais se pode raciocinar acerca da informação contida na ontologia e, por conseguinte, raciocinar sobre itens que são classificados pela ontologia (MCCOMB, 2004).

A importância das ontologias para expressar semântica é reconhecida em várias áreas, tais como *Web* Semântica e Gerência de Conhecimento (DAVIES *et al.*, 2003). Essas áreas têm em comum o problema de crescimento rápido e contínuo do volume de informações, o que torna difícil encontrar, organizar, acessar e manter as informações. Informações importantes estão normalmente espalhadas em vários repositórios e os usuários têm que

despender quantidade substancial de tempo acessando-os e os lendo, para então descobrir como esses recursos de informação estão relacionados e como eles se encaixam na estrutura geral do domínio do problema. Nesse contexto, ontologias são uma tecnologia habilitadora chave. Elas podem combinar o entendimento humano dos símbolos com a capacidade de processamento das máquinas. Além disso, o modelo de domínio descrito por uma ontologia pode ser considerado como uma estrutura unificadora, dando à informação uma representação comum e semântica (DAVIES *et al.*, 2003).

O uso de meta-dados capazes de serem manipulados pela máquina e baseados em ontologias é apontado como uma das mais promissoras formas de lidar com o problema do rápido crescimento do volume de informações. Como “dados a respeito de dados”, meta-dados podem tratar grande parte da semântica dos sistemas, armazenando o significado dos próprios dados que descrevem (MCCOMB, 2004).

3.2.3 – ADS Semântico

Partindo de uma reflexão sobre o discutido, observando as vantagens que a aplicação de semântica pode proporcionar aos sistemas, a necessidade por esse tipo de tratamento dada pelas características dos ADSs e os benefícios que a abordagem semântica tem provido às áreas relacionadas, este trabalho pretende mostrar que, para lidar com a complexidade inerente a esses ambientes, é fundamental tratar explicitamente semântica em ADSs, evoluindo-os para ADSs Semânticos (ou, em inglês, *Semantic Software Engineering Environment* - SSEE) (FALBO *et al.*, 2005b).

McComb (2004) aponta que a relevância e o sucesso de um sistema se devem a “o quê” os símbolos manipulados significam no mundo real e ao grau em que as pessoas e outros sistemas compreendem e concordam com os significados aplicados. Isso é especialmente importante em ADSs. Ao longo de todo o desenvolvimento de software, muitas informações são produzidas e requeridas por diferentes pessoas em diversos momentos. E, em muitas situações, é essencial estabelecer conexões entre recursos de informação para se obter o conjunto necessário de informações para apoiar a realização de uma atividade (FALBO *et al.*, 2004c).

Para lidar com essa questão, ontologias podem ser utilizadas para estabelecer um entendimento comum sobre o domínio de engenharia de software, domínios de aplicação e sobre as tarefas. Amarrando os recursos de informação dos ADSs com meta-dados baseados

em ontologias, é possível adicionar semântica a eles, o que abre espaço para que um ADS proveja novos serviços, de maior qualidade.

Na próxima seção discute-se como ontologias estão sendo utilizadas no ambiente ODE e o que este trabalho tem contribuído para o objetivo de evoluí-lo para um ADS Semântico.

3.3. Arquitetura Conceitual de ODE

O Ambiente ODE (*Ontology-based software Development Environment*) (FALBO *et al.*, 2004c), conforme apresentado no Capítulo 2, tem a base ontológica como um de seus fundamentos. Essa base é composta, originalmente, pela Ontologia de Processo de Software, proposta em (FALBO, 1998) e recentemente evoluída em (BERTOLLO, 2006), e por outras ontologias da área de engenharia de software, que foram integrando-se à primeira ao longo da evolução do ambiente. A Figura 3.1 apresenta a base ontológica atual de ODE e a inter-relação entre as ontologias. Vale destacar que a ontologia de organização de software foi desenvolvida neste trabalho e é apresentada no Capítulo 4.

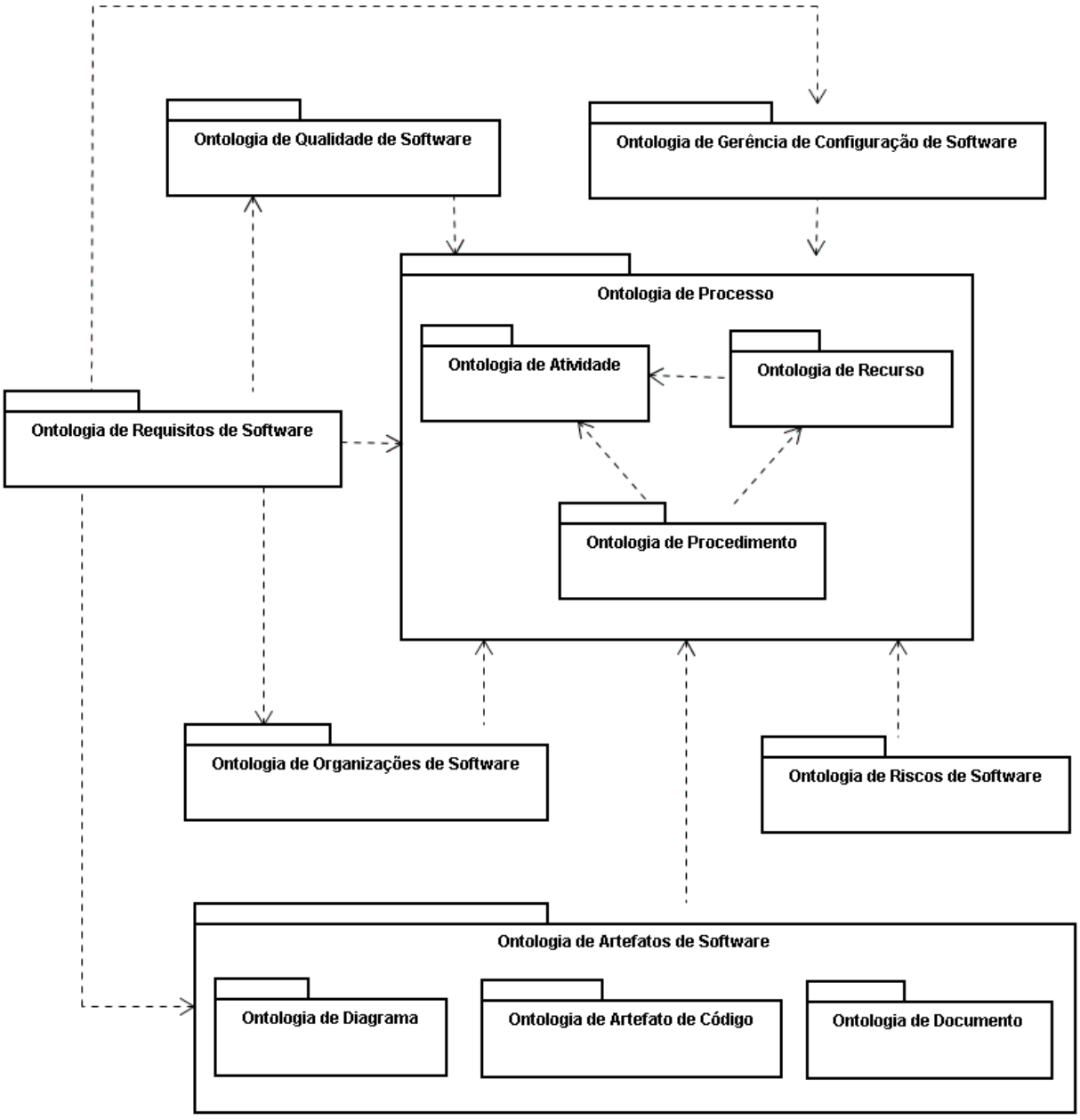


Figura 3.1 – Base Ontológica de ODE

ODE realiza grande parte de suas tarefas utilizando e respeitando os modelos e restrições impostos pelas ontologias sobre as quais se fundamenta. Assim, tem-se buscado implementar essa estreita relação entre ambiente e ontologias de uma forma natural, uma vez que há uma enorme dependência do ambiente em relação às ontologias.

É necessário, portanto, que o ambiente e cada uma das ferramentas que o compõem sejam estruturados e construídos refletindo os princípios das ontologias. Dado que ODE é desenvolvido usando a tecnologia de objetos, optou-se por realizar um mapeamento de ontologias para modelos de objetos. Esse mapeamento é feito aplicando-se parcialmente a

abordagem sistemática de derivação de infra-estruturas de objetos a partir de ontologias, descrita em (FALBO *et al.*, 2002a). Essa abordagem permite que os elementos definidos em uma ontologia (conceitos, relações, propriedades e restrições definidas como axiomas) sejam mapeados para um modelo de objetos que é, então, integrado como parte fundamental da estrutura do ambiente.

Ontologias permitem que os objetos de negócio de um ADS possam ser amarrados aos seus conceitos originais, possibilitando ao ADS a realização de ações semânticas. Como os dados manipulados por ODE estão associados a ontologias, essa ligação se estende também aos projetos realizados, processos definidos, recursos alocados, riscos gerenciados, requisitos levantados, itens de conhecimento criados ou capturados. Dessa forma, a base ontológica é capaz de apoiar inúmeras tarefas no ambiente.

3.3.1 – Estrutura em Níveis

Para apresentar a estrutura em níveis de ODE, toma-se, inicialmente, um dos primeiros trabalhos no contexto do ambiente, que mostra a concepção original dessa estrutura.

Conforme descrito em (FALBO *et al.* 2002b), o estilo arquitetural de ODE reflete sua base em ontologias. Ele possui dois níveis: o nível base, que possui classes de aplicação e modela os objetos de atividades de engenharia de software; e o meta-nível, que define as classes que descrevem o conhecimento sobre os objetos do nível base. As classes do meta-nível são derivadas diretamente de ontologias. Assim, os objetos do meta-nível podem ser vistos como instâncias de ontologias. As classes do nível base também são construídas baseadas em ontologias, porém, podem sofrer influência de características necessárias em um nível de aplicação. Por fim, diversas classes do nível base possuem uma classe de conhecimento correspondente no meta-nível, que pode ser usada para descrever os objetos do nível base.

Na abordagem original, as ontologias eram utilizadas como base para a estruturação de ODE e suas ferramentas. A estratégia consistia em primeiro desenvolver uma ontologia, a partir da qual, era derivado um *framework* de objetos e, a partir dele, a ferramenta baseada na ontologia era construída, utilizando os dois níveis citados acima (FALBO *et al.* 2002b).

Dessa forma, as ontologias não eram realmente registradas no ambiente. Elas eram apenas um modelo de referência para a derivação de classes do meta-nível e do nível base. As únicas amarrações que existiam eram uma relação conceitual entre as classes dos dois níveis,

em que objetos do meta-nível eram utilizados como uma espécie de meta-dados dos objetos do nível base.

Com o intuito de estabelecer e manter uma amarração ontológica mais forte entre os objetos de ODE, possibilitando a incorporação de novas ontologias e um tratamento mais semântico aos itens do ambiente, sua arquitetura conceitual foi remodelada para três níveis, tendo sido adicionado o nível ontológico à estrutura do ambiente, como mostra a Figura 3.2.

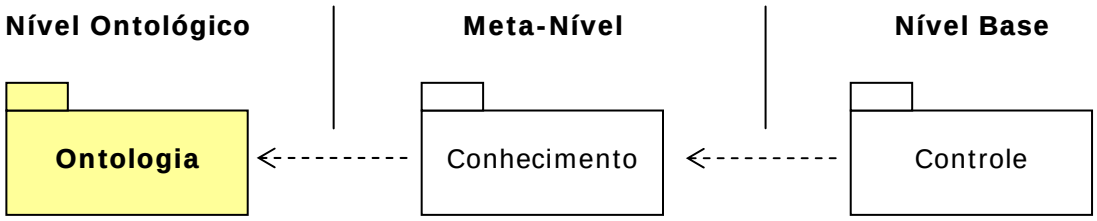


Figura 3.2 – Arquitetura Conceitual em 3 Níveis

Vale esclarecer que o termo “arquitetura conceitual” é utilizado para indicar uma decomposição em alto nível dos pacotes do ambiente, em contraste com a arquitetura de software em camadas utilizada para implementá-lo efetivamente. Cada um dos três níveis arquiteturais é descrito a seguir.

O Nível Ontológico, ou pacote `Ontologia`, é responsável pela descrição das ontologias em si. Nele encontram-se as classes que definem explicitamente uma ontologia e seus elementos e, portanto, o modelo de classes desse nível (Figura 3.3) corresponde à meta-ontologia adotada no ambiente ODE. Essa é a parte fixa da estrutura. O objetivo do Nível Ontológico é registrar as ontologias em ODE, descrevendo o que é essencial para um determinado domínio (compromissos ontológicos mínimos) e, portanto, não é de sua responsabilidade prover detalhes de implementação de sistemas nesse domínio. Contudo, vale ressaltar que as instâncias do nível ontológico guiam a definição das classes dos outros níveis, originando as principais classes tanto do meta-nível quanto do nível base. No nível ontológico encontram-se classes como `Ontologia`, `Conceito` e `Relação`, que têm como instâncias, tomando como exemplo a ontologia de processo de software, `Processo de Software`, `Atividade` e `depende de`, respectivamente (maiores detalhes são exibidos na Figura 3.4).

O Meta-nível, ou pacote `Conhecimento`, abriga as classes que descrevem o conhecimento em relação a um domínio de aplicação. Suas classes são derivadas das ontologias e as instâncias dessas classes atuam no papel de conhecimento sobre os objetos do nível base. A derivação das classes do nível de Conhecimento é feita diretamente a partir do nível ontológico, isto é, suas classes correspondem aos conceitos representados como

instâncias da classe *Conceito* (vide Figura 3.3) no pacote *Ontologia*. Elas constituem o conhecimento do ambiente, que pode ser utilizado tanto pelo ambiente quanto pelas ferramentas que o compõem. No meta-nível, encontram-se classes como `KAtividade`, `KFerramentaSoftware` e `KArtefato`, assim como suas respectivas instâncias `Planejamento de Projeto`, `Modelagem UML` e `Plano de Riscos` (maiores detalhes são exibidos na Figura 3.5).

O Nível Base, ou pacote `Controle`, define as classes responsáveis por implementar as aplicações no contexto do ambiente (funcionalidades da infra-estrutura do ambiente e suas ferramentas). Essas classes são também derivadas das ontologias, mas tipicamente incorporam detalhes não descritos por elas, necessários para implementar as aplicações do ambiente. Por não se ter uma definição de características particulares para um sistema no nível ontológico, muitas vezes, é necessária a criação de novas classes, associações, atributos e operações com o intuito de se tratar decisões específicas do nível de aplicação. No nível base, há classes como `Atividade`, `FerramentaSoftware` e `Usuario`, com suas respectivas instâncias `Planejamento do Projeto Passatempo`, `Rational Rose` e `José da Silva` (maiores detalhes são mostrados na Figura 3.6).

A nova arquitetura conceitual facilita o estabelecimento de uma correlação entre objetos dos diferentes níveis de ODE, permitindo anotar os objetos com informação semântica, agora sim, dada efetivamente pelas ontologias. A seguir, discute-se como a estrutura do ambiente é desenvolvida tomando por base essa arquitetura conceitual.

3.3.2 – Derivação das Classes do Ambiente

Conforme discutido anteriormente, o nível Ontológico descreve as ontologias em ODE e, para tal, o ambiente disponibiliza o editor de ontologias ODEd (*ODE's ontology Editor*) (MIAN *et al.*, 2003). ODEd é uma ferramenta gráfica que apóia o desenvolvimento de ontologias através da definição de seus conceitos, relações e propriedades, e que permite a definição de axiomas e avaliação de ontologias, através de um editor de axiomas integrado (SOUZA *et al.*, 2005).

O modelo de meta-ontologia mostrado na Figura 3.3 é utilizado para construir e registrar as ontologias do ambiente no nível ontológico. Esse modelo é compatível com a especificação do meta-modelo UML, ou seja, a meta-ontologia de ODEd pode ser vista como uma extensão do meta-modelo da UML. Assim, valem citar algumas relações entre a meta-ontologia e o meta-modelo da UML, tais como: `Ontologia` é um `Modelo` e os demais

elementos da meta-ontologia são elementos de modelo (`ElementoModelo`). Por exemplo, `Conceito` é um `Classificador` e `Propriedade` é um `Atributo`. Com essa abordagem, todos os modelos de ODE, incluindo ontologias, podem ser tratados uniformemente, permitindo reúso e facilitando a integração (SOUZA, 2004).

O propósito de uma ontologia e seus usos pretendidos são identificados através de questões de competência (`QuestaoCompetencia`). Uma ontologia contém conceitos (`Conceito`), que são relacionados por relações (`Relacao`), sendo que conceitos e relações podem possuir propriedades (`PropriedadeOntologia`). Por fim, uma ontologia pode ter axiomas (`Axioma`) estabelecidos (SOUZA, 2004).

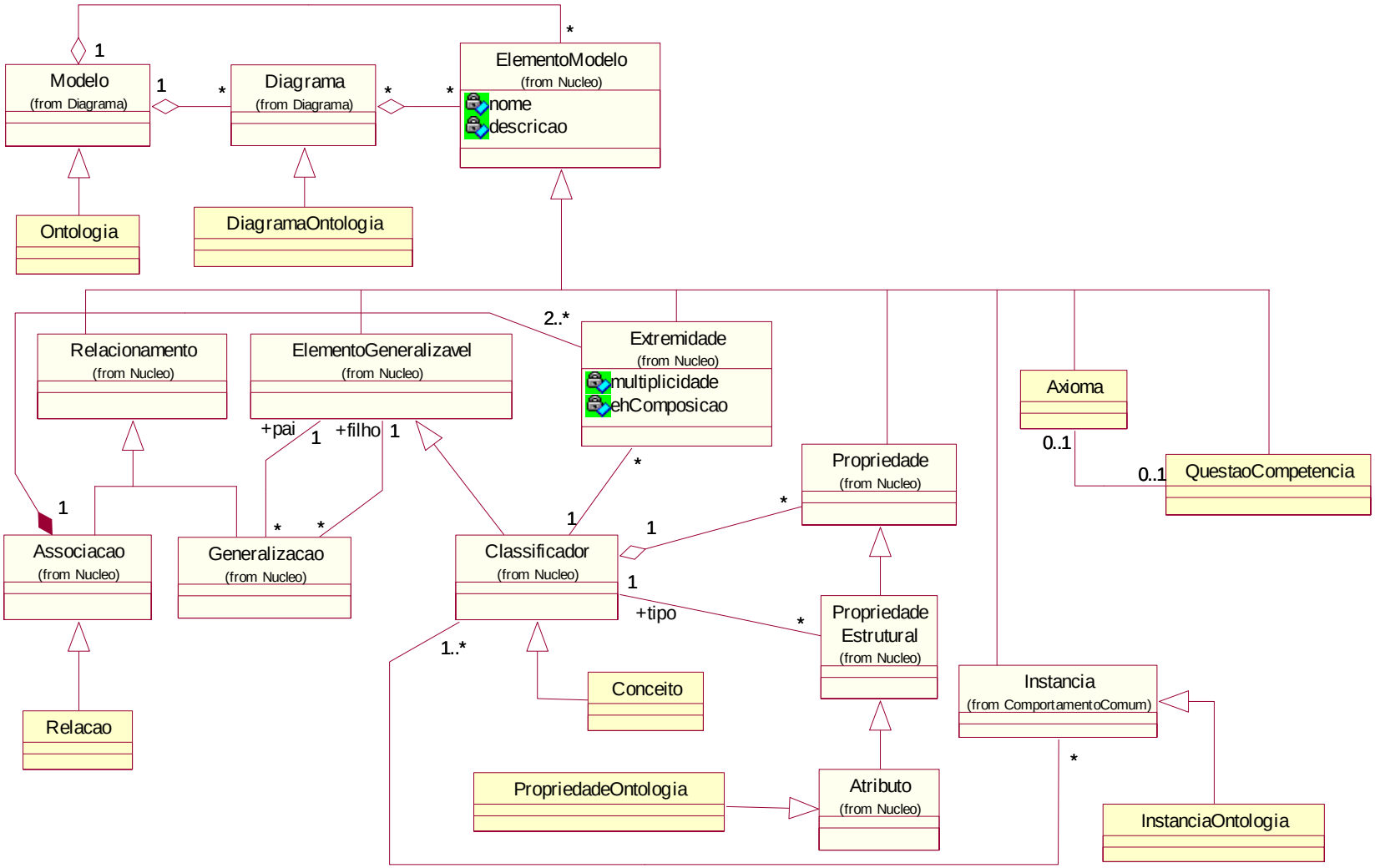


Figura 3.3 - Modelo de Classes do Nível Ontológico (SOUZA, 2004)

No nível ontológico, os objetos criados a partir da edição de uma ontologia representam os elementos pertencentes às ontologias de ODE. Estas têm sido desenvolvidas utilizando o método SABiO (***S**ystematic **A**pproach for **Bui**lding **O**ntologies*) (FALBO, 2004) que define um processo para construção de ontologias e advoga o uso de um perfil UML como linguagem gráfica para representação de ontologias. Esse perfil utiliza um subconjunto de elementos da UML exercendo o mesmo papel da notação de LINGO (FALBO, 1998), a linguagem originalmente proposta. LINGO possuía primitivas para representar conceitos, relações e propriedades, e alguns tipos de relacionamentos que possuíam uma semântica bem-estabelecida, tais como relações *todo-parte* e *sub-tipo-de*, para os quais um conjunto de axiomas formais, ditos axiomas independentes de domínio, era definido. Assim, apesar de se utilizar os elementos de modelo da UML, a semântica imposta é a mesma que a estabelecida para os correspondentes elementos em LINGO. Segundo esse perfil UML, classes com estereótipo `<<conceito>>` representam conceitos da ontologia. Relações são definidas como associações nomeadas. Propriedades de conceitos e relações são representadas como atributos de classes. Relações que contêm propriedades ou que possuem aridade maior que dois são representadas como classes associativas com estereótipo `<<relação>>`. Relacionamentos de supertipo e todo-parte são representados como generalização/especialização e agregação/composição, respectivamente. Finalmente, condicionantes entre relações são representados por restrições entre associações (MIAN, 2003).

A Figura 3.4 apresenta parte da Ontologia de Processo de Software (BERTOLLO, 2006) utilizando o perfil UML para modelagem de ontologias proposto em (MIAN, 2003).

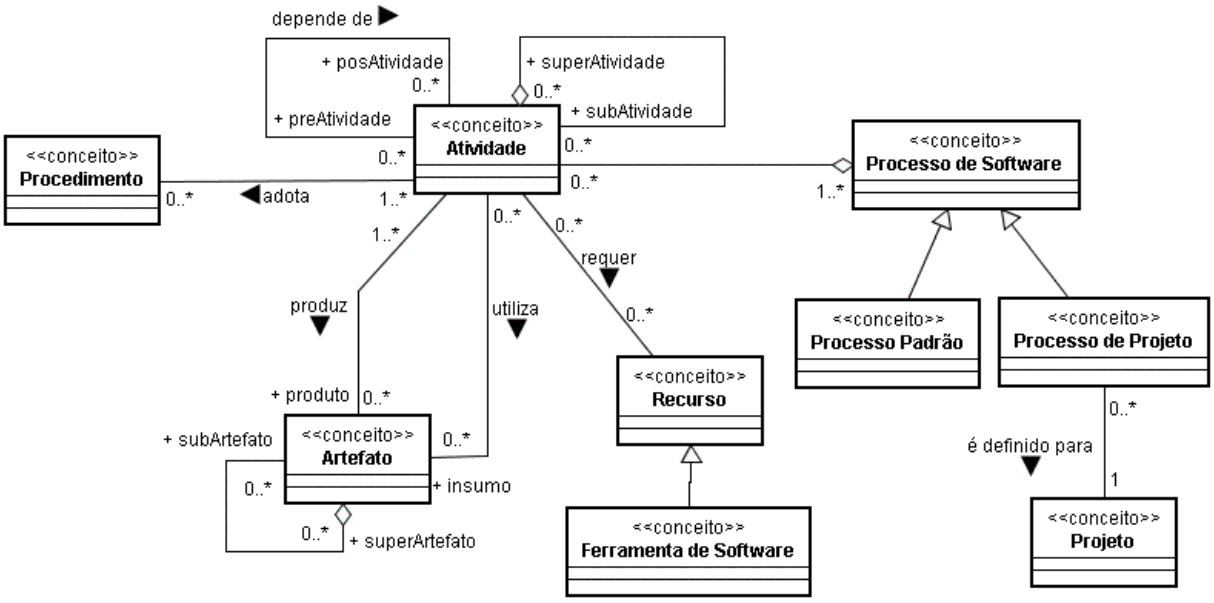


Figura 3.4 – Modelo Parcial da Ontologia de Processo de Software (BERTOLLO, 2006)

Fazendo uma analogia entre a ontologia de processo de software e como ela é representada no modelo da meta-ontologia de ODE (pacote `Ontologia`), tem-se que a própria *Ontologia de Processo de Software* é representada como uma instância da classe `Ontologia`; *Projeto*, *Processo de Software*, *Processo Padrão, Processo de Projeto, Atividade*, *Procedimento, Artefato, Recurso*, *Recurso Humano e Ferramenta de Software*, por sua vez, são instâncias da classe `Conceito`; *é definido para, decomposição de processo*, *decomposição de atividade*, *decomposição de artefato*, *depende de*, *adota*, *utiliza*, *produz* e *requer* são instâncias da classe `Relacao`; e, finalmente, existem ainda algumas questões de competências e axiomas (não exibidos graficamente) que são tratados como instâncias das classes `QuestaoCompetencia` e `Axioma`, respectivamente.

Uma vez definida uma ontologia no nível ontológico, é possível derivar os modelos de objetos que compõem os outros dois níveis da arquitetura conceitual de ODE. Nesse momento a abordagem de derivação proposta em (FALBO *et al.*, 2002a) é parcialmente utilizada, da mesma forma como originalmente proposta no Projeto ODE, discutida no início da seção 3.3.1. De forma geral, conceitos e relações são diretamente mapeados em classes e associações em um modelo de objetos. Propriedades dos conceitos e relações são mapeadas em atributos das classes originadas a partir do respectivo conceito/relação. Axiomas, por sua vez, são mapeados em métodos. A infra-estrutura básica de objetos gerada é, posteriormente, alterada pelos desenvolvedores do ambiente para incorporar as características adicionais necessárias.

Como a arquitetura conceitual de ODE possui, além do nível ontológico, outros dois níveis, o processo de derivação das ontologias não dá origem a apenas um modelo de objetos, mas a dois. Durante o processo de derivação, um determinado conceito pode derivar uma classe somente no nível de conhecimento, uma classe somente no nível de controle, classes em ambos os níveis, ou mesmo em nenhum deles. A Tabela 3.1 mostra esse mapeamento para a parte da ontologia de processo de software apresentada na Figura 3.4.

Tabela 3.1 - Conceitos e Classes Derivadas

Instâncias dos Conceitos	Classes do Meta-Nível	Classes do Nível Base
Atividade	`KAtividade`	`Atividade`
Artefato	`KArtefato`	`Artefato`
Recurso	`KRecurso`	`Recurso`
Ferramenta de Software	`KFerramentaSoftware`	`FerramentaSoftware`
Procedimento	`KProcedimento`	`-`
Projeto	`-`	`Projeto`
Processo	`KProcesso`	`-`
Processo Padrão	`-`	`ProcessoPadrao`
Processo de Projeto	`-`	`ProcessoProjeto`

Os dois modelos derivados pertencem aos pacotes `Conhecimento` e `Controle` e, apesar de distintos, estão integrados por meio de associações entre as classes derivadas do mesmo conceito e entre as classes cujos conceitos são relacionados.

As Figuras 3.5 e 3.6 apresentam, respectivamente, uma simplificação dos modelos de objetos derivados da parte da Ontologia de Processo de Software apresentada na Figura 3.4 para os níveis de Conhecimento e de Controle. É importante apontar que, em ODE, as classes de conhecimento (meta-nível) são nomeadas com o prefixo K (de *Knowledge*) e são subclasses da classe `Conhecimento`. Além disso, como a navegabilidade entre os modelos acontece do nível de Controle para o nível de Conhecimento, as associações entre as classes dos diferentes pacotes são representadas no modelo de controle (Figura 3.6) por meio de atributos das classes. Para uma discussão completa acerca desses modelos e do processo de derivação dos mesmos a partir da ontologia de processo de software, vide (BERTOLLO, 2006).

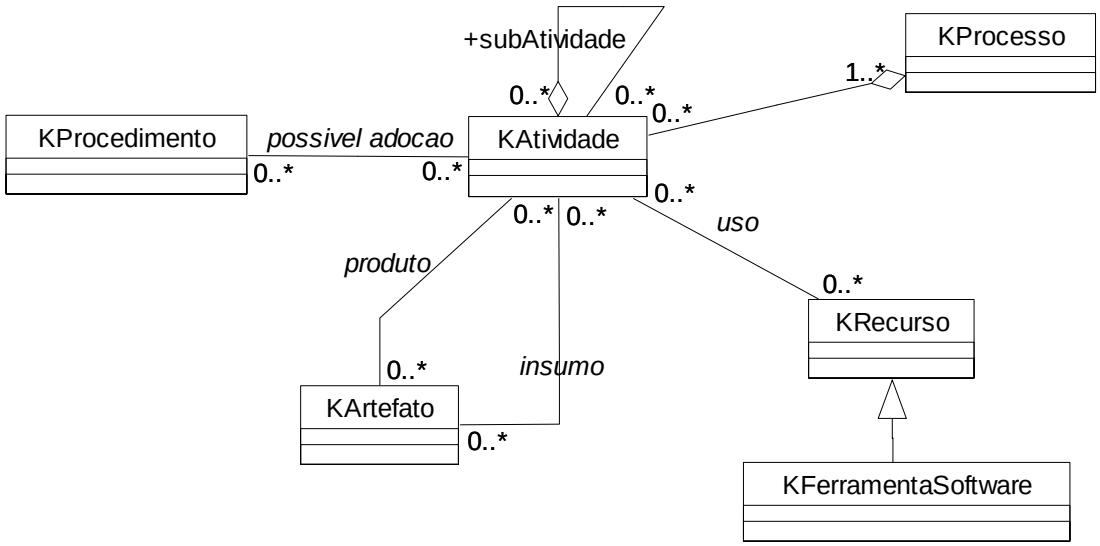


Figura 3.5 – Modelo do Meta-Nível (pacote `Conhecimento`)

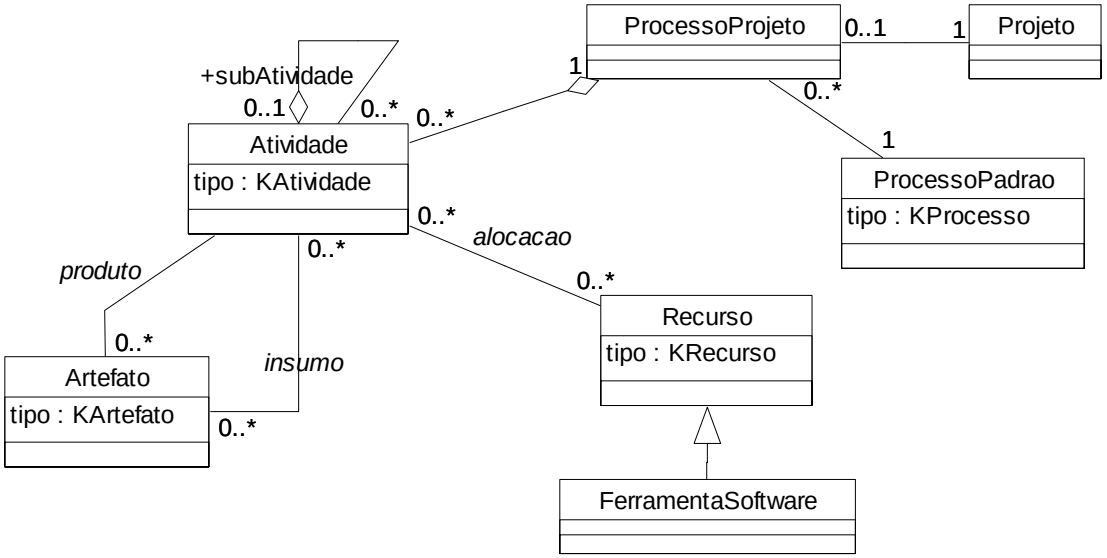


Figura 3.6 – Modelo do Nível Base (pacote `Controle`)

Uma vez que, durante o processo de derivação, cada conceito pode derivar classes em diferentes níveis, segue uma discussão sobre a lógica por trás desse mapeamento, abordando cada uma das quatro possibilidades (FALBO *et al.*, 2004c):

i) Nenhuma classe é criada.

Apesar de pouco freqüente, alguns conceitos podem não ser necessários fora do escopo da ontologia. Eles podem ter sido definidos somente para esclarecer algum aspecto da ontologia, mas quando observados do ponto de vista sistêmico, eles podem não ter um papel relevante em um modelo de objetos. Nesse caso, não é necessário criar classe alguma para o conceito.

ii) São criadas classes em ambos os níveis: de Conhecimento e de Controle.

Muitas vezes, um conceito de uma ontologia é necessário nos dois níveis: no de conhecimento, determinando o tipo dos objetos concretos do mundo real, e no nível de aplicação, representando os próprios objetos do mundo real. Nessa situação, os objetos do meta-nível são utilizados pelo nível base para descrever seus objetos, por meio de uma referência (atributo), como mostra a Figura 3.7.

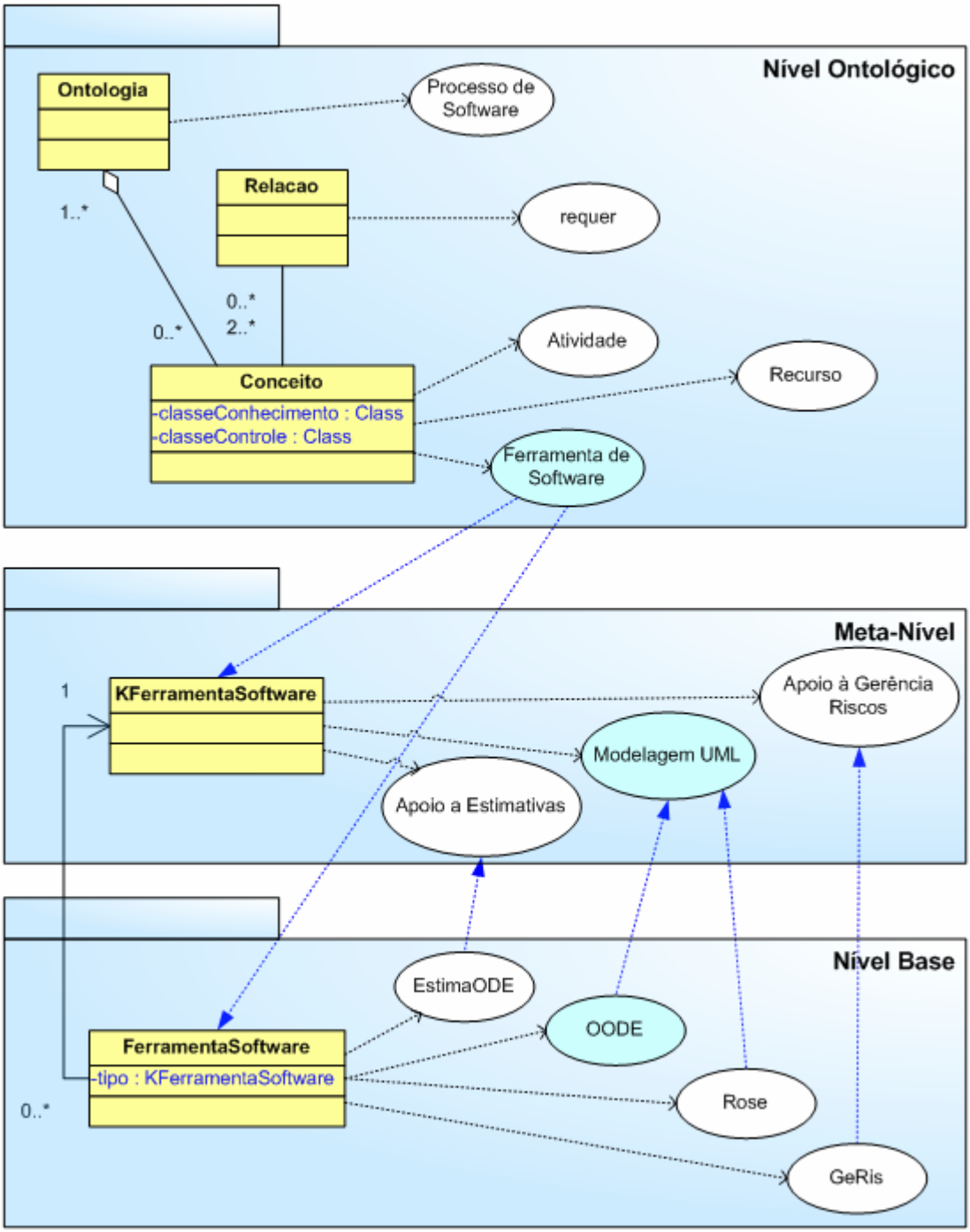


Figura 3.7 - Derivação de Classes nos Níveis de Conhecimento e de Aplicação e Amarração Semântica entre os Níveis Arquiteturais

A Figura 3.7 apresenta os três níveis arquiteturais de ODE, ilustrando, para cada nível, exemplos de classes (ícones de classes UML do lado esquerdo), instâncias (formas ovais do lado direito) e suas relações (linhas e setas). No exemplo, *Ferramenta de Software*, instância da classe `Conceito` no nível ontológico, dá origem a duas classes: `KFerramentaSoftware` no nível de conhecimento e `FerramentaSoftware` no nível base. A primeira representa os "tipos" de ferramentas potencialmente importantes em um processo de desenvolvimento de

software, tais como ferramentas de *Modelagem UML*, *Apoio a Estimativas*, *Apoio à Gerência de Riscos* etc. Essas instâncias do nível de conhecimento são utilizadas para classificar os objetos concretos do nível base (*OODE*, *Rose*, *EstimaODE*, *GeRis* etc). Dessa maneira, pode-se dizer que *OODE* é uma ferramenta de *Modelagem UML* que, por sua vez, é uma *Ferramenta de Software*.

De maneira análoga, o conceito *Atividade* dá origem às classes `KAtividade` no nível de conhecimento e `Atividade` no nível base, e podem-se utilizar essas classes para descrever, respectivamente, tipos de atividades no desenvolvimento de software (por exemplo, *Planejamento*, *Especificação de Requisitos*, *Análise* etc) e atividades concretas realizadas no contexto de um projeto específico (por exemplo, *Planejamento Inicial do Projeto X*, *Especificação de Requisitos Preliminar do Projeto X*, *Análise Orientada a Objetos do Projeto X* etc). Nesse caso, uma vez que os objetos da classe `Atividade` são anotadas por meio de referências às respectivas instâncias da classe `KAtividade`, pode-se dizer que *Planejamento Inicial do Projeto X* é uma atividade do tipo *Planejamento*.

Finalmente, como a ontologia de processo de software (Figura 3.4) define que atividades requerem recursos, pode-se descrever no nível de conhecimento que atividades do tipo *Análise* requerem ferramentas de software do tipo *Modelagem UML*. Na alocação de ferramentas para o *Projeto X*, essa informação é utilizada para apontar que *OODE e Rose* podem ser alocadas à atividade *Análise Orientada a Objetos do Projeto X*, já que são ferramentas de software compatíveis com as necessidades dessa atividade. Desta forma, o nível de conhecimento pode ser usado, também, para guiar a realização de atividades do nível base.

iii) É criada somente uma classe no Nível de Conhecimento.

Essa situação acontece quando o conceito só possui um nível relevante de instâncias. Por exemplo, seja o conceito *Procedimento*. Instâncias desse conceito incluem, dentre outros, *Análise Estruturada*, *Análise Orientada a Objetos*, *Inspeção de Código* etc. Essas instâncias são suficientes para ambos os níveis de conhecimento e de aplicação. Tomando os exemplos de atividades mencionados anteriormente, é possível dizer, no nível de conhecimento, que a atividade de *Especificação de Requisitos* pode adotar como procedimento para sua execução o método da *Análise Orientada a Objetos*. Por outro lado, pode-se dizer, no nível de aplicação, que a atividade *Especificação de Requisitos Preliminar do Projeto X* é conduzida seguindo o método da *Análise Orientada a Objetos*. Ou seja, apenas uma classe é suficiente para atender

às necessidades dos dois níveis. Nesse caso, como a classe derivada é importante para o nível de conhecimento, isto é, o que está sendo descrito por ela é, de fato, um conhecimento sobre os procedimentos que podem ser adotados no desenvolvimento de software, ela é criada no nível de conhecimento (`KProcedimento`). Como o nível base tem acesso ao nível de conhecimento, ele também pode utilizá-la quando necessário.

iv) É criada somente uma classe no Nível de Controle.

Finalmente, em algumas situações, certos conceitos originam classes somente no nível base. Isso ocorre quando o conceito só possui um nível relevante de instâncias e essas instâncias são importantes apenas no nível de aplicação, como é o caso em que um tipo não é necessário, mas os objetos concretos do mundo real o são. Seguindo o exemplo da ontologia de processo de software, este é o caso do conceito *Projeto*. Quando se fala em um projeto, está se referindo a um projeto específico, tal como o *Projeto X*, que possui as atividades *Planejamento Inicial do Projeto X*, *Especificação de Requisitos Preliminar do Projeto X* etc. Assim, esse conceito é derivado somente para o nível base, dando origem à classe `Projeto`.

Observando-se a dependência entre os níveis arquiteturais de ODE (Figura 3.2), verifica-se que o nível base depende do nível de conhecimento, que, por sua vez, depende do nível ontológico. E as classes de cada nível são desenvolvidas respeitando essas dependências. Assim, foram criadas duas formas de associar os objetos pertencentes aos três níveis:

i) as classes do nível base possuem uma associação com sua correspondente no nível de conhecimento, associando os objetos de aplicação com seus “tipos” (vide Figuras 3.6 e 3.7);

ii) a classe `Conceito` provê dois atributos para que suas instâncias possam endereçar as classes correspondentes nos níveis de conhecimento e controle. Esses atributos são do tipo `Class` (meta-classe de Java), conforme apresentado na Figura 3.7.

A primeira forma de anotação é a mais usada e a maioria das ferramentas do ambiente a utilizam para obter diretamente os tipos dos objetos que manipulam. A segunda forma é mais complexa e somente é acessada por algumas infra-estruturas mais especializadas do ambiente. O próximo capítulo a apresenta em detalhes e mostra como o seu uso pode ampliar as possibilidades de ações semânticas que podem ser exploradas no contexto de ODE.

Ambas são formas de estabelecer anotações entre os conceitos das ontologias e os objetos dos outros dois níveis. Essa anotação permite que os objetos do ambiente possam ter sua origem ontológica identificada. Assim, a navegação entre os níveis é simplificada e diversas tarefas do ambiente (tais como definição de processos, alocação de recursos, gerência de riscos, configuração do ambiente etc.) podem ser apoiadas e validadas utilizando uma perspectiva semântica, sempre suportada pela base ontológica de ODE.

3.3.3 – Aplicação da Arquitetura Conceitual em outras Infra-estruturas de ODE

A Arquitetura Conceitual estabelece a estrutura na qual ODE é desenvolvido, estruturando, além da base ontológica, seus dois principais níveis, o de Conhecimento e o de Controle. Conseqüentemente, as ferramentas que compõem o ambiente estão inseridas nessa estrutura, assim como outras infra-estruturas que são parte do ambiente. Essas infra-estruturas complementam e fortalecem a estrutura do ambiente, fornecendo apoio mais efetivo às ferramentas e aos usuários de ODE.

A Infra-estrutura de Gerência de Conhecimento de ODE

Uma infra-estrutura do ambiente que tem seu projeto intimamente ligado à arquitetura conceitual de ODE é a Infra-Estrutura de Gerência de Conhecimento, originalmente proposta em (NATALI *et al.*, 2003) e evoluída posteriormente. Seu objetivo é integrar e gerenciar o conhecimento adquirido durante o desenvolvimento de projetos de software em ODE. Para isso, ela é composta por uma memória organizacional e por um conjunto de serviços de gerência de conhecimento: criação e captura, recuperação e acesso, uso, manutenção e disseminação do conhecimento.

Conforme apontado por (STAAB *et al.*, 2001), ontologias são a cola que mantém ligadas as atividades da gerência de conhecimento, definindo um vocabulário comum a ser utilizado pelo sistema e, por conseguinte, facilitando os serviços da infra-estrutura de gerência de conhecimento. Com base nessa premissa, a estrutura da memória organizacional de ODE tem sua definição, também, fortemente apoiada em ontologias. Em decorrência da nova proposta de arquitetura conceitual de ODE desenvolvida neste trabalho, a memória organizacional do ambiente foi alterada, passando a ser estruturada segundo o modelo mostrado na Figura 3.8.

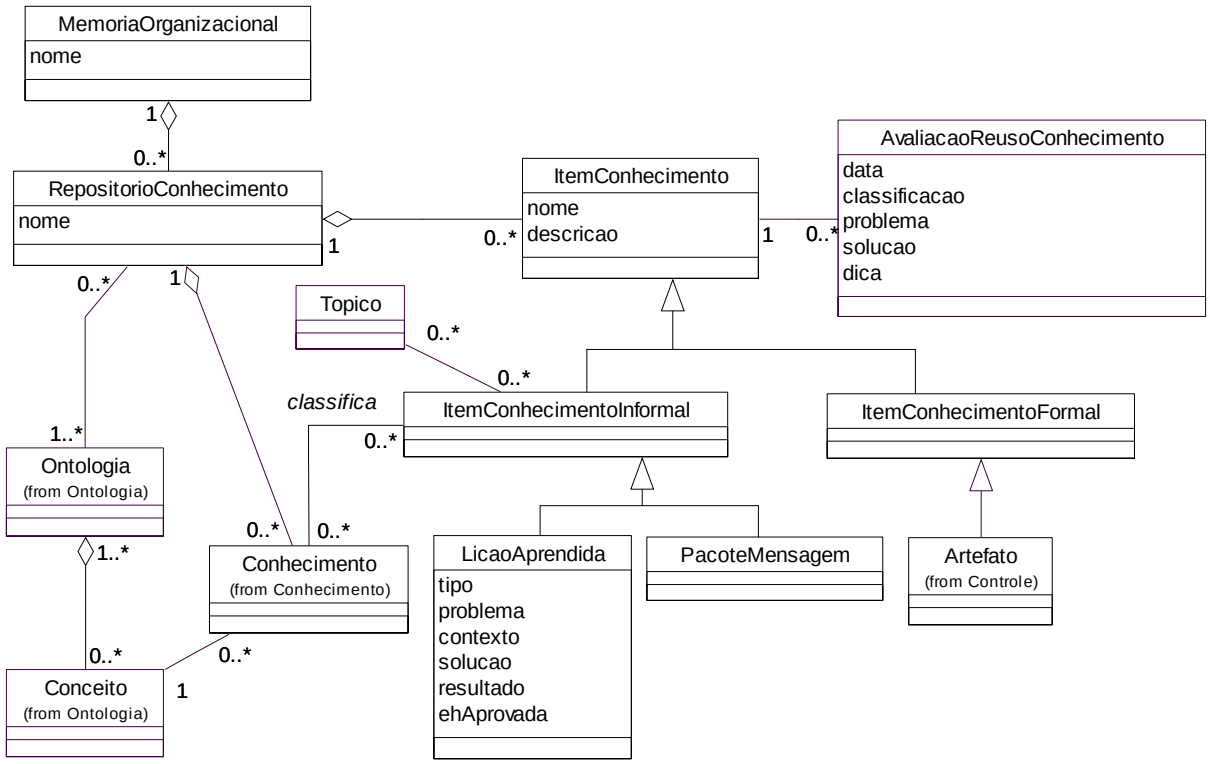


Figura 3.8 - Modelo da Infra-estrutura de Gerência de Conhecimento de ODE, adaptado de (NATALI, 2003)

Tanto os *itens de conhecimento* quanto as instâncias de ontologias (objetos da classe `Conhecimento`) compõem os *repositórios de conhecimento* do ambiente, que formam a *memória organizacional*. De fato, os objetos da classe `Conhecimento` também podem ser vistos como um tipo de *item de conhecimento formal*, já que são itens de conhecimento utilizados em várias situações no ambiente e são formalmente definidos a partir de ontologias. Entretanto, como mostra a Figura 3.8, optou-se por não colocá-los como subclasse de `ItemConhecimentoFormal`, uma vez que parte dos serviços da gerência de conhecimento (tal como a caracterização de reúso) não se aplica a esses objetos.

Vale comentar que a concepção original da infra-estrutura de Gerência de Conhecimento (NATALI *et al.*, 2003) explorava de forma simples a importante relação entre os objetos da infra-estrutura e os níveis arquiteturais de ODE. As anotações ontológicas, ainda com pouca força no ambiente, apesar de presentes em alguns casos, não eram utilizadas em todo o seu potencial.

Contudo, buscando explorar mais profundamente o aspecto semântico da infra-estrutura, esta foi remodelada, juntamente com seus serviços de busca. Fruto dessa evolução, os itens de conhecimento, formais ou informais, são anotados segundo conceitos de

ontologias. Sejam os *itens de conhecimento formais*, os *artefatos*. Eles fazem parte do nível base e possuem uma correspondência com a classe `KArtefato` do nível de conhecimento, que, por sua vez, está ligada ao conceito *Artefato* no nível ontológico. Já os *itens de conhecimento informais* (lições aprendidas e pacotes de mensagens), são classificados usando objetos da classe `Conhecimento`, correlacionando-se, assim, indiretamente a instâncias da classe `Conceito`, oriundas das ontologias.

Convém esclarecer que a associação entre as classes `Conhecimento` e `Conceito`, apresentada na Figura 3.8, é apenas uma representação do atributo `classeConhecimento` mostrado na Figura 3.7. Ele foi assim representado, pois o atributo possui como valor a classe do meta-nível criada como derivação do conceito em questão. Como todas as classes do meta-nível herdam da classe Conhecimento, a associação é possível.

A sistemática de anotação ontológica facilita a oferta de diversos serviços da gerência de conhecimento de ODE, tal como o serviço de busca de itens de conhecimento. Exemplificando, a busca de uma lição aprendida podia ter como critérios de filtro: tópicos relacionados, palavras-chave referentes à sua descrição, o projeto no qual a lição foi derivada ou, ainda, o tipo da mesma. Na evolução da infra-estrutura, pode-se contar com um dos aspectos mais relevantes da busca, que é a possibilidade de se recuperar dos repositórios de conhecimento itens de acordo com sua classificação segundo instâncias de ontologias, ou seja, permitir a busca dos itens indexados com base nas ontologias. No exemplo da Figura 3.9, deseja-se recuperar lições aprendidas que, entre outros critérios, tenham sido classificadas segundo duas instâncias da ontologia de processo de software, neste caso, instâncias do conceito *Atividade*. Assim, dois objetos da classe `KAtividade` (*Planejamento* e *Realização de Estimativas*) são usados na recuperação das lições aprendidas, utilizando as amarrações semânticas baseadas nas ontologias.

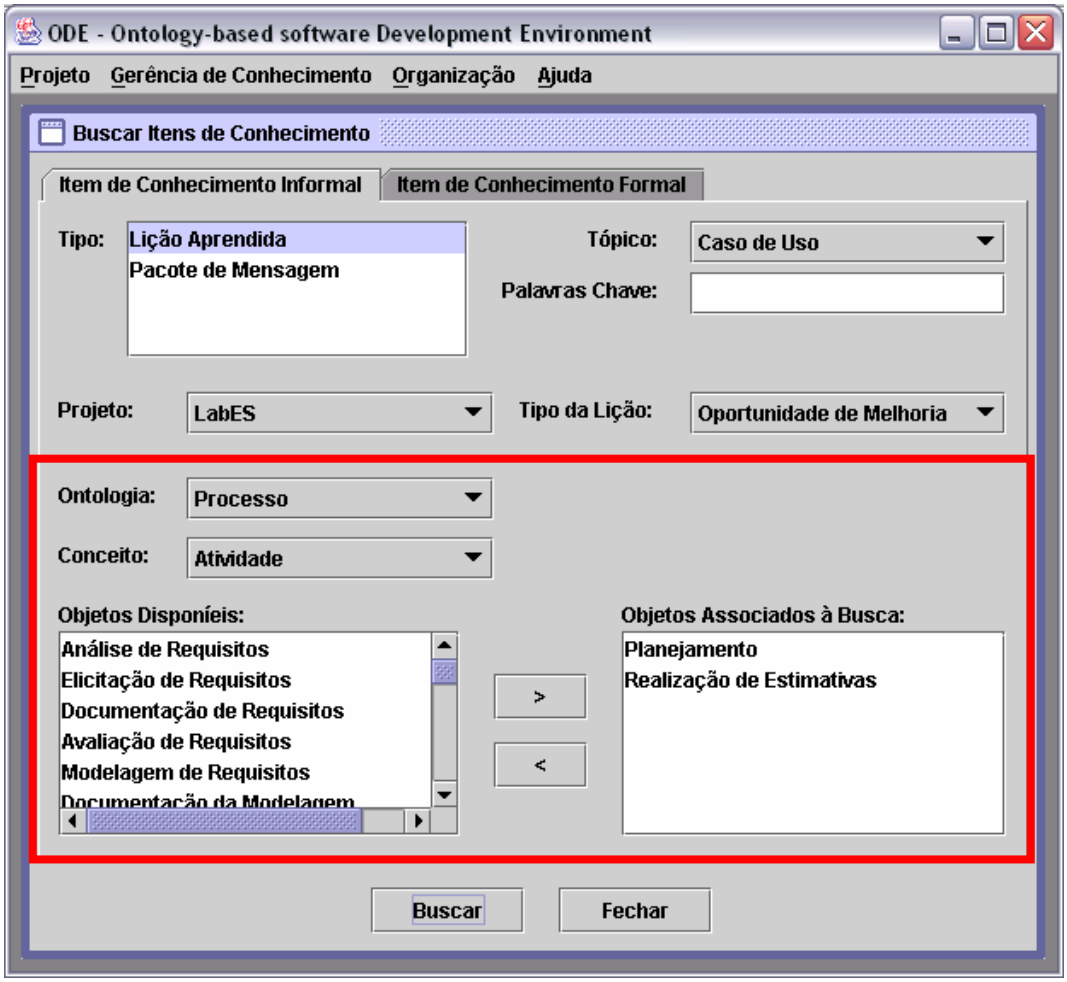


Figura 3.9 - Serviço de Busca da Gerência de Conhecimento de ODE

A Infra-estrutura de Agentes de ODE

Outra tecnologia importante aplicada a ODE, que utiliza a arquitetura conceitual e necessita de um entendimento a respeito do ambiente e de sua estrutura, são os agentes, inseridos no ambiente por meio de AgeODE (PEZZIN *et al.*, 2004). AgeODE é uma Infra-estrutura para Construção de Agentes que fornece ao ambiente o arcabouço necessário para que sejam desenvolvidos agentes de maneira padronizada, capazes de se comunicar utilizando a mesma linguagem (baseada em KQML - *Knowledge Query and Manipulation Language*), e de enviar mensagens às ferramentas e aos usuários de ODE.

Baseando-se nessa infra-estrutura, são construídos agentes responsáveis por apoiar e executar tarefas como: sugerir potenciais riscos de um projeto baseando-se em suas características, sinalizar atividades que podem ser iniciadas ou que estão em atraso nos projetos, identificar as pessoas mais adequadas para serem alocadas às atividades, disseminar

o conhecimento da memória organizacional, gerenciar preferências dos usuários etc. (PEZZIN, 2004).

Uma das premissas de AgeODE é que os agentes devem conhecer o modelo do ambiente para que possam agir. Em primeira instância, a própria comunicação é realizada com base nas ontologias, uma vez que os agentes devem "conversar na mesma língua", os parâmetros das mensagens emitidas pelos agentes incluem os conceitos e a ontologia na qual estão se comunicando. Além disso, é preciso que os agentes conheçam também os outros dois níveis da arquitetura, dado que precisam conhecer as ferramentas nas quais atuam e os objetos com os quais estão lidando para realizar suas funções.

A figura 3.10 ilustra um exemplo de comunicação entre dois agentes de informação existentes no ambiente. No exemplo, O *AgenteGerenteProjetos* questiona ao *AgenteAssistentePessoal* sobre o projeto aberto no momento. Vale observar, através dos parâmetros `:ontology` e `:content`, que é necessário que os agentes possuam conhecimento das ontologias e dos modelos do ambiente (derivado das ontologias) para que possam se comunicar.

```
ask-one
    :sender AgGerenteProjeto
    :receiver AgAssistentePessoal
    :language XML
    :ontology Processo
    :content <Ode.Controle.Cdp.Projeto/>

reply
    :sender AgAssistentePessoal
    :receiver AgGerenteProjeto
    :language XML
    :ontology Processo
    :content <Ode.Controle.Cdp.Projeto id="2:101"/>
```

Figura 3.10 – Comunicação entre Agentes utilizando Ontologias

A Infra-estrutura de Inferência de ODE

A correspondência entre ontologias e modelos de objetos pode ser ainda mais amplamente explorada através do uso de máquinas de inferência. Embora axiomas possam ser mapeados para métodos, em alguns casos, uma abordagem mais interessante é mapeá-los para regras que possam ser manipuladas por máquinas de inferência. Para lidar com isso, foi desenvolvida em ODE uma infra-estrutura de inferência (RUY *et al.*, 2004) que permite a definição de regras em Prolog a partir das várias definições existentes no ambiente, entre elas,

os axiomas ontológicos. O processamento dessas regras em conjunto com objetos do sistema por uma máquina de inferência oferece às ferramentas do ambiente um apoio inteligente na realização de suas tarefas.

Como exemplo do uso de inferências, seja o axioma da ontologia de processo de software, que diz que "*se um recurso R é usado por uma atividade A e A é sub-atividade de outra atividade B, então R também é usado por B*" ((∀ a,b,r)(uso(a,r) ∧ subatividade(a,b))→ uso (b,r)). Esse axioma pode ser convertido em uma regra da infra-estrutura de inferências (Figura 3.11) e, posteriormente, utilizado na ferramenta de definição de processos para apresentar uma sugestão mais ampla de tipos de recursos que podem ser usados por uma determinada atividade.

```
% Sub-activity Rule (A is sub-activity of B)
subActivity(A, B) :- subActivity_(A, B).
subActivity(A, B) :- subActivity_(A, X), subActivity(X, B).

% Resource Rule (Activity A uses Resource R)
use(B, R) :- subActivity(A, B), use(A, R).
```

Figura 3.11 - Regras Prolog Convertidas de Axiomas

Nesse exemplo, é possível observar também a navegação entre os níveis da arquitetura conceitual. O que a ferramenta de definição de processos possui inicialmente é uma atividade concreta "***a***" (objeto da classe `Atividade` do pacote `Controle`) e o objetivo de obter todos os tipos de recursos (objetos da classe `KRecurso` do pacote `Conhecimento`) que possam ser usados na atividade "***a***". Assim, a ferramenta obtém o tipo de "***a***", "***ka***" (objeto da classe `KAtividade`, pacote `Conhecimento`) através da associação *tipo*, que corresponde a uma das amarrações semânticas entre os níveis de controle e conhecimento. Então, "***ka***" é submetido à regra *uso*, que em sua resolução infere todos os tipos de recursos associados a "***ka***" e às suas subatividades. Finalmente, os tipos de recursos deduzidos logicamente são apresentados pela ferramenta de definição de processos como opção para uso pela atividade "***a***".

Nesse processo, uma tarefa do nível de aplicação é resolvida caminhando-se para o meta-nível (em busca dos tipos) e então para o nível ontológico (ou um mapeamento dele, como regra de inferência). Assim como no exemplo, em diversas outras tarefas do ambiente, os objetos dos níveis de aplicação ou de conhecimento podem ter suas amarrações semânticas utilizadas para realização de tarefas de forma mais inteligente, consistente e ontologicamente adequadas.

3.4. Conclusões do Capítulo

As pesquisas quanto à semântica têm se disseminado e vêm se tornando parte importante em várias áreas da computação. É essencial estabelecer conexões entre os recursos de informação produzidos e manipulados pelos sistemas para que seja possível tratá-los semanticamente, oferecendo apoio mais efetivo aos usuários.

Com os ADSs não é diferente. A área de ambientes de desenvolvimento de software é mais uma em que a semântica traz diversos benefícios. A sua aplicação pode ser utilizada para estruturar os ambientes, prover conexões entre os itens manipulados, fornecer um entendimento sobre os dados e tarefas às ferramentas e aos usuários, habilitar o uso mais efetivo de tecnologias, dentre outros, sempre com o objetivo de tornar os ambientes mais úteis.

E, neste caso, ontologias exercem um papel fundamental em ADSs como *estabelecedoras* e *organizadoras* da semântica. No caso de ODE, elas são responsáveis por estabelecer as conexões entre itens distribuídos pelos níveis arquiteturais, ou seja, elas possibilitam amarrações semânticas por meio de meta-dados ontológicos.

Essas amarrações têm sido cada vez mais utilizadas no ambiente, à medida que seu potencial é detectado. Exemplo disso são as diversas tarefas em que características semânticas são necessárias e as anotações ontológicas são utilizadas, tais como a definição de processos (BERTOLLO, 2006), alocação de recursos, identificação de riscos (SCHWAMBACH, 2004) e gerência de conhecimento. É importante notar que, atualmente, a maior parte das ações semânticas é realizada com base na principal ontologia do ambiente, a de Processo de Software. No próximo capítulo será possível acompanhar a incorporação de conhecimento organizacional à base ontológica de ODE, bem como alguns experimentos semânticos com esse novo tipo de conhecimento.

Embora as pesquisas de semântica em ADSs ainda sejam iniciais, é possível uma busca de apoio em áreas em que este aspecto é mais amplamente explorado, como a *Web* Semântica. Em diversas situações, analogias entre essas duas áreas distintas são cabíveis e podem fornecer soluções interessantes no contexto de ADSs.

Assim, um dos objetivos é ampliar o uso de ontologias, buscando incorporar a ODE meios de lidar mais efetivamente com a semântica. Em um ambiente que possui como características marcantes o uso de ontologias, a anotação dos objetos com meta-dados e a realização de inferências, uma tecnologia importante é a OWL (*Ontology Web Language*)

(SMITH *et al.*, 2004), incluindo a linguagem em si e máquinas de inferência associadas, que também será explorada no próximo capítulo.

Capítulo 4

Tratamento Semântico ao Conhecimento Organizacional em ODE

Conforme discutido no Capítulo 2, a infra-estrutura de gerência de conhecimento de ODE oferece serviços para a administração do conhecimento do ambiente. Contudo, o foco maior tem sido a gerência de conhecimento de Engenharia de Software, tendo sido dada pouca ênfase à gerência de conhecimento sobre organizações e seus membros, elementos fundamentais para o aprendizado organizacional. Assim, é necessário que esse tipo de conhecimento seja introduzido no ambiente, seguindo os princípios discutidos no Capítulo 3. Este capítulo trata dessa questão atentando para a formalização do Conhecimento Organizacional por meio de uma ontologia de organizações de software, com foco em competências, e a sua derivação para a estrutura do ambiente.

Uma vez incorporado ao ambiente, o conhecimento organizacional, assim como os outros tipos de conhecimento formalizados a partir de ontologias, pode receber um tratamento semântico. Neste capítulo é proposta uma infra-estrutura semântica baseada em uma linguagem para representação de ontologias e em uma biblioteca para sua manipulação. A infra-estrutura proposta é capaz de instanciar bases de conhecimento ontológico utilizando as ontologias e amarrações semânticas do ambiente e, a partir dessas bases, realizar inferências para deduzir novo conhecimento ou validar o existente.

Este capítulo está organizado da seguinte maneira: a seção 4.1 – A Metodologia Utilizada na Construção da Ontologia de Organizações de Software – apresenta a metodologia de construção de ontologias SABiO e uma forma de representá-las usando UML; a seção 4.2 – Uma Ontologia de Organizações de Software – mostra a construção de uma ontologia de organizações de software a partir de outras existentes na literatura; a seção 4.3 – Derivação de Classes do Ambiente a partir da Ontologia – discute como é realizada a derivação da ontologia definida em modelos de objetos e as decisões tomadas durante esse processo; a

seção 4.4 – Infra-estrutura Semântica – propõe uma infra-estrutura capaz de dar suporte à realização de serviços semânticos; na seção 4.5 – Serviços Semânticos – são apresentados alguns dos serviços suportados pela infra-estrutura definida; por fim, a seção 4.6 apresenta as considerações finais e conclusões do capítulo.

4.1. A Metodologia Utilizada na Construção da Ontologia de Organizações de Software

Na construção das ontologias desenvolvidas no contexto do Projeto ODE, tem-se utilizado o método SABiO (***S****ystematic* ***A****pproach for* ***Bu****ilding* ***O****ntologies*) (FALBO, 2004), que define um processo para construção de ontologias, cujas principais atividades são:

i) identificação do propósito e especificação de requisitos, que visa a identificar questões que a ontologia deve ser capaz de responder (questões de competência);

ii) captura da ontologia, que tem por objetivo capturar os conceitos, relações, propriedades e restrições relevantes sobre o domínio em questão;

iii) formalização, que busca escrever os axiomas da ontologia em uma linguagem formal;

Paralelamente a essas atividades, ocorrem as atividades de:

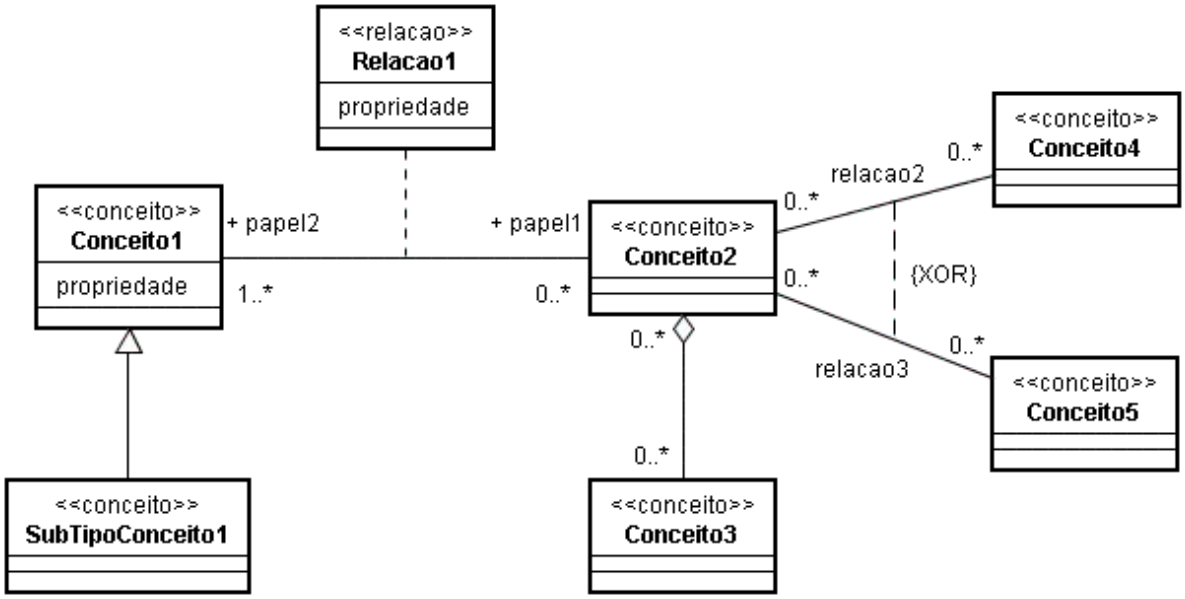
iv) integração com ontologias existentes, que visa a reutilizar conceituações existentes e integrar a ontologia em desenvolvimento a uma rede de ontologias mais ampla;

v) avaliação da ontologia, que, dentre outros, trata de avaliar se a ontologia é capaz de responder às questões de competência; e

vi) documentação da ontologia, que visa a registrar o desenvolvimento da ontologia.

Convém mencionar que, para a construção da Ontologia de Organizações de Software, as atividades ii) e iv) se confundem, pois duas características marcantes dessa ontologia são sua forte integração com a ontologia de processo de software (BERTOLLO, 2006) e o fato de que foi construída a partir de ontologias desenvolvidas em outros trabalhos de pesquisa, com destaque para (LIMA, 2004), (COTA, 2002) e (LEÃO et al., 2004) . Dessa forma, o trabalho realizado quanto a esses dois passos foi, a partir dos propósitos e questões de competência identificados, levantar as características relevantes das ontologias base e compilá-las em uma nova ontologia que atendesse aos requisitos propostos.

SABiO advoga o uso de uma linguagem gráfica para facilitar a comunicação com os especialistas do domínio, sendo que, em sua versão atual (FALBO, 2004), indica-se o perfil UML para modelagem de ontologias proposto em (MIAN, 2003), apresentado na Figura 4.1. Esse perfil UML define elementos para representar conceitos, relações e propriedades, bem como define a axiomatização associada a notações específicas para certas relações, seguindo a linha de pensamento da linguagem LINGO. Esses axiomas são ditos *independentes de domínio* e, usando o perfil proposto, estão implicitamente definidos pela linguagem de modelagem, não sendo necessário que o engenheiro de ontologias os escreva explicitamente.



Axiomas:

Todo-parte:

(AE1) ($\forall$ x) $\neg$ parteDe (x,x)
(AE2) ($\forall$ x,y) parteDe (y,x) $\leftrightarrow$ todoDe (x,y)
(AE3) ($\forall$ x,y) parteDe (y,x) $\rightarrow \neg$ parteDe (x,y)
(AE4) ($\forall$ x,y) parteDe (y,x) $\rightarrow$ ($\exists$z) (parteDe (z,x))
(AE5) ($\forall$ x,y,z) parteDe (z,y) $\wedge$ parteDe (y,x) $\rightarrow$ parteDe (z,x)

Sub-tipo-de:

(AE6) ($\forall$ x,y,z) subTipoDe (x,y) $\wedge$ subTipoDe (y,z) $\rightarrow$ subTipoDe (x,z)
(AE7) ($\forall$ x,y) subTipoDe (x,y) $\rightarrow$ superTipoDe (y,x)

Ou-exclusivo (XOR):

(AE8) ($\forall$ a $\in$ C2) (($\exists$b) (b $\in$ C3) $\wedge$ R2(a,b)) $\rightarrow \neg$ (($\exists$c $\in$ C4) $\wedge$ R3(a,c)))
(AE9) ($\forall$ a $\in$ C2) (($\exists$c) (c $\in$ C4) $\wedge$ R3(a,c)) $\rightarrow \neg$ (($\exists$b $\in$ C3) $\wedge$ R2(a,b)))

Figura 4.1 – Perfil UML para Construção de Ontologias e seus Axiomas (MIAN, 2003)

4.2. Uma Ontologia de Organizações de Software

A Ontologia de Organizações de Software tem como propósitos formalizar o conhecimento sobre organizações de software, fornecer um vocabulário comum que possa ser utilizado para representar conhecimento útil para os engenheiros de software sobre as organizações envolvidas em projetos de software e permitir que esse conhecimento seja manipulado por ODE, fornecendo apoio mais efetivo a seus usuários.

Alguns dos benefícios esperados com o desenvolvimento e utilização dessa ontologia no ambiente ODE são:

- estabelecimento de uma estrutura formalmente definida para tratar o conhecimento organizacional;
- facilitação da integração de ferramentas em várias de suas dimensões (assim como as demais ontologias de ODE);
- crescimento da base ontológica de ODE, por meio da integração com as outras ontologias que fazem parte da base ontológica do ambiente.
- possibilidade de trabalhar semanticamente com a estrutura organizacional e o conhecimento acerca do capital intelectual introduzidos ao ambiente.

Essa ontologia foi desenvolvida com base nos trabalhos de (LIMA, 2004), (COTA, 2002) e (LEÃO *et al.*, 2004), que, por sua vez, se basearam nos projetos TOVE (*Toronto Virtual Enterprise)* (FOX *et al.*, 1996) e *Enterprise* (USCHOLD *et al.*, 1998) e em (FALBO, 1998). Além disso, a ontologia de organizações de software resultante está integrada a outras ontologias do ambiente, mais especificamente com a Ontologia de Processo de Software (BERTOLLO, 2006), da qual alguns conceitos são utilizados, e com a Ontologia de Requisitos de Software (NARDI *et al.*, 2006), que utiliza conceitos desta ontologia.

A Figura 4.2 apresenta as relações da ontologia de Organizações de Software com as demais ontologias do Projeto ODE.

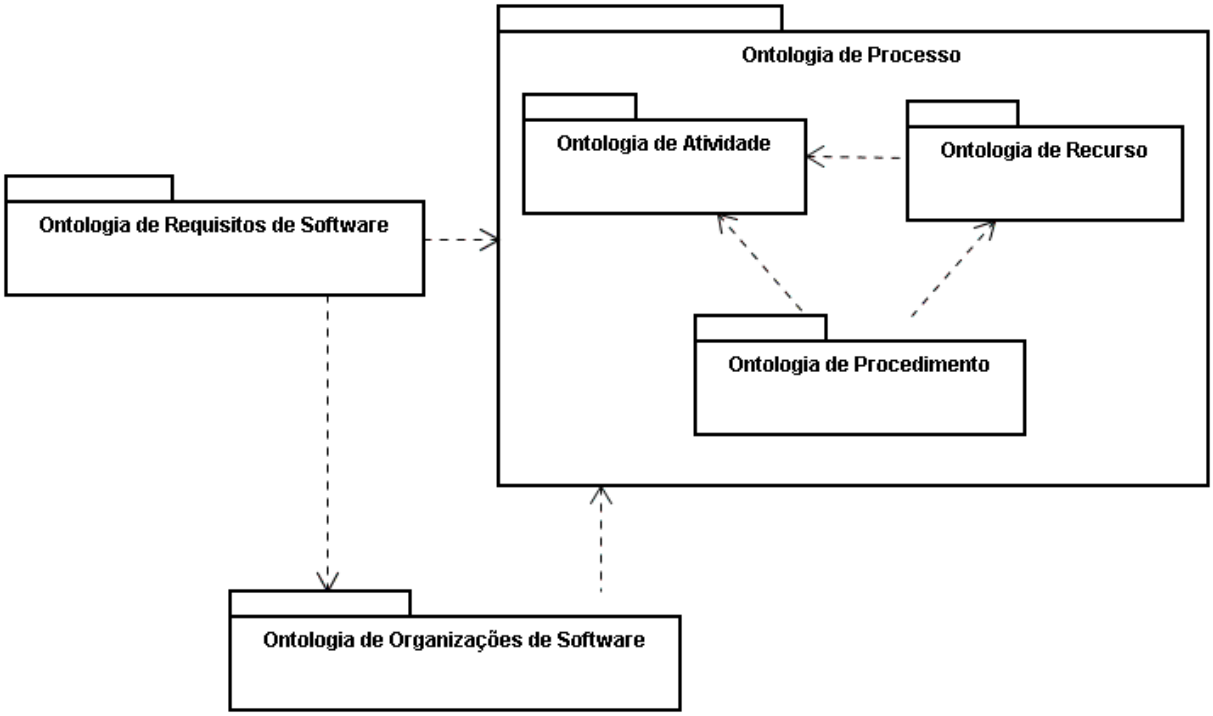


Figura 4.2 – A Ontologia de Organizações de Software e suas Relações com as demais ontologias da Base Ontológica de ODE

Para o desenvolvimento da Ontologia de Organizações de Software, alguns aspectos importantes foram considerados, como o que um ADS precisa conhecer das organizações nas quais está implantado para que possa fornecer apoio mais efetivo.

Dessa forma, itens como a estrutura da organização, sua interação com os projetos, papéis existentes, equipes de trabalho, distribuição das pessoas por unidades, cargos e projetos na organização, competências acumuladas por pessoas, requeridas por atividades ou demandas para que se possa assumir um papel, entre outros, foram considerados na determinação do escopo da ontologia. Além disso, é importante levar em conta alguns cenários que ocorrem no cotidiano das organizações atuais:

- Quando está sendo definida uma equipe para um projeto, é preciso encontrar as pessoas mais adequadas para executar as atividades do projeto;
- Quando algum membro da organização necessita de apoio na execução de suas tarefas, é importante localizar quais as pessoas que estão aptas a apoiá-lo;
- Quando se deseja elaborar um programa de treinamento, é relevante levantar quais são as principais necessidades da organização e das pessoas que a compõem;
- É importante saber em quais unidades organizacionais as pessoas estão lotadas e em quais projetos trabalham, ou trabalharam.

Esses requisitos foram traduzidos em Questões de Competência. São elas:

- Em relação às Competências Organizacionais:

  QC1. Quais pessoas na organização possuem uma determinada competência?
  QC2. Em quais atividades da organização uma determinada competência é requerida?
  QC3. Por quais cargos da organização uma determinada competência é demandada?
  QC4. A que domínio de conhecimento pertence um conhecimento?
  QC5. Quais os domínios de conhecimento nos quais a organização atua?

- Em relação à Estrutura da Organização:

  QC6. Como a organização é decomposta em unidades organizacionais?
  QC7. Como as unidades organizacionais relacionam-se entre si?
  QC8. Como as pessoas estão distribuídas nas unidades da organização?
  QC9. Quais os cargos e papéis existentes na organização?
  QC10. Quais as pessoas que possuem um determinado cargo na organização?
  QC11. Quais pessoas compõem uma determinada equipe?

- Em relação aos Projetos da Organização:

  QC12. Quais são as pessoas que estão alocadas a uma determinada atividade?
  QC13. Quais são os projetos nos quais a organização participa?

Uma vez levantadas as questões de competências e examinados os trabalhos anteriormente citados, passou-se à fase de captura da Ontologia de Organizações de Software. Dada a complexidade do domínio, a mesma foi dividida em duas sub-ontologias: de Competências e de Estrutura Organizacional. A seguir, as mesmas são apresentadas com destaque para os modelos gráficos elaborados usando o perfil UML proposto (MIAN *et al.*, 2003). Além disso, definições de termos e axiomas são também apresentados.

4.2.1 – Sub-Ontologia de Competências

A sub-ontologia de Competências, apresentada na Figura 4.3, estabelece o vocabulário necessário para descrever as competências de uma organização e quais pessoas as acumulam, respondendo às questões de competência QC01 a QC05. Como o ponto principal dessa sub-ontologia são pessoas e competências, pode-se dizer que ela trata também de aspectos ligados à cultura organizacional. Os seguintes aspectos são tratados pela sub-ontologia: taxonomia de competência, organização de conhecimento e acúmulo de competências.

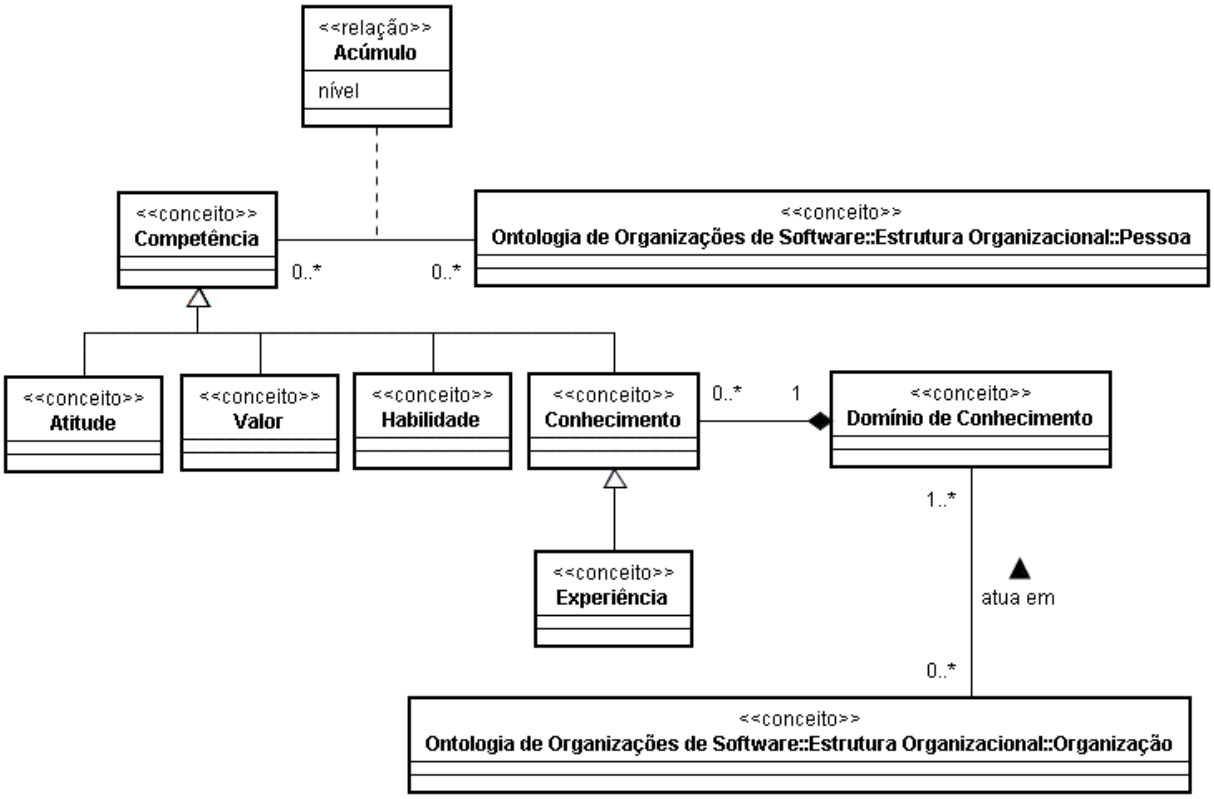


Figura 4.3 – Sub-Ontologia de Competências

A taxonomia de competências define como as competências podem ser classificadas de acordo com a sua natureza. A classificação ocorre segundo os seguintes conceitos (LIMA, 2004), (LEÃO *et al.*, 2004):

- Competência: característica que torna uma pessoa capaz de executar atividades que envolvem algum grau de dificuldade (LIMA, 2004). Competências podem ser classificadas em Atitude, Valor, Habilidade, Conhecimento e Experiência.
- Conhecimento: apropriações de objetos pelo pensamento através de definição, percepção clara, análise, apreensão completa ou outra forma de apropriação

(HOLANDA, 1999). Por exemplo: conhecimento sobre orientação a objetos, sobre UML, sobre uma norma de qualidade, sobre linguagens de programação etc.

- Experiência: conhecimento mais aprofundado, adquirido através da prática, ou seja, através da execução de atividades (LIMA, 2004). Por exemplo: experiência em projeto OO, na gerência de projetos, na escrita de artigos etc.
- Habilidade: aptidões natas ou adquiridas não associadas a uma atividade ou domínio específico (LIMA, 2004). Por exemplo: liderança, atenção concentrada, habilidade de negociação, de argumentação, facilidade de trabalho em equipe etc.
- Atitude: diz respeito aos aspectos sociais e afetivos relacionados ao trabalho. São determinantes do comportamento, pois estão relacionadas com percepção, personalidade, aprendizagem e motivação (FLEURY *et al.*, 2000).
- Valor: representa as percepções e crenças, motivação, caráter, costumes, ética, moral, consciência social, comportamento, regras da vida em sociedade e da conduta entre os indivíduos, que determinam seus deveres entre si e para com a sociedade (MASLOW, 2000).

As competências do tipo Conhecimento e, por conseguinte, Experiência são organizadas segundo Domínios de Conhecimento. As Organizações atuam em, ao menos, um domínio de conhecimento, adotando-se as seguintes definições (LIMA, 2004):

- Domínio de Conhecimento: conjunto de conhecimentos reunidos de acordo com a homogeneidade de conteúdo. Por exemplo: domínio de engenharia de software, de inteligência artificial, de administração, de literatura.
- atua em: Organizações atuam em Domínios de Conhecimento. Ou seja, possuem capital intelectual relativo aos domínios e executam atividades que requerem conhecimentos pertencentes a esses domínios.

As competências são um ponto chave na organização, podendo ser acumuladas por pessoas e associadas a outros aspectos importantes organizacionalmente. O conceito de Pessoa e a relação acúmulo são definidos conforme a seguir (LIMA, 2004):

- Pessoa: indivíduo fundamental ao funcionamento de uma organização, atuando na execução de atividades necessárias ao comprimento da missão da organização. Ao

longo da trajetória profissional, pessoas acumulam competências, disponibilizando-as para a organização em que trabalham. As competências possuídas pelos profissionais de uma organização são de grande importância para os próprios profissionais, pois são utilizadas para estabelecer o seu papel e seu valor na organização, e também para a organização, pois representam o seu capital intelectual.

- Acúmulo: Pessoas acumulam Competências. Esta relação indica quais as competências acumuladas por uma pessoa e em que nível.

Alguns axiomas independentes de domínio, como os oriundos da hierarquia apresentada, são atribuídos à ontologia. Entretanto, conforme citado anteriormente, como podem ser derivados diretamente do modelo, os mesmos não são apresentados. Neste capítulo são apresentados apenas os dependentes do domínio. Esses axiomas podem ser de dois tipos: axiomas de consolidação e axiomas de derivação. O primeiro impõe restrições que precisam ser atendidas para que uma relação seja estabelecida consistentemente. O último pretende representar o conhecimento declarativo que pode derivar novo conhecimento a partir de conhecimento factual representado na ontologia, mas que não é capturado pela estruturação de conceitos e relações da ontologia.

4.2.2 – Sub-Ontologia de Estrutura Organizacional

A sub-ontologia de Estrutura Organizacional, apresentada na Figura 4.4, estabelece o vocabulário necessário para descrever a estrutura de organizações de software e como elas se relacionam com seus projetos, processos e pessoal, respondendo às questões de competência QC06 a QC13. Os seguintes aspectos são tratados por essa sub-ontologia: estruturação da organização e suas unidades; contratação e lotação de pessoas; competências necessárias a atividades e recursos humanos e participação da organização em projetos. Os dois últimos itens são tratados na interface com a ontologia de processo de software.

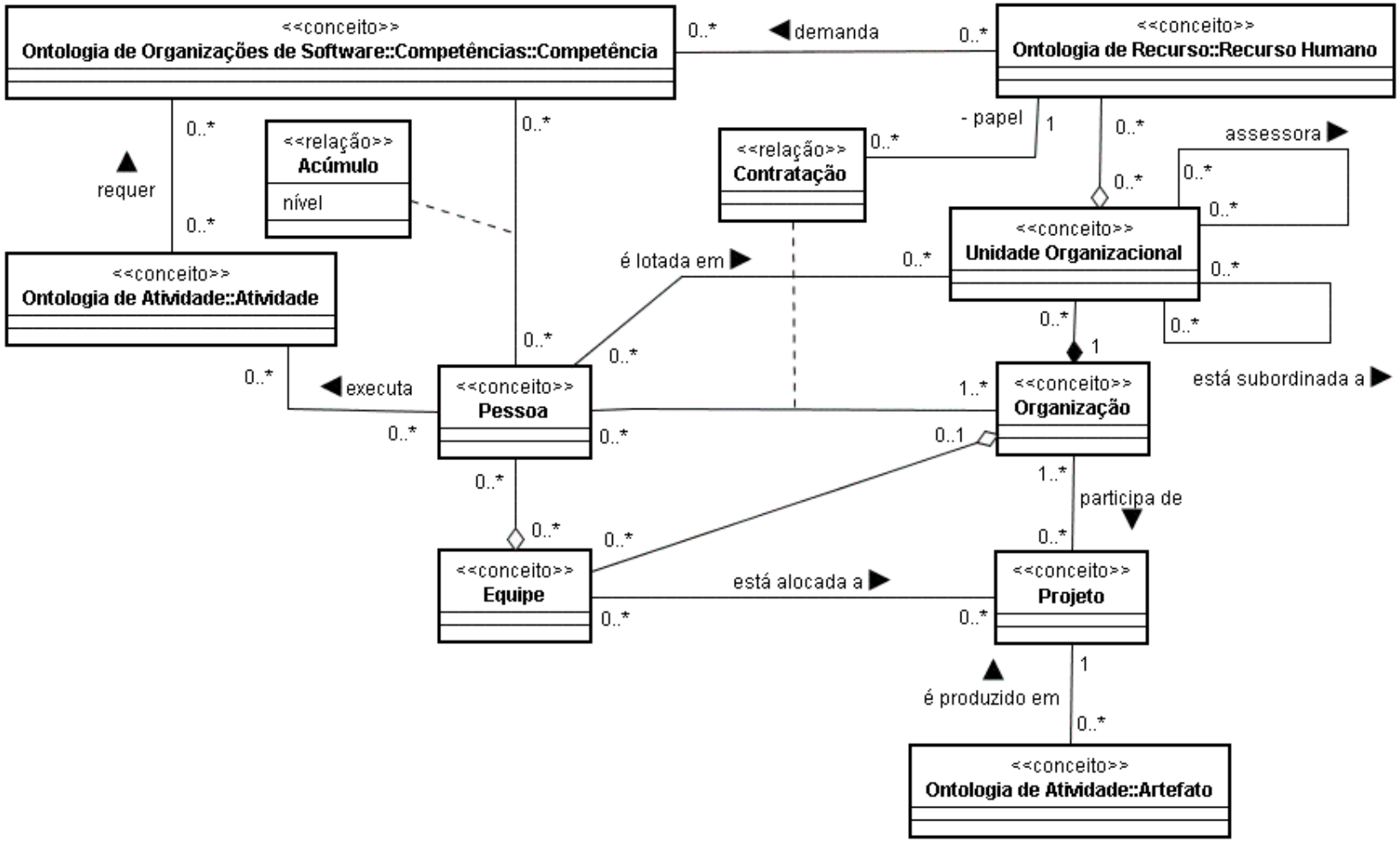


Figura 4.4 – Sub-Ontologia de Estrutura Organizacional

Uma organização, um dos conceitos centrais desta ontologia, é decomposta em unidades organizacionais, que, por sua vez, estão associadas entre si por relações de subordinação ou assessoria (LIMA, 2004).

- Organização: grupo de pessoas trabalhando em conjunto para o cumprimento de uma missão. Segundo (MEGGINSON *et al.*, 1986), a divisão do trabalho é o princípio base das organizações, permitindo que grupos de pessoas, trabalhando em conjunto, de modo cooperativo e coordenado, possam realizar mais do que cada uma agindo de forma independente.
- Unidade Organizacional: agrupamento de componentes da organização (atividades e pessoas) de acordo com a homogeneidade de conteúdo para que a organização possa ser econômica e eficiente. Por exemplo: departamento pessoal, setor de produção, divisão de vendas, unidade administrativa.
- está subordinada a: indica que uma Unidade Organizacional está subordinada ou precisa se reportar a outra para realizar suas atividades.
- assessora: indica que uma Unidade Organizacional assessora, ou presta serviços a outra.

Em organizações de software, entidades que têm o capital intelectual como seu bem de mais alto valor, o conceito de Pessoa está fortemente presente e é relacionado a diversos outros conceitos. Pessoas são contratadas para uma organização, na qual assumem papéis e são lotadas em unidades organizacionais. Além disso, pessoas acumulam competências e executam atividades nas equipes dos projetos em que estão alocadas.

Vale esclarecer uma adaptação feita em relação à Ontologia de Processo de Software (BERTOLLO, 2006). Nela, o conceito de Recurso Humano significa um “agente humano necessário para realização de uma atividade”, podendo instanciar as pessoas em si que realizam uma atividade ou, em um nível mais abstrato, os cargos ou papéis que seriam necessários à execução, tais como gerente de projeto, analista, desenvolvedor. Na ontologia de Organizações de Software, essas duas visões são separadas em conceitos distintos: Recurso Humano, que passa a representar somente as instâncias de titulação, e recebe o sinônimo de Cargo, e Pessoa, conceito já apresentado, responsável por representar os seres humanos

contratados na organização. Naturalmente, essas pessoas possuem um cargo, ou melhor, assumem o papel de um Recurso Humano quando contratadas em uma organização.

- Contratação: Pessoas são contratadas para Organizações, assumindo papéis.
- papel: Pessoas, quando contratadas para uma organização, exercem o papel de um Recurso Humano (Cargo).
- está lotada em: Pessoas, após contratadas, são lotadas em Unidades Organizacionais.

Devido à sua grande integração com a Ontologia de Processo de Software, há diversas relações entre as duas ontologias. O conceito de *Atividade* está presente, uma vez que, para que atividades sejam executadas, são necessárias Pessoas e requisitadas Competências. Presente também está o conceito de Recurso Humano, já relacionado a Pessoa, e que compõe o plano de cargos de uma Unidade Organizacional. Além disso, um Recurso Humano, visto como um cargo, também demanda Competências.

Além disso, há uma forte relação com Projetos e Processos, dado que as Pessoas compõem Equipes que estão alocadas a Projetos dos quais a Organização participa. Segue a definição dos conceitos e relações envolvidos nesta ontologia:

- Atividade: ação básica que transforma artefatos de entrada (insumos) em artefatos de saída (produtos) (FALBO, 1998).
- Equipe: agrupamento de pessoas com finalidade determinada e, normalmente, por período limitado (LIMA, 2004). Equipes podem estar relacionadas a projetos (por exemplo: equipe de desenvolvimento, equipe de analistas) ou somente à organização (equipe de treinamento, equipe de testes, equipe de auditores).
- Projeto: empreendimento requerendo esforço organizado, visando ao desenvolvimento ou manutenção de um produto específico. Compreende um conjunto gerenciado de recursos inter-relacionados que entrega produtos a um cliente ou usuário final e tipicamente opera de acordo com um plano. Ex: Projeto do Portal do LabES.
- executa: Pessoas executam Atividades (LIMA, 2004).
- requer: Atividades requerem Competências como recurso intelectual (LIMA, 2004).
- demanda: Recursos Humanos (cargos) demandam Competências.

- está alocada a: Equipes são alocadas a Projetos (LIMA, 2004).
- participa de: Organizações participam de Projetos (LIMA, 2004).

Algumas das relações apresentadas possuem mais detalhes do que uma simples definição. Dessa forma, parte da semântica existente entre os conceitos e relações da Ontologia de Organizações de Software é formalizada por meio dos axiomas discutidos a seguir.

O primeiro axioma diz respeito à execução de atividades por pessoas (relação *executa*): *se uma pessoa acumula alguma competência requisitada por uma atividade, então essa pessoa pode participar da execução dessa atividade*. Ou, expresso em lógica de primeira ordem:

$$(\forall\ p,a,c)\quad (acumulo(p,c,\_) \wedge requer(a,c) \rightarrow executa(p,a)) \qquad (A1)$$

onde os predicados `acumulo(p,c,_)`, `requer(a,c)` e `executa(p,a)` indicam, respectivamente, que uma pessoa `p` acumula uma competência `c` em algum nível, uma atividade `a` requer uma competência `c` como recurso intelectual e uma pessoa `p` pode executar uma atividade `a`.

Outro axioma, semelhante ao anterior, diz que: *se uma pessoa acumula as competências demandadas por um recurso humano (cargo), então a pessoa pode ser contratada assumindo esse papel*.

$$(\forall\ p,r,c)\quad ((demanda(r,c_1) \wedge \ldots \wedge demanda(r,c_n)) \wedge$$
$$((acumulo(p,c_1,\_) \wedge \ldots \wedge acumulo(p,c_n,\_)) \rightarrow contratacao(p,\_,r)) \qquad (A2)$$

onde os predicados `demanda(r,c)` e `contratacao(p,_,r)` indicam, respectivamente, que um recurso humano (cargo) `r` demanda uma competência `c`, e uma pessoa `p` pode ser contratada por alguma organização assumindo o papel `r`.

Os dois axiomas acima estabelecem um conhecimento que não é representado nos modelos da ontologia e são bastante úteis na alocação de pessoas (A1) e na contratação ou determinação dos papéis das pessoas (A2). Além desses, mais um axioma, este de consolidação, é definido.

Organizações que atuam em domínios de conhecimento contratam pessoas que possuem competências. Então, *se uma organização atua em um domínio de conhecimento, esta deve ter pessoas contratadas que acumulem algum conhecimento deste domínio de conhecimento*.

```
(∀ o,d) (atuaEm(o,d) → (∃ p,k) (contratacao(p,o,_) ∧
                                acumulo(p,k,_) ∧
                                parteDeDomínioConhecimento(k,d))) (A3)
```

onde os predicados `atuaEm(o,d)` e `parteDeDomínioConhecimento(k,d)` indicam, respectivamente, que uma organização `o` atua em um domínio de conhecimento `d`, e um conhecimento `k` faz parte do domínio de conhecimento `d`.

4.3. Derivação de Classes do Ambiente a partir da Ontologia

Uma vez definida a Ontologia de Organizações de Software, ela foi incorporada a ODE, criando as devidas instâncias do nível ontológico.

Para derivação da ontologia nos outros dois níveis, Conhecimento e Controle, foi utilizada, parcialmente, a abordagem sistemática de derivação de infra-estruturas de objetos a partir de ontologias (FALBO *et al.*, 2002), discutida no Capítulo 3. Considera-se o uso da abordagem como parcial, pois esta prevê, entre outras transformações, a derivação de axiomas em métodos, o que não foi realizado neste caso, pois os axiomas foram derivados em regras de uma linguagem lógica, conforme discutido na seção 4.4.

Durante a derivação, algumas decisões foram tomadas para determinar como cada conceito e relação seriam refletidos nos níveis de Conhecimento e Controle. Ou seja, apenas no meta-nível, apenas no nível base, em ambos os níveis ou em nenhum deles.

Essas decisões foram tomadas apoiadas por algumas características que valem ser lembradas: primeiramente, o fato da Ontologia de Organizações de Software ser fortemente integrada à Ontologia de Processo de Software, que já faz parte do ambiente e tem suas classes derivadas como parte fundamental do núcleo de ODE. Isso faz com que a estrutura definida originalmente pela ontologia de processos tenha alguns privilégios em relação à da ontologia de organizações em um momento de definição e implementação dos modelos. Um segundo aspecto é a tendência de se incorporar à aplicação apenas as características que são aplicáveis e relevantes ao ambiente. Dessa forma, algumas relações foram desconsideradas e

outras tiveram suas cardinalidades restringidas, tornando algumas partes da estrutura mais simples.

Por outro lado, algumas características não expressas ontologicamente são úteis em cada um dos outros dois níveis. Assim, algumas classes receberam novos atributos que possuem um sentido e uma utilidade nos níveis de Conhecimento e Controle.

Cada uma dessas decisões é exposta durante a apresentação dos modelos, a seguir.

4.3.1 - Modelo de Objetos do Meta-Nível

O modelo do meta-nível agrupa as classes de conhecimento, cujos objetos descrevem ou caracterizam os objetos do nível base.

A hierarquia de competências se encaixa bem nesse padrão. Os objetos dessas classes são tipicamente utilizados para descrever as competências acumuladas por pessoas, demandadas por recursos humanos ou requeridas por atividades Neste sentido, são conhecimento acerca das competências e, portanto, os conceitos *Competência*, *Conhecimento*, *Experiência*, *Habilidade*, *Atitude* e *Valor* dão origem a classes apenas do nível de conhecimento, conforme mostra a Figura 4.5. O mesmo acontece com *Domínio de Conhecimento*, que origina `KDominioConhecimento`, responsável por agrupar os objetos de `KConhecimento` e caracterizar as áreas de atuação de uma organização de software.

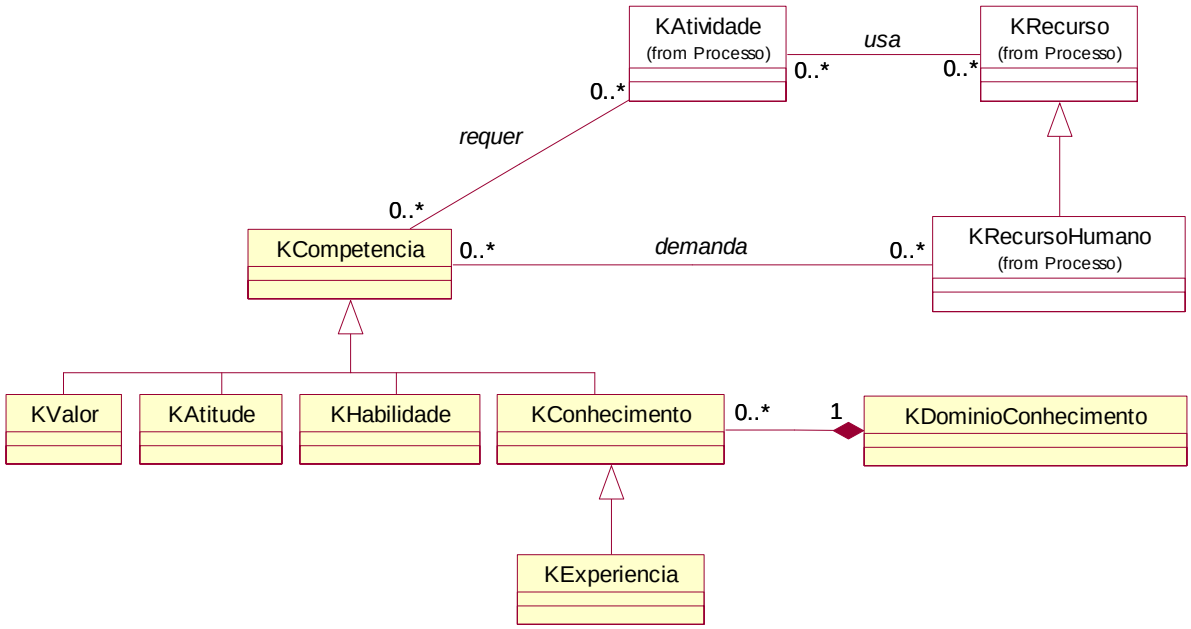


Figura 4.5 – Modelo de Objetos do Meta-Nível (Pacote `Competencia`)

4.3.2 - Modelo de Objetos do Nível Base

O modelo de classes do pacote Controle, apresentado na Figura 4.6, trata das classes cujos objetos são instâncias concretas, geralmente criadas no contexto de algum projeto ou, neste caso, de alguma organização.

Iniciando pelo conceito de *Organização*, sua derivação se dá diretamente para uma classe no nível Base. A classe `Organizacao` recebe o atributo `missao`, derivado de sua propriedade ontológica, e o atributo `nome`, requisito de nível de aplicação. A relação *atua em* é transposta para a associação `atua`, que relaciona a organização a seus domínios de conhecimento. O conceito de *Unidade Organizacional* também é derivado para o nível de controle, criando a classe `UnidadeOrganizacional`, cujos objetos compõem uma organização e que possuem planos de cargos compostos por tipos de recursos humanos (`KRecursoHumano`). As relações *está subordinada a* e *assessora* não foram derivadas neste momento.

Voltando à organização, a relação *participa de* também se mantém. Entretanto, ela é restringida, permitindo que os projetos do ambiente sejam conduzidos por uma única organização.

Na verdade, a decisão tomada foi admitir o uso do ambiente por uma única organização. Alguns estudos foram realizados no sentido de trabalhar com conhecimento multi-organizacional, pois traria benefícios em diversas situações, principalmente no que se refere à gerência de conhecimento. Porém, a complexidade envolvida extrapolou o escopo deste trabalho.

Com uma única organização registrada no ambiente, algumas relações se restringem ou perdem seu sentido e não são mapeadas. É o caso da relação de composição de *Organização* com *Equipe*. Dado que todas as equipes pertencem a projetos da única organização, não é mais necessário mapear quais equipes compõem a organização.

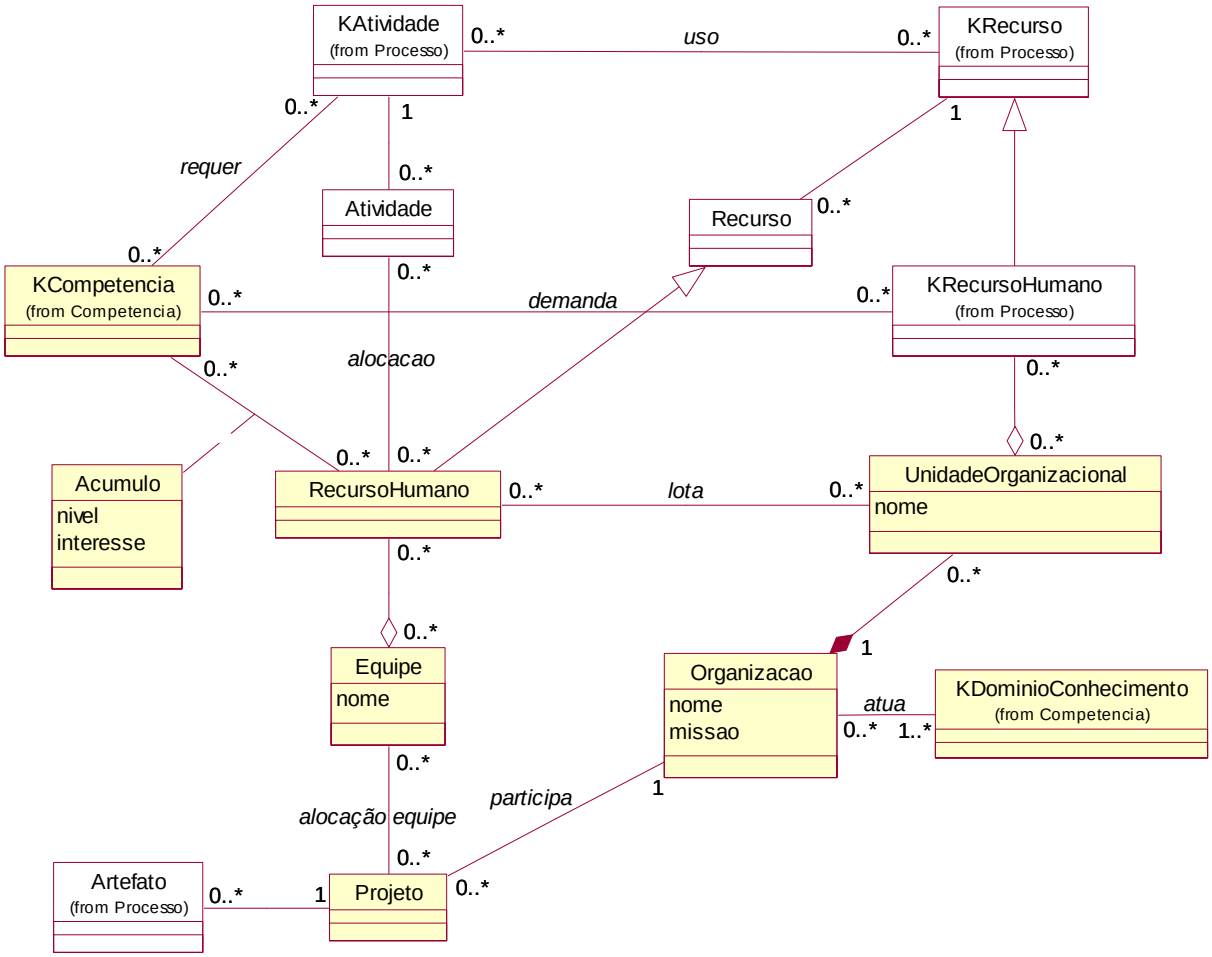


Figura 4.6 – Modelo de Objetos do Nível Base (Pacote `Controle`)

Outra derivação relevante ocorre com o conceito *Pessoa*. As instâncias desse conceito são um subconjunto da antiga concepção dada pelo conceito de *Recurso Humano*, que agrupava, além das titulações recebidas pelas pessoas, também as próprias pessoas. Em decorrência disso, em ODE, existem as classes `KRecursoHumano`, representando os cargos ou papéis das pessoas, e `RecursoHumano`, representando as próprias pessoas. Assim, devido aos impactos da alteração no núcleo do ambiente serem relativamente altos, a opção foi admitir a classe `RecursoHumano` como a derivação do conceito *Pessoa*, uma vez que as instâncias são as mesmas e os significados se equivalem.

Partindo dessa opção, no modelo de classes do pacote *Controle*, equipes são compostas por recursos humanos (pessoas), e recursos humanos (pessoas) são lotados em unidades organizacionais. Além disso, a relação *executa* é representada pela associação `alocacao`, que também já existia no nível base. A relação *contratação* perde seu sentido quando o ambiente assume uma única organização. Entretanto a relação *papel* é mantida,

admitindo, como derivação, a associação `tipo`, existente entre `RecursoHumano` e `KRecursoHumano`.

Finalmente, a relação *acúmulo* é derivada para o nível base, originando a classe `Acumulo`, que associa os objetos `RecursoHumano` e `KCompetencia`, atribuindo um `nivel` de acúmulo de competência a uma pessoa. O atributo `interesse` foi acrescentado para possibilitar serviços personalizados de distribuição de conhecimento.

A Tabela 4.1 apresenta um sumário do mapeamento feito, listando os conceitos da Ontologia de Organizações de Software e suas respectivas classes derivadas nos níveis de conhecimento e controle.

Tabela 4.1 - Conceitos e Classes Derivadas

Origem	Conceitos / relações	Classes do Meta-Nível	Classes do Nível Base
Competências	*Competência*	`KCompetencia`	-
	Conhecimento	`KConhecimento`	-
	Experiência	`KExperiencia`	-
	Habilidade	`KHabilidade`	-
	Valor	`KValor`	-
	Domínio de Conhecimento	`KDominioConhecimento`	-
	Pessoa	-	`RecursoHumano`
	Acúmulo	-	`Acumulo`
Estrutura Organizacional	*Organização*	-	`Organizacao`
	Unidade Organizacional	-	`UnidadeOrganizacional`
	Equipe	-	`Equipe`
	Projeto	-	`Projeto`
	Contratação		
Processo de Software	*Atividade*	`KAtividade`	`Atividade`
	Artefato	`KArtefato`	`Artefato`
	Recurso	`KRecurso`	`Recurso`
	Recurso Humano	`KRecursoHumano`	-

4.4. Infra-estrutura Semântica

Conforme apresentado no Capítulo 3, a Arquitetura Conceitual de ODE possui três níveis (Ontológico, Meta-Nível e Base) dependentes entre si. Essas dependências são ligações entre as classes e objetos, com alguma origem ontológica, de cada um dos níveis e são chamadas anotações ontológicas, ou de forma mais geral, amarrações semânticas.

As anotações são utilizadas de duas formas:

i) por meio da associação `tipo`, responsável por amarrar as classes do nível base às respectivas classes do nível de conhecimento;

ii) acessando os atributos `classeConhecimento` ou `classeControle` pertencentes à classe `Conceito`, do nível ontológico. Esses atributos são do tipo `class` e apontam para as classes derivadas de cada conceito nos pacotes *Conhecimento* (meta-nível) e *Controle* (nível base).

A segunda forma, embora mais poderosa, vem sendo utilizada apenas por infra-estruturas mais especializadas do ambiente e depende de reflexão computacional para se obter as classes e objetos desejados. Essa dificuldade de acesso e manipulação das anotações da classe `Conceito` limita em parte o potencial semântico do ambiente.

Outra característica limitante dessa abordagem é que essa informação somente pode ser manipulada por uma linguagem orientada a objetos. Muitas definições ontológicas, principalmente axiomas, podem ser manipuladas mais adequadamente através de linguagens lógicas, principalmente quando inferências podem ser utilizadas.

Dado que há uma ontologia definida, derivada em modelos de conhecimento e controle, com suas instâncias criadas e devidamente anotadas com conhecimento ontológico, o uso de toda essa informação não deve ser restringido, seja por dificuldades de uso ou por características de linguagens.

Dessa forma, partiu-se para a proposição de uma infra-estrutura capaz de manipular de forma mais abrangente o conhecimento ontológico, oferecendo facilidades aos desenvolvedores do ambiente e permitindo que serviços semânticos sejam mais amplamente utilizados para apoiar as tarefas do ambiente.

Para o desenvolvimento da infra-estrutura, foi necessário pesquisar qual meio utilizar para representar adequadamente as ontologias no ambiente e manipular os indivíduos das ontologias de forma a obter respostas aos questionamentos necessários, usando uma máquina de inferência.

Com a disseminação da *Web* Semântica, uma linguagem para representação de ontologias que vem se tornando popular é OWL (*Ontology Web Language*) (SMITH *et al.*, 2004). OWL é a especificação de uma linguagem para ontologias desenvolvida pela W3C e faz parte da lista de recomendações da W3C relacionadas ao desenvolvimento da *Web* Semântica. Esta linguagem oferece mecanismos para representar explicitamente o significado dos termos e dos relacionamentos entre eles (SMITH *et al.*, 2004), ou seja, conceitos e relações. Além disso, OWL permite que sejam representadas propriedades, restrições e indivíduos de uma ontologia.

OWL deriva de outras duas linguagens, OIL (*Ontology Inference Layer*) (HORROCKS *et al.*, 2000) e DAML (*DARPA Agent Markup Language*) (DAVIES *et al.*, 2003). OIL foi a primeira dessas linguagens e teve como principal requisito a facilidade de adoção por parte dos desenvolvedores, servindo principalmente à comunidade ligada à *Web* semântica. Mais adiante, as duas foram unidas, dando origem à linguagem DAML+OIL (CONNOLLY *et al.*, 2001) que, então, deu origem a OWL. Além disso, essas linguagens são influenciadas por RDF (*Resource Description Framework*) (BRICKLEY *et al.*, 1999), linguagem de representação de informação na *Web*, cujos objetivos são criar um modelo simples de dados, com uma semântica formal e um vocabulário baseado em XML. A Figura 4.7 mostra essa evolução até se chegar à OWL.

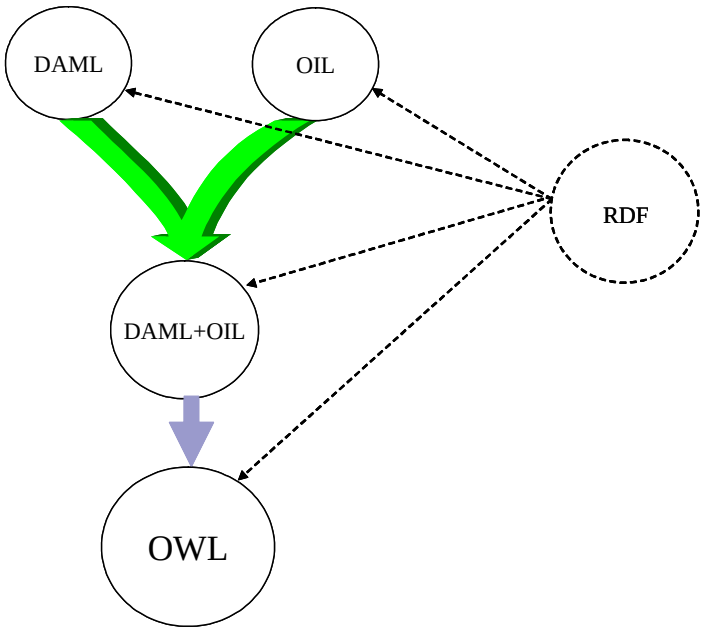


Figura 4.7 – Origem da Linguagem OWL

Assim, a linguagem OWL possui as características do RDF e um vocabulário maior que DAML+OIL, oferecendo mais recursos para utilização. Com isso, OWL é projetada para

ser utilizada por aplicações que necessitam realizar o processamento do significado das informações antes de apresentá-las aos usuários (SMITH *et al.*, 2004).

Os recursos que a linguagem OWL oferece são divididos em três sub-linguagens (SMITH *et al.*, 2004):

- **OWL *Lite***: oferece uma hierarquia de classificação e funcionalidades de restrições simples. Possui baixa complexidade formal.
- **OWL DL** (*Description Logics*): oferece máxima expressividade ao mesmo tempo em que seus sistemas mantêm a completude (garantia que todas as conclusões serão executadas) e decidibilidade (todos os cálculos terminarão em tempo finito). OWL DL inclui todos os itens da linguagem, mas impõe restrições quanto à utilização.
- **OWL *Full***: oferece máxima expressividade e a liberdade sintática do RDF. Ainda permite que uma ontologia aumente o significado do vocabulário pré-definido. Entretanto, não é esperado que algum software suporte todas as suas características.

OWL *Full* pode ser vista como uma extensão do RDF, enquanto OWL *Lite* e OWL DL são extensões de uma visão delimitada de RDF. Portanto, cada uma dessas três sub-linguagens é uma extensão de seu predecessor mais simples.

OWL DL foi projetada para suportar a lógica descritiva (*Description Logics*) e possui propriedades computacionais desejáveis em sistemas que usam capacidades de raciocínio.

Uma ontologia OWL pode incluir descrições de classes, propriedades e seus indivíduos. Dada uma ontologia, a semântica formal de OWL especifica como derivar suas conseqüências lógicas, ou seja, fatos não literalmente presentes na ontologia, mas implícitos semanticamente (SMITH *et al.*, 2004).

Com base na pesquisa realizada, a Infra-estrutura Semântica adotou a linguagem OWL, mais especificamente, seu subconjunto OWL DL para representação de ontologias. Uma vez estabelecido como descrever ontologias, é necessário definir uma forma de manipular o conhecimento contido nas ontologias. Assim, buscou-se uma biblioteca que pudesse ser facilmente integrada ao ambiente, capaz de manipular ontologias em OWL e, além disso, que oferecesse capacidades de inferência.

Jena (MCBRIDE *et al.*, 2004) é uma API Java usada na criação e manipulação de grafos RDF, incluindo OWL. A API foi projetada para representar os modelos, recursos, propriedades e literais do RDF.

Para o suporte à manipulação de ontologias, Jena oferece um pacote específico, chamado *Jena 2 Ontology API*, que possui classes para leitura e manipulação de ontologias em OWL. Jena oferece também o *OWL Reasoner*, uma máquina de inferência integrada à API que provê suporte para realização de inferências sobre o OWL DL.

Definidas a linguagem para representação (OWL DL) e a biblioteca para acesso, manipulação e realização de inferências sobre ontologias (Jena), o projeto da Infra-Estrutura pôde ser estabelecido, e é apresentado a seguir.

4.4.1 – Modelo da Infra-Estrutura Proposta

Com o objetivo de experimentar no ambiente ODE serviços de natureza semântica mais aprofundados, foi projetada a infra-estrutura apresentada na Figura 4.8, cujo processo para realização dos serviços consiste de três etapas principais, organizadas em passos:

- *Importação da Ontologia*: momento em que uma ontologia é incorporada ao ambiente, definindo os objetos do nível ontológico e suas amarrações com os outros dois níveis. Basicamente envolve os passos de edição da ontologia e importação de sua estrutura.

- *Criação da Base de Conhecimento Ontológica*: etapa em que, a partir da representação de uma ontologia importada para o nível ontológico, os objetos do ambiente são recuperados e inseridos como indivíduos de um modelo OWL. Envolve dois passos: a obtenção, a partir do repositório de ODE, dos objetos do Meta-Nível (MN) e do Nível Base (NB), e a conversão desses objetos para indivíduos OWL.

- *Realização de Inferências*: ocorre quando uma questão é submetida à infra-estrutura que, através da máquina de inferência de Jena, gera as respostas ontologicamente corretas e as retorna à aplicação. Para tal, dois passos são realizados: Submissão de Questões para o Motor Jena e Conversão das Respostas (indivíduos OWL) para objetos (Java).

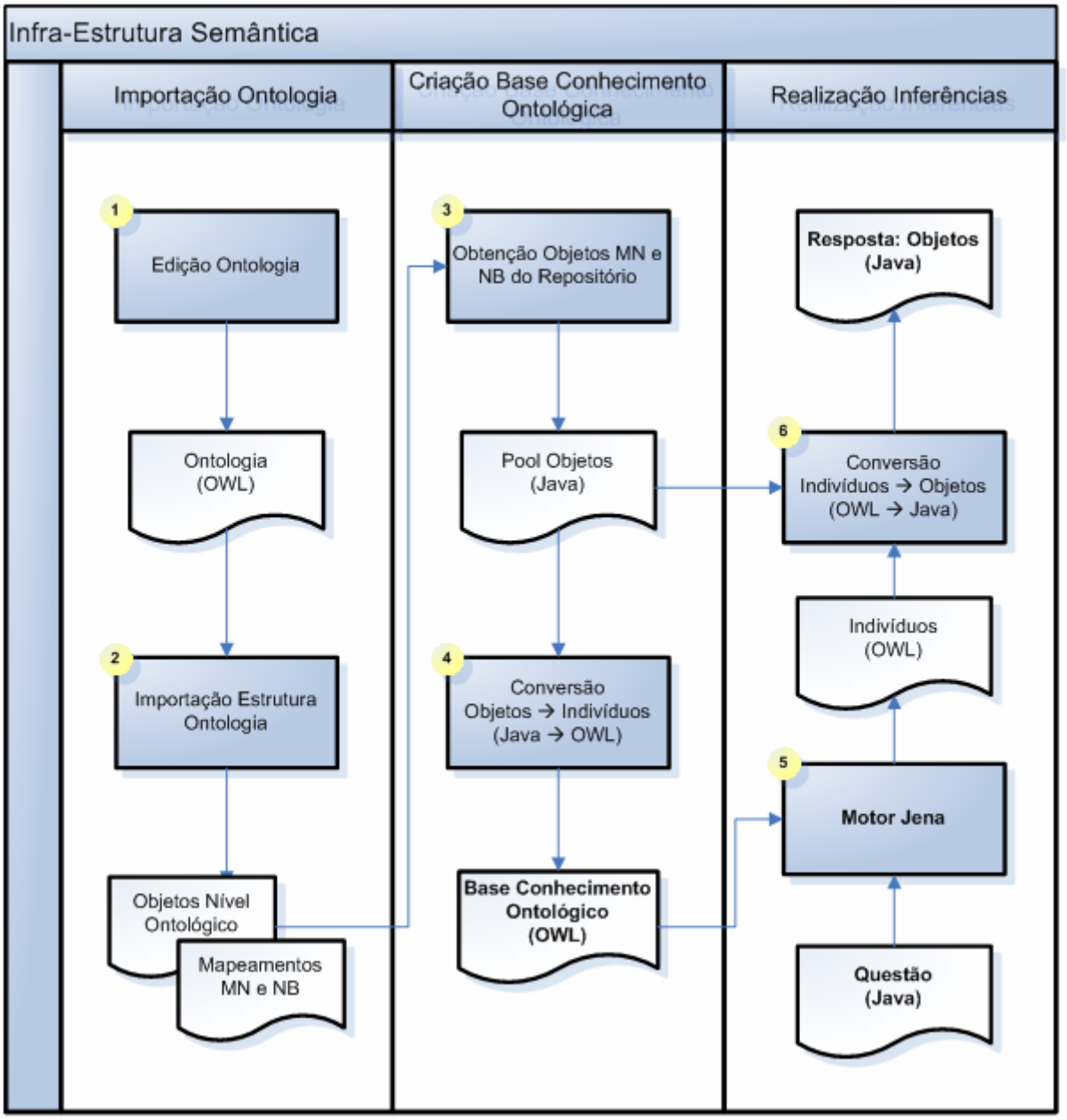


Figura 4.8 – Modelo da Infra-Estrutura Semântica

A seguir, cada uma das etapas e seus respectivos passos são detalhados. Alguns exemplos baseados na Ontologia de Organizações de Software são apresentados.

Importação da Ontologia

A primeira etapa tem por objetivo descrever uma ontologia, previamente definida, na linguagem OWL e incorporá-la ao ambiente. Isso resulta na criação dos objetos do Nível Ontológico e suas amarrações com as classes do Meta-Nível e do Nível Base.

Esta etapa é realizada apenas uma vez para cada ontologia. A partir dela, os dados gerados ficam armazenados no repositório do ambiente. A importação da ontologia consiste de dois passos:

1) Edição da Ontologia

A partir de uma ontologia definida, neste passo, ela tem seus conceitos, relações, propriedades e axiomas descritos segundo a linguagem OWL, originando um arquivo neste formato. Neste trabalho, a edição da ontologia foi realizada utilizando a ferramenta *Protégé* (PROTÉGÉ, 2006). Esta ferramenta dispõe do editor *Protégé-OWL* uma extensão de *Protégé* que suporta a edição de OWL. A Figura 4.9 mostra a edição da Ontologia de Organizações de Software.

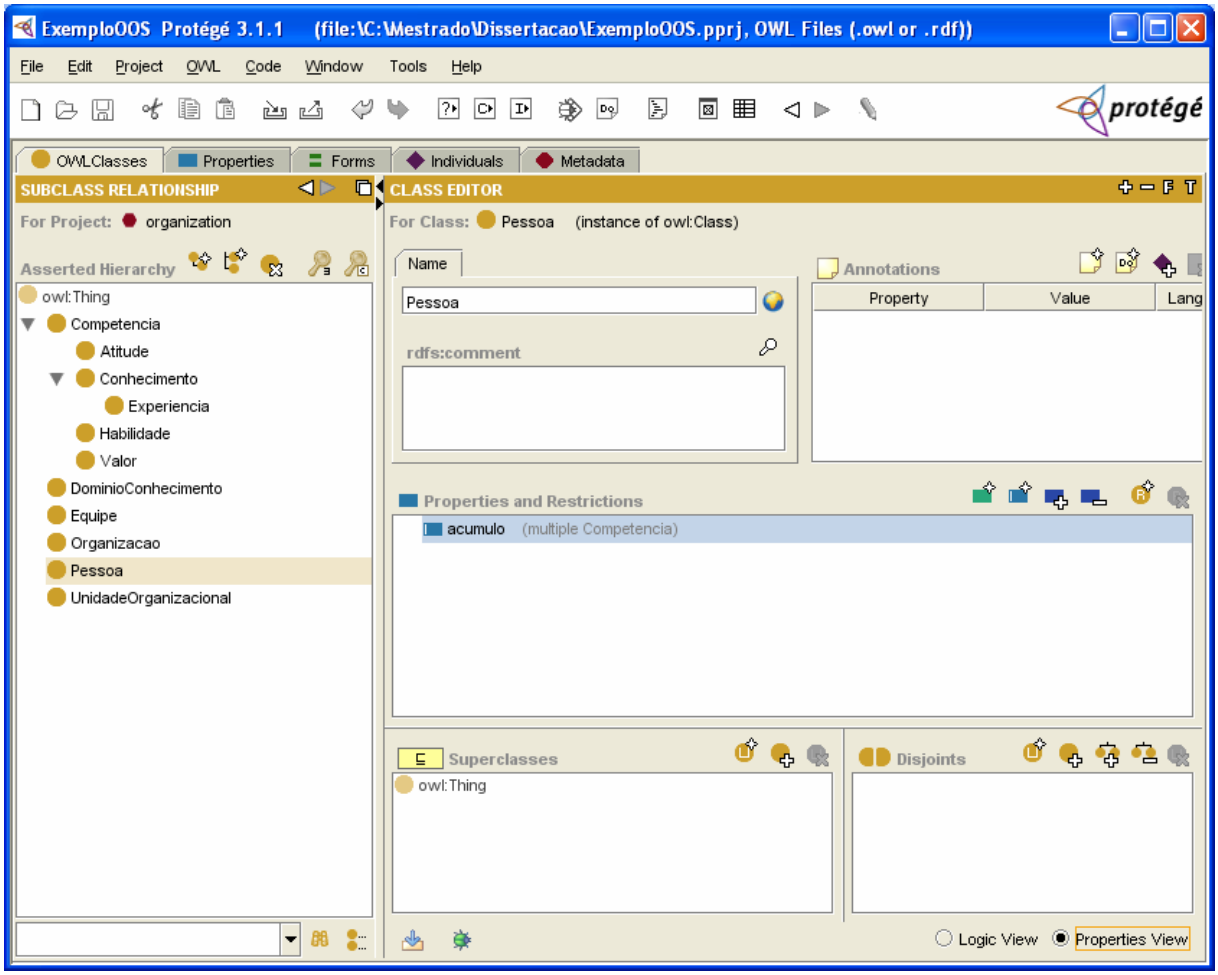


Figura 4.9 – Edição da Ontologia de Organizações de Software no *Protégé*

Após a definição das classes, propriedades e restrições OWL, a ferramenta gera um arquivo contendo a ontologia no formato da linguagem. A Figura 4.10 exibe parte do arquivo gerado, apresentando a definição do conceito *Pessoa* e sua relação *acúmulo*.

```
<owl:Ontology rdf:about="#OrganizacaoSoftware" />
...
<owl:Class rdf:about="#Pessoa" />
<owl:ObjectProperty rdf:about="#acumulo">
    <rdfs:domain rdf:resource="#Pessoa" />
    <rdfs:range rdf:resource="#Competencia" />
</owl:ObjectProperty>
...
```

Figura 4.10 – Definição do conceito *Pessoa* em OWL

2) Importação da Estrutura da Ontologia

Este passo é responsável pela incorporação da ontologia OWL nos níveis arquiteturais do ambiente. A partir dele, as definições da ontologia passam para o ambiente, instanciando os objetos do nível ontológico e suas amarrações com os demais níveis. De maneira geral, os itens devem ser mapeados conforme o esquema mostrado na Tabela 4.2.

Tabela 4.2 – Mapeamento OWL – Objetos de ODE

OWL	Nível Ontológico	Níveis de Conhecimento e Controle
Ontology	Ontologia	(Pacote)
Class	Conceito	Classe
ObjectProperty	Relação	Associação / Classe
DatatypeProperty	Propriedade	Atributo
Restriction	(Axioma)	(Método)

Por exemplo, uma classe OWL é importada como um objeto da classe `Conceito` do nível ontológico, que pode ser derivado em classes dos níveis de conhecimento e/ou controle e essas amarrações são mantidas através dos atributos `classeConhecimento` e `classeControle` de `Conceito`.

Neste ponto, é relevante esclarecer que o processo de criação das amarrações entre os níveis não é feito de maneira completa. Conforme apresentado no Capítulo 3, os conceitos do nível ontológico são devidamente amarrados às suas classes derivadas nos outros dois níveis. Entretanto, somente a amarração entre os conceitos e classes não é suficiente para a criação de uma base de conhecimento completa. Dessa forma, a próxima atividade (3 – Obtenção de Objetos do Meta-Nível e Nível Base a partir do Repositório) é capaz de obter, automaticamente, somente os objetos, que são transformados em indivíduos em OWL. As

associações entre os objetos e os valores dos atributos não são possíveis de serem recuperados automaticamente, pois não há amarrações entre as *relações* (nível ontológico) e as associações (demais níveis), ou entre as *propriedades* (nível ontológico) e os atributos (demais níveis).

Na verdade, essas amarrações são possíveis se forem estabelecidos atributos nas classes do nível ontológico, da mesma forma como foi feito com a classe `Conceito`. Entretanto, à medida que mais itens são mapeados, a complexidade e a quantidade de variações de mapeamentos possíveis crescem muito. Enquanto um *conceito* pode derivar entre zero e duas classes, uma *relação*, por ser binária, pode derivar entre zero e quatro associações, com cardinalidades variadas, dependendo das opções tomadas no processo de derivação. Assim, cada vez mais, este deixa de ser um processo automático, devido à quantidade de variações possíveis no processo de derivação de ontologias, tornando necessária a criação de um novo módulo capaz de interagir com o usuário para capturar as várias decisões tomadas durante o processo de derivação.

Esse módulo foi retirado do escopo deste trabalho e optou-se por realizar o mapeamento da forma mais simples, relacionando-se apenas os conceitos e classes. Para suprir a falta de amarrações entre as relações e propriedades, durante a atividade 4, é realizada uma simulação para que sejam obtidas as associações e atributos desejados, que são inseridos na instanciação da ontologia.

Criação da Base de Conhecimento Ontológica

A segunda etapa tem a finalidade de instanciar a ontologia OWL, criando em um arquivo, os indivíduos (instâncias) existentes como objetos do ambiente. Durante a utilização do ambiente, constantemente, objetos são criados e manipulados. Nesta etapa, os objetos com origem em algum conceito ontológico são mapeados como indivíduos na linguagem OWL.

3) Obtenção de Objetos do Meta-Nível e do Nível Base a partir do Repositório de ODE

Tomando como base os atributos `classeConhecimento` e `classeControle` da classe `Conceito`, é possível identificar as classes derivadas de cada conceito e obter seus objetos do repositório de dados do ambiente. Esses objetos são, então, incluídos em um *pool* de objetos para que sejam convertidos em indivíduos.

4) Conversão de Objetos para Indivíduos OWL

Neste passo, a ontologia em OWL, que teve apenas a estrutura da ontologia definida (conceitos, relações e propriedades), é instanciada. Assim, a ontologia passa a conter também as instâncias (indivíduos).

Os objetos recuperados do repositório são convertidos em indivíduos OWL e acrescentados aos devidos conceitos. Além disso, é realizada a recuperação dos valores das associações e atributos para que estes indivíduos recebam os valores de suas propriedades OWL (relações e propriedades) e a ontologia possa ser instanciada por completo.

A Figura 4.11 apresenta o método de criação das instâncias a partir de um conceito e uma lista de objetos.

```
/* Tranforma Objetos em Indivíduos e os insere no Modelo da Ontologia.
*/
public void criarIndividuos(OntClass classe, List listaObjetos) {

  for (int i=0; i<listaObjetos.size(); i++) {
    ObjetoPersistente objeto = (ObjetoPersistente) listaObjetos.get(i);

    Individual ind = classe.createIndividual(NS + objeto.obterIDO());
    ind.addLabel(objeto.toString(), null);
  }
}
```

Figura 4.11 – Criação de Instâncias no Modelo OWL

O resultado desse passo é uma base de conhecimento no formato de um arquivo OWL completo, contendo a estrutura da ontologia e os indivíduos, a partir da qual é possível realizar os questionamentos desejados. Esta base de conhecimento pode variar em razão da ontologia escolhida. Dessa forma, em ODE, podem ser criadas bases de conhecimento sobre processos, riscos, qualidade, organizações etc.

Esse passo cria uma base de conhecimento estática em relação ao ambiente. Ou seja, as modificações nos objetos do ambiente realizadas após a criação da base de conhecimento não são refletidas nela, a menos que o passo seja executado novamente.

Realização de Inferências

A terceira e última etapa tem o intuito de permitir a realização de deduções lógicas sobre a base de conhecimento criada. A API Jena é utilizada para compor o Motor Jena,

responsável por receber os questionamentos do ambiente e responder segundo uma perspectiva ontológica.

5) Motor Jena

O Motor Jena foi desenvolvido para encapsular a API de mesmo nome, provendo aos desenvolvedores do ambiente uma forma facilitada de utilizar os serviços semânticos.

Uma vez instanciada uma base de conhecimento, o método **questionarModelo**(relacao, valor) da classe BaseConhecimento, exibido na Figura 4.12, pode ser invocado para questionar o modelo da ontologia, inferir os indivíduos cujo valor é desejado e retornar os objetos correspondentes aos indivíduos obtidos como resultado.

```
/* Questiona ao modelo da Ontologia os Indivíduos que podem assumir um
valor. */
public List questionarModelo(String relacao, ObjetoPersistente valor) {
  InfModel modeloInfer = ModelFactory.createInfModel(reasoner, modelo);

  OntProperty prop = modelo.getOntProperty(NS + relacao);
  Individual ind = modelo.getIndividual(NS + valor.obterIDO());

  List listaResultado = new ArrayList();
  Iterator i = modeloInfer.listSubjectsWithProperty(prop, ind);
  while (i.hasNext()) {
    Resource resp = (Resource)i.next();
    listaResultado.add(resp.getLocalName());
  }

  return converterIndividuos(listaResultado);
}



/* Converte os Indivíduos da lista passada nos Objeto originais. */
private List converterIndividuos(List listaIndividuos) {
  List listaObjetos = new ArrayList();
  for (int i=0; i<listaIndividuos.size(); i++) {
    String ido = (String) listaIndividuos.get(i);
    ObjetoPersistente objeto = ObjetoPersistente.obterPorIDO(ido);
    listaObjetos.add(objeto);
  }

  return listaObjetos;
}
```

Figura 4.12 – Métodos para Realização de Inferências

6) Conversão de Indivíduos OWL para Objetos

O sexto e último passo é o mais simples e apenas restaura os objetos a partir de seus identificadores (ido) contidos na lista de indivíduos, utilizando o método **converterIndividuos**(listaIndividuos), apresentado na Figura 4.12.

Para exemplificar a realização da última etapa, Realização de Inferências, é apresentado um exemplo de um serviço bastante comum e útil em diversas ocasiões. Em muitas situações em uma organização de software é necessário identificar **"quem sabe o quê"**. Essa é uma informação útil quando se deseja: localizar especialistas na organização, alocar alguma pessoa a uma atividade, preparar um treinamento, montar uma equipe, direcionar uma pergunta a alguém apto a responder etc.

Esse serviço pode ser criado de forma simples, com base na ontologia de organizações. A partir da base de conhecimento instanciada, basta questionar o modelo da ontologia quanto aos objetos que possuem a relação *acúmulo* com a competência desejada. A Figura 4.13 exibe o método implementado para prover esse serviço e o resultado de sua execução.

```
/* Identifica quais pessoas acumulam a competência passada. */
public List quemSabeOQue(KCompetencia competencia) {
  return baseConhecimento.questionarModelo("acumulo", competencia);
}


----------------------------------------------------------------------
Quem sabe 'Java'?
--> Lucas
--> Aline
--> Rodrigo
--> Silvano
--> Julio
```

Figura 4.13 – Serviço "Quem Sabe o Quê"

O serviço "quem sabe o quê" pode ser tomado como base para outros serviços a serem utilizados em várias situações em ODE. Esse e outros serviços são apresentados na próxima seção, como forma de avaliar a infra-estrutura semântica proposta.

4.5. Serviços Semânticos

Com o objetivo de experimentar a infra-estrutura semântica provendo alguns serviços úteis ao ambiente, durante este trabalho foram criados serviços de gerência de conhecimento relacionados ao conhecimento organizacional apresentados nesta seção.

O primeiro serviço diz respeito à alocação de pessoas a atividades. ODE possui uma ferramenta de alocação chamada GerênciaRH, exibida na Figura 4.14, na qual as pessoas que fazem parte das equipes do projeto são alocadas às atividades definidas em um processo.

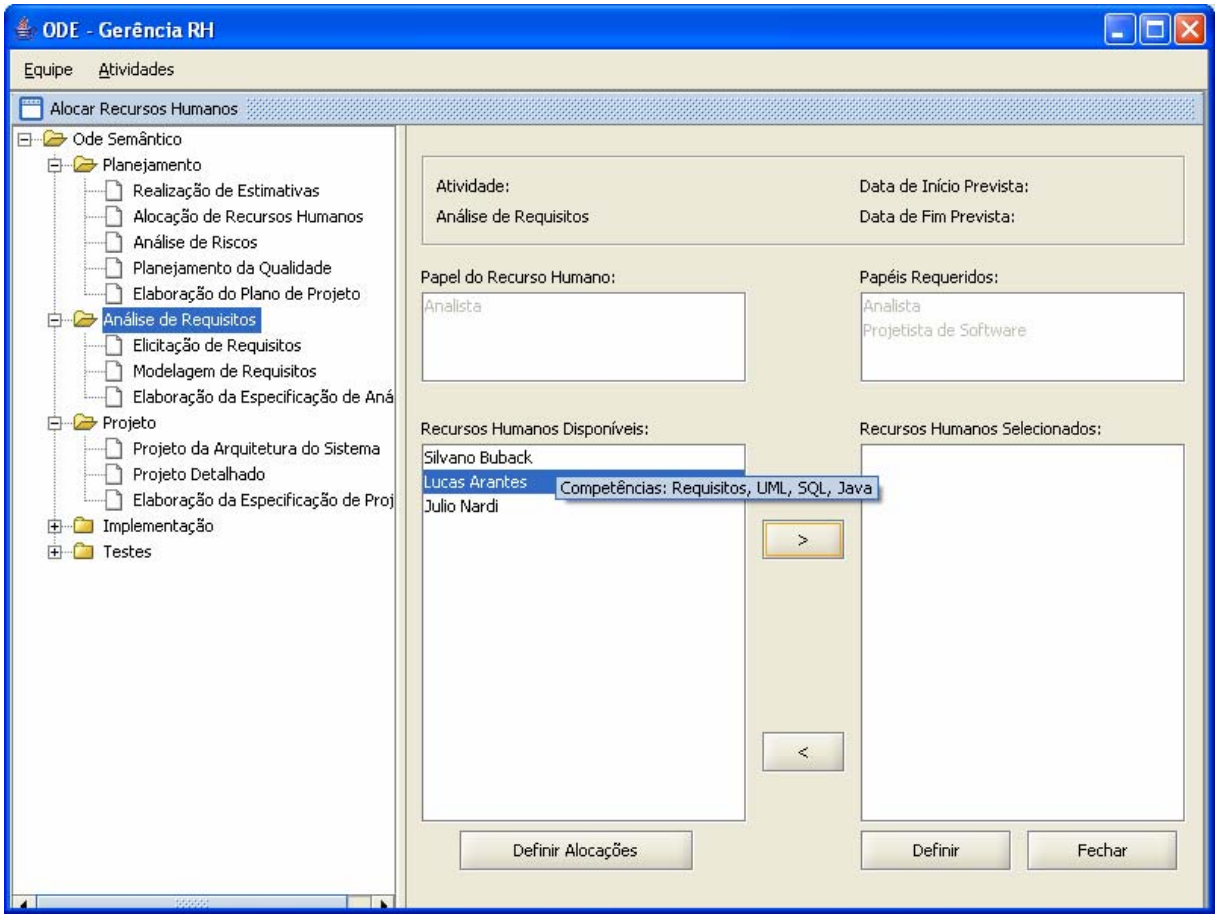


Figura 4.14 – Ferramenta GerênciaRH

Em sua forma original, a ferramenta sugeria quais pessoas poderiam executar a atividade, baseando-se em seu tipo (conhecimento). Ou seja, se uma atividade precisa de Analistas, a ferramenta sugeria todos os Analistas da equipe. Utilizando o novo serviço definido a partir da infra-estrutura, é possível realizar uma sugestão mais elaborada, baseando-se não apenas nos papéis assumidos pelas pessoas, mas também nas competências acumuladas por elas e requisitadas pelas atividades.

Esse serviço é capaz de listar as pessoas baseando-se no axioma (A1) sobre a relação *execução*, que diz que *se uma pessoa acumula alguma competência requisitada por uma atividade, então essa pessoa pode participar da execução dessa atividade*. O método que representa o serviço é mostrado na Figura 4.15.

```
/* Identifica quais pessoas podem executar a atividade passada. */
public List execucaoAtividade(Atividade atividade) {
  KAtividade tipo = atividade.obterConhecimento();
  return baseConhecimento.questionarModelo("execucao", tipo);
}
```

Figura 4.15 – Serviço "Execução Atividade"

O segundo serviço objetiva identificar quais as pessoas da organização capazes de responder a determinadas questões. Esse serviço tem sua principal utilização na ferramenta de correio eletrônico de ODE (ARANTES, 2004), exibida na Figura 4.16. Nessa ferramenta, antes de enviar as mensagens, o usuário deve classificá-las segundo as competências existentes na organização. Dessa forma, um usuário do ambiente, ao enviar um questionamento para que alguém o responda, pode: (i) preencher o campo "Destinatário" com as pessoas que suponha que irão respondê-lo ou (ii) deixar o campo "Destinatário" em branco e permitir que a ferramenta utilize o serviço "Quem Sabe o Quê" (Figura 4.13) para enviar a mensagem.

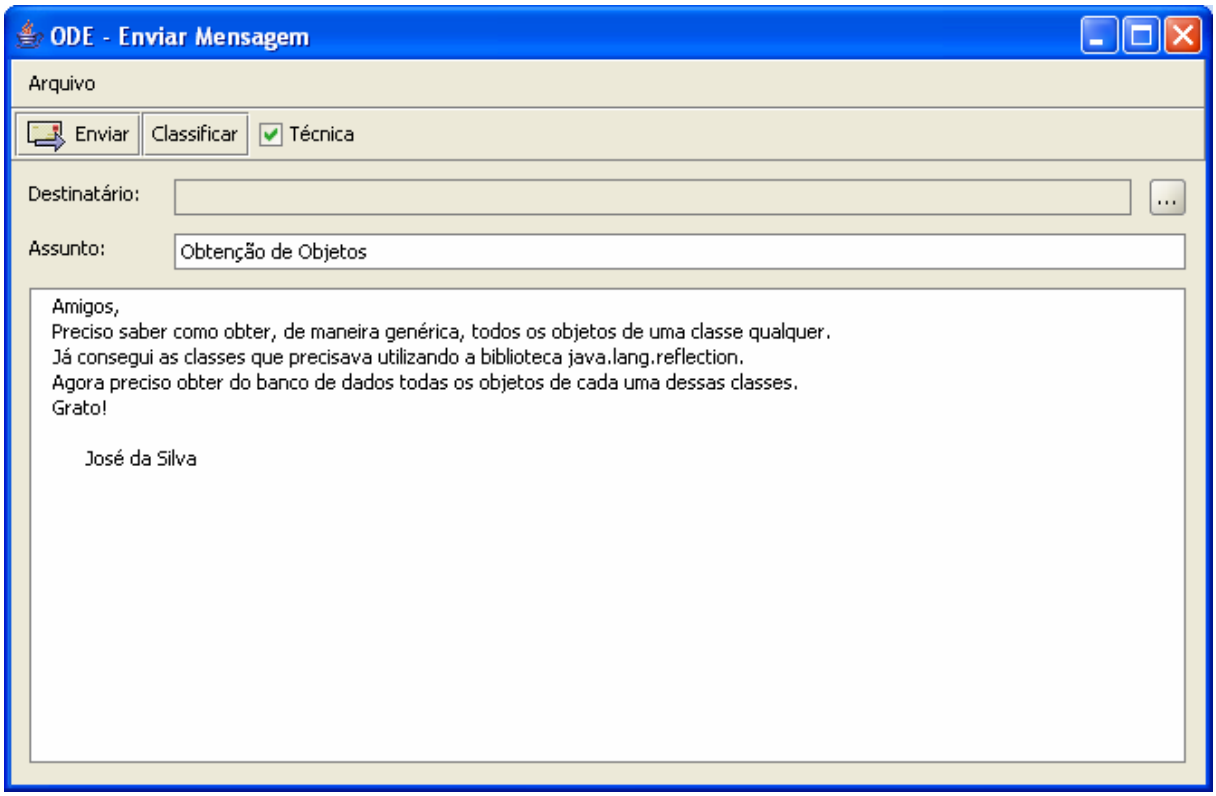


Figura 4.16 – Ferramenta de Correio Eletrônico de ODE (ARANTES, 2004)

Agindo da segunda forma, o ambiente é capaz de identificar quais pessoas acumulam as competências nas quais a mensagem foi classificada e enviá-la somente a estas pessoas.

O terceiro serviço é uma derivação do "Quem Sabe o Quê", fazendo uso da propriedade *nível* da relação *acúmulo*. Ele explora a tarefa de aprovação de lições aprendidas da infra-estrutura de gerência de conhecimento (NATALI, 2003), sendo utilizado para identificar quais pessoas acumulam determinadas competências a partir de um certo nível.

Uma das preocupações dessa infra-estrutura é a sobrecarga de aprovações para uma ou poucas pessoas, uma vez que a responsabilidade de aprovar é do Gerente de Conhecimento e qualquer pessoa com acesso ao ambiente está apta a registrar lições aprendidas. Assim, o serviço objetiva filtrar as solicitações de aprovação, da mesma maneira que faz com as mensagens no serviço anterior. Dessa forma, a responsabilidade de aprovação pode ser distribuída entre os especialistas que trabalham no ambiente. Classificando as lições aprendidas segundo competências, as suas aprovações são solicitadas às pessoas que acumulam as competências classificadas a partir de determinado nível.

A Figura 4.17 apresenta o serviço "Quem Sabe o Quanto".

```
/* Identifica quais pessoas acumulam a competência no nível passados. */
public List quemSabeOQuanto(KCompetencia competencia, int nivel) {
  return baseConhecimento.questionarModelo("acumulo", competencia,
nivel);
}
```

Figura 4.17 – Serviço "Quem Sabe o Quanto"

Outro serviço permite verificar se uma pessoa pode assumir um determinado papel na organização. Ele é realizado utilizando-se a o axioma (A2), referente à relação *contratação*, que diz que: *se uma pessoa acumula as competências demandadas por um recurso humano, então a pessoa pode ser contratada assumindo este papel*. O serviço é apresentado na Figura 4.18 e utiliza o método **`verificarModelo`**`(relacao, individuo, valor)` da classe `BaseConhecimento`.

```
/* Verifica se uma pessoa pode assumir um determinado papel. */
public boolean assumePapel(RecursoHumano pessoa, KRecursoHumano papel) {
  return baseConhecimento.verificarModelo("contratacao", pessoa, papel);
}
```

Figura 4.18 – Serviço "Assume Papel"

Por fim, um serviço que ainda não possui utilização no ambiente ODE foi testado para exemplificar um uso interessante em organizações de software: quais as pessoas que precisam ser capacitadas em uma determinada competência. Muitas vezes, é necessário possuir equipes qualificadas em algumas competências para atuar em determinados projetos. A identificação de quais pessoas estão aquém da capacidade necessária e precisam ser treinadas é um ponto de grande apoio em organizações de software.

Esse serviço proporcionaria alguns resultados similares aos do sistema de catálogo de competências apresentado em (DAVIES *et al.*, 2003), e utilizaria o serviço "Quem Sabe o Quanto", informando o nível de competência que, se acumulado, é necessário o treinamento.

É possível notar que os serviços criados, embora úteis em diversas tarefas, são estabelecidos de forma bastante simples. Este é um dos objetivos da Infra-estrutura Semântica: facilitar a utilização dos serviços semânticos pelos desenvolvedores de ferramentas de ODE, estendendo o seu uso pelo ambiente.

Para isso, a infra-estrutura provê uma interface simplificada. No entanto, ações mais elaboradas são possíveis, embora requeiram modificações na infra-estrutura que permitam esse tipo de acesso.

4.6 – Conclusões do Capítulo

O conhecimento tem sido considerado o patrimônio mais importante de uma organização, possuindo influência decisiva em sua competitividade (O'LEARY, 1998). No entanto, para a vantagem competitiva promovida pelo conhecimento ser sustentável, o conhecimento não pode estar no nível do indivíduo (DAVENPORT *et al.*, 1998). O conhecimento individual é perdido quando o indivíduo sai da organização. O conhecimento organizacional registrado em papel também representa um problema, pois pode não ser facilmente acessado, compartilhado e atualizado (O'LEARY, 1998).

Este capítulo tratou a incorporação de conhecimento organizacional em ODE. Com isso, gerência de conhecimento no ambiente passa tratar, além de vários tipos de conhecimento relacionados à Engenharia de Software, também o Conhecimento Organizacional. Esse novo conhecimento foi introduzido ao ambiente por meio da construção de uma Ontologia de Organizações de Software, que agora faz parte de sua base ontológica, e

da derivação desta ontologia em modelos de objetos que passam a compor o ambiente. A gerência do conhecimento organizacional em ODE aborda aspectos como estrutura organizacional, membros da organização e, principalmente, competências acumuladas por estes membros e requeridas para assumir papéis ou executar atividades.

Também foi abordada a proposta de uma Infra-estrutura Semântica, que objetiva ampliar as capacidades semânticas de ODE, utilizando as amarrações semânticas para instanciar bases de conhecimento ontológicas, representadas por uma linguagem de representação de ontologias (OWL) e manipuladas por uma biblioteca dotada de um motor de inferência (Jena). Também é intuito da infra-estrutura facilitar e disseminar o uso dos serviços semânticos pelos desenvolvedores do ambiente, tornando o tratamento da questão semântica uma forma de realmente ampliar o suporte oferecido pelas ferramentas de ODE.

Por fim, foi apresentada a experimentação realizada sobre a infra-estrutura semântica, criando serviços que apóiam tarefas do ambiente ODE atuando sobre o novo tipo de conhecimento incorporado, o conhecimento organizacional.

Capítulo 5

Considerações Finais

Este capítulo apresenta as considerações finais a respeito do trabalho desenvolvido. Na seção 5.1 são apresentadas as conclusões e principais contribuições do trabalho e na seção 5.2, são discutidas perspectivas futuras para dar continuidade ao trabalho de pesquisa.

5.1. Conclusões

Diante da crescente demanda de produtos de software, cada vez é mais importante para as organizações de software produzir com agilidade e qualidade. Crescente também é a complexidade dos sistemas nos dias atuais. E para atender esses requisitos do mercado é fundamental a utilização de ferramentas adequadas, processos de trabalho eficientes, pessoal especializado e meios de se aproveitar, ao máximo, o capital intelectual existente na organização.

Com o intuito de atender às necessidades de ferramentas, provendo apoio automatizado ao processo de software, surgem as ferramentas CASE e os Ambiente de Desenvolvimento de Software (ADSs). Um grande objetivo dos ADSs é tornarem-se ambientes completos, capazes de oferecer apoio não somente às atividades técnicas da Engenharia de Software, mas a uma amplitude muito maior das necessidades das organizações que produzem software.

Dessa maneira, muito se tem trabalhado para atender também às outras necessidades organizacionais. Os processos de trabalho são alvo de pesquisa da área de qualidade de software e diversas vezes integram-se aos ambientes de desenvolvimento de software (FUGGETTA, 2000), (BERTOLLO, 2006).

A utilização de pessoal especializado é normalmente trabalhada em conjunto com o aproveitamento de seu capital intelectual. É o que a pesquisa em ADSs tem feito na linha de

Gerência de Conhecimento (NATALI, 2003), (OLIVEIRA *et al.*, 2004), (LIMA, 2004), (HOLZ *et al.*, 2001). A gerência de conhecimento tem seu foco voltado para a solução de questões relacionadas a: como as organizações podem tirar maior proveito do conhecimento existente dentro delas, como membros da organização podem distribuir o conhecimento para quem este pode ser útil, como registrar as soluções adotadas para tratar problemas, como reter o conhecimento de seus especialistas mesmo quando estes deixam a organização, além de discutir formas de se gerar novo conhecimento a partir do conhecimento existente dentro da organização ou a partir de fontes externas (DAVENPORT *et al.*, 1998).

Um aspecto essencial é garantir que esse conhecimento esteja adequadamente organizado. Não basta acumular conhecimento se o seu acesso não for facilitado. As organizações têm problemas em identificar o conteúdo, localização e uso do conhecimento. Um melhor aproveitamento desse conhecimento é a motivação básica para a gerência de conhecimento na engenharia de software (RUS *et al.*, 2002). Ontologias são fundamentais neste ponto, sendo utilizadas para formalizar o conhecimento que se deseja gerenciar. Ontologias possuem um conjunto rico de relacionamentos, restrições e regras sobre as quais se pode raciocinar acerca da informação contida e, por conseguinte, raciocinar sobre itens que são classificados pela ontologia (MCCOMB, 2004). Elas permitem que os ADSs com gerência de conhecimento consigam administrar o conhecimento capturado de forma estruturada e forneçam apoio mais especializado aos usuários do ambiente.

O apoio automatizado tem sido foco de muitas investigações na história da Engenharia de Software. Entretanto, muito se evoluiu e com grande ênfase na captura de conhecimento, em como manipulá-lo, em como encontrá-lo e no que se entende por ele. Nesse contexto, a questão semântica vem sendo valorizada como possível solução para o complexo objetivo de fornecer apoio automatizado à prática de engenharia de software.

Este trabalho discutiu diversos dos aspectos acima mencionados, partindo desde as primeiras ferramentas de apoio automatizado e passando pelos ADSs e pesquisas relacionadas à gerência de conhecimento, ontologias e sistemas semânticos.

O levantamento do histórico dos ADSs, enfatizando seus principais tipos, características e tecnologias utilizadas, foi fundamental para detectar as principais necessidades de tratamento semântico dos recursos de informação presentes nos ADSs. O trabalho de (BROWN *et al.*, 1992), que no início desse histórico já apontava para um “nível semântico”, foi um marco importante para a elaboração de algumas discussões.

Também foi abordada a pesquisa com sistemas semânticos. Foram apresentadas as vantagens que a aplicação de semântica pode proporcionar aos sistemas, a necessidade por esse tipo de tratamento expressa por diversas características dos ADSs e os benefícios que a abordagem semântica tem provido às áreas relacionadas, como *Web* Semântica e Gerência de Conhecimento. E, partindo dessas discussões, foi apresentado o objetivo de tratar explicitamente semântica em ADSs, evoluindo-os para ADSs Semânticos.

A questão semântica deve ser tratada em detalhes nos ambientes, estando atrelada à sua estrutura. Como um dos objetivos é fazer com que cada item tenha seu significado compreendido, a semântica deve ser uma característica inerente à modelagem do sistema. Este foi o trabalho realizado em ODE, com o objetivo de evolução. A arquitetura conceitual do ambiente foi remodelada para três níveis, admitindo uma presença mais forte das ontologias, e permitindo que cada objeto e classe do ambiente esteja semanticamente amarrado a sua origem ontológica. Além disso, a nova arquitetura abriu espaço para a reestruturação da infra-estrutura de gerência de conhecimento e possibilitou o uso mais amplo das características semânticas no ambiente.

O trabalho discutiu, ainda, a incorporação de conhecimento organizacional em ODE e apresentou a Ontologia de Organizações de Software definida e como se deu o seu processo de derivação em modelos dos outros dois níveis arquiteturais. Finalizando, foi apresentada a proposta de uma infra-estrutura de serviços semânticos que, para facilitar e ampliar a realização de ações semânticas no ambiente, utiliza as ontologias e o seu conhecimento derivado em três passos: importação da ontologia, criação de uma base de conhecimento ontológico com indivíduos recuperados a partir dos objetos do ambiente, e a realização de deduções lógicas suportada por um motor de inferências baseado em OWL. Como forma de experimentação e avaliação da infra-estrutura proposta, foram criados alguns serviços para agir semanticamente sobre o conhecimento organizacional incorporado ao ambiente.

Diante dos pontos discutidos, é possível apontar as principais contribuições deste trabalho:

- Remodelagem da arquitetura em níveis de ODE, incluindo o nível ontológico e fortalecendo as amarrações com meta-dados baseados em ontologias;
- Revisão da infra-estrutura de Gerência de Conhecimento, intensificando o uso de anotações ontológicas nos serviços de busca;

- Definição de uma Ontologia de Organizações de Software, a partir de outras ontologias de organização publicadas na literatura (LIMA, 2004), (COTA, 2002), (LEÃO *et al.*, 2004), com foco em competências, integrada às demais ontologias de ODE;
- Definição da Infra-Estrutura Semântica proposta para ampliar e facilitar o uso de ações semânticas no ambiente;
- Experimentação de uma linguagem moderna para representação de modelos ontológicos (OWL) e de bibliotecas capazes de explorar a semântica desses modelos utilizando capacidades de inferência (Jena).

São visíveis ao ambiente as vantagens da nova arquitetura conceitual, capaz de associar os objetos e classes do ambiente à sua origem ontológica. A introdução de conhecimento organizacional atende a uma das expectativas de evolução do ambiente ODE. E a Infra-estrutura Semântica abre espaço para que mais tarefas do ambiente sejam apoiadas por conhecimento ontológico. Contudo, sabe-se que a infra-estrutura semântica é apenas uma proposta e representa um primeiro passo na linha de trabalhos dessa natureza. Assim, a próxima seção apresenta esta e outras perspectivas de evolução deste trabalho.

5.2 – Perspectivas Futuras

Buscando-se melhorar e expandir a abordagem semântica proposta, algumas perspectivas de trabalhos futuros podem ser destacadas. Algumas delas representam apenas melhorias funcionais e correção de pontos falhos do trabalho. Outras devem ser trabalhadas em um âmbito maior e representam evoluções deste trabalho.

Inicialmente, cabe listar alguns pontos negativos do trabalho. A maioria deles referente à infra-estrutura semântica, que ainda tem muito a evoluir, e à arquitetura conceitual, que, após as experimentações, pôde evidenciar algumas de suas falhas.

O primeiro item é quanto ao esquema de amarrações entre ontologias e modelos de objetos do ambiente. Atualmente, existem amarrações somente entre conceitos e suas derivações, as classes. Essas ligações são suficientes para a maioria das ações exploradas nesse contexto. Entretanto, para permitir a rastreabilidade completa entre ontologias e modelos, é necessário armazenar também as ligações entre os outros elementos de modelo (propriedades e atributos, e relações e associações) e as decisões tomadas durante o processo de derivação.

Aliado à completude de rastreamento, torna-se extremamente útil prover um módulo interativo para importação e amarração de ontologias, dadas as inúmeras decisões que podem ser tomadas durante o processo de derivação de ontologias em modelos de objetos. Na verdade, o processo de amarração compartilharia boa parte de suas atividades com um processo de derivação semi-automática dos modelos. Assim, poder-se-ia tratar a incorporação, seguida de derivação e amarração em conjunto, produzindo, além dos objetos do nível ontológico, suas amarrações e as próprias classes dos modelos de conhecimento e controle. O trabalho de Mian (2003) realizou alguns experimentos de derivação semi-automática de ontologias, mas com as considerações feitas neste trabalho, as pesquisas realizadas precisam evoluir.

Outro ponto a ser melhorado é a forma como os serviços semânticos são criados na infra-estrutura semântica. Atualmente, um serviço é, basicamente, um método criado que invoca a infra-estrutura, questionando sobre uma relação e um valor associado. Dessa forma, os serviços são, de certo modo, um pouco inflexíveis, além de acessar diretamente o nome das relações desejadas. Uma sugestão é possibilitar o cadastramento de serviços, permitindo uma maior facilidade de criação, manutenção e utilização dos serviços semânticos. A idéia é seguir um pouco os moldes do que já havia sido feito nas experiências com serviços semânticos usando Prolog (RUY, 2003).

É válido também tornar as bases de conhecimento ontológico (modelos OWL instanciados) mais dinâmicas, permitindo que, quando os objetos do ambiente forem alterados, sejam parcialmente re-instanciadas antes que um questionamento seja respondido. Essa modificação permitiria uma capacidade de aprendizado em tempo real das bases de conhecimento com o próprio ambiente.

Ponto também importante é aprofundar a utilização da linguagem OWL e da API Jena, que possuem muito mais funcionalidades do que as utilizadas neste trabalho. O objetivo é fazer uso mais especializado das capacidades de inferência permitidas.

Antes de partir para as perspectivas mais gerais, convém mencionar que, em meio à realização deste trabalho, o ambiente ODE sofreu uma reestruturação quanto ao seu modelo de persistência, passando a utilizar o *framework Hibernate*. Dessa forma, como muitas das funcionalidades ligadas a este trabalho, tais como as de gerência de conhecimento, não haviam sido migradas para a nova versão do ambiente, foi tomada a decisão de criar a infra-estrutura semântica na versão antiga do ambiente. Apesar de ser um módulo de baixo acoplamento, a sua migração para a nova versão é um trabalho a ser realizado, assim que as funcionalidades necessárias também o forem.

Em um âmbito mais geral do ambiente, vale também destacar algumas das perspectivas deixadas por este trabalho. A primeira delas relaciona-se a ODEd, o editor de ontologias de ODE. O trabalho aqui desenvolvido seguiu com independência desta ferramenta. Entretanto, é sabido que uma integração entre as partes pode trazer muitos benefícios, como a possibilidade de editar as ontologias diretamente em ODEd, em vez de utilizar o Protégé, gerando o modelo OWL necessário. Além disso, a edição da ontologia e os processos de derivação de modelos de objetos e de amarração entre os níveis podem ser realizados em uma única ferramenta.

Quanto à Ontologia de Organizações de Software, seria de grande relevância uma maior exploração de questões relacionadas à Cultura Organizacional (MARTIN, 2002), (DAVIES *et al.*, 2003), envolvendo conceitos como estrutura hierárquica e horizontal, regras, clima, valores, liderança, comunicação, aprendizado etc. Dessa forma, a elaboração de novas questões de competência e posterior evolução da ontologia poderia torná-la mais completa em um âmbito organizacional. Outras vertentes de expansão da ontologia estão relacionadas ao aprofundamento dos conceitos existentes, relacionados à área de Organização de Agentes (*Agent Organization*) (TAKEDA *et al.*, 1995) e Aprendizado Organizacional (DIERKES *et al.*, 2001).

Outras evoluções do trabalho são a utilização mais extensa de semântica nos outros serviços de gerência de conhecimento, tais como a captura, reúso e disseminação de conhecimento; e o uso de ações semânticas associadas a agentes (FALBO *et al.*, 2005b), potencializando o apoio inteligente e pró-ativo aos usuários.

Portanto, muito trabalho ainda deve ser realizado. Os resultados e contribuições apresentados nesta dissertação representam um primeiro passo na evolução de ODE em direção a um ADS Semântico, com as capacidades desejadas e que esteja apto a fornecer a seus usuários e às organizações em que atue apoio mais efetivo em suas atividades.

Acredita-se que as contribuições realizadas neste trabalho possam intensificar o uso de semântica no ambiente e abram caminho para que sejam desenvolvidas ferramentas mais especializadas e capazes de atender aos objetivos pretendidos pela Engenharia de Software.

Referências Bibliográficas

(AMBRIOLA, *et al.*, 1997) AMBRIOLA, V., CONRADI, R., FUGGETTA, A., "*Assessing Process-Centered Software Engineering Environments*". ACM Transactions on Software Engineering and Methodology, v. 6, n.3, 283-328, July, 1997.

(ARANTES, 2004) ARANTES, D.O., "Integrando Ferramentas de Comunicação com Gerência de Conhecimento em ODE". Projeto de Graduação, Curso de Ciência da Computação, UFES, Abril 2004.

(ARBAOUI *et al.*, 2002) ARBAOUI, S., DERNIAME, J., OQUENDO, F., *et al.*, "*A Comparative Review of Process-Centered Software Engineering Environments*". Annals of Software Engineering, v. 14, 311-340, Netherlands, 2002.

(BENJAMINS *et al.*, 1998) BENJAMINS, V.R., FENSEL, D., PÉREZ, A.G., "*Knowledge Management through Ontologies*". Proc. of the 2nd International Conference on Practical Aspects of Knowledge Management (PAKM98), Switzerland, 1998.

(BERTOLLO *et al.*, 2002) BERTOLLO, G., RUY, F.B., MIAN, P.G., PEZZIN, J., SCHWAMBACH, M., NATALI, A.C.C., FALBO, R.A., "*ODE – Um Ambiente de Desenvolvimento de Software Baseado em Ontologias*". Anais do XVI Simpósio Brasileiro de Engenharia de Software - Caderno de Ferramentas, Gramado, Outubro de 2002.

(BERTOLLO *et al.*, 2006) BERTOLLO, G., FALBO R. A., "*Definição de Processos em um Ambiente de Desenvolvimento de Software Baseado em Ontologias*". Anais do V Simpósio Brasileiro de Qualidade de Software, p. 72-86, Vila Velha, Brasil, Maio 2006.

(BERTOLLO, 2006) BERTOLLO, G., "*Definição de Processos em um Ambiente de Desenvolvimento de Software*". Dissertação de Mestrado, Mestrado em Informática, UFES, Vitória, Junho 2006.

(BRICKLEY *et al.*, 1999) BRICKLEY, D., GUHA, R.V., "*Resource Description Framework (RDF) Schema Specification. Technical Report, W3C*". W3C Proposed Recommendation. http://www.w3.org/TR/PR-rdf-schema. 1999.

(BROMMÉ *et al.*, 1999) BROMMÉ, M., RUNESON, P., "*Technical Requirements for the Implementation of an Experience Base*". Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, 1999.

(BROWN *et al.*, 1992) BROWN, A.W., McDERMID, J.A., "*Learning from IPSE's Mistakes*". IEEE, march, 1992.

(CARVALHO, 2002) CARVALHO, A.L., "*SIDER: Um Ambiente de Desenvolvimento de Software Orientado ao Domínio Siderúrgico*". Dissertação de Mestrado, UFES, Vitória, Brasil, Julho de 2002.

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

(CHANDRASEKARAN *et al.*, 1999) CHANDRASEKARAN, B., JOSEPHSON, J.R., BENJAMINS, V.R., "*What Are Ontologies and Why Do We Need Them?*". IEEE Intelligent Systems, pp. 20-26, Jan. 1999.

(CHEN *et al.*, 1992) CHEN, M., NORMAN, R.J., "*A Framework for Integrated CASE*". IEEE Software, March, 1992.

(CHRISTIE, 1995) CHRISTIE, A. M., "S*oftware Process Automation: The Technology and its Adoption*". Peittsburghm Pennsylvannia, Springer-Verlag Berlin Heidelberg, 1995.

(CONNOLLY *et al.*, 2001) 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.

(COTA, 2002) COTA, R., "*Modelagem Computacional para Gestão Empresarial*". Dissertação de Mestrado, UFES, Vitória, Brasil, 2002.

(DAL MORO *et al.*, 2005) DAL MORO, R., NARDI, J.C., FALBO, R.A., "*ControlPro: Uma Ferramenta de Acompanhamento de Projetos Integrada a um Ambiente de Desenvolvimento de Software*". XII Sessão de Ferramentas do Simpósio Brasileiro de Engenharia de Software, SBES'2005, Uberlândia, Brasil, Outubro 2005.

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

(DAVIES *et al.*, 2003) DAVIES, J., FENSEL, D., VAN HARMELEN, F., "*Towards the Semantic Web: Ontology-driven Knowledge Management*". John Wiley & Sons, 2003.

(DIERKES *et al.*, 2001) DIERKES, M., ANTAL, A.B., CHILD, J., NONAKA, I., "*Handbook of Organizational Leaning and Knowledge*". New York: Oxford University Press, 2001.

(DUARTE, 2001) DUARTE, K.C., "*Um Framework de Reuso no Domínio de Qualidade de Software*". Dissertação de Mestrado, UFES, Vitória, Junho 2001.

(FALBO, 1998) FALBO, R.A., "I*ntegração de Conhecimento em um Ambiente de Desenvolvimento de Software*". Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil, 1998.

(FALBO *et al.*, 1998) FALBO, R.A., MENESES, C.S., ROCHA, A.R.C., "*A Systematic Approach for Building Ontologies*". Proceedings of the 6th Ibero-American Conference on Artificial Intelligence,

Lecture Notes in Computer Science, vol. 1484, Lisbon, Portugal, 1998.

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

(FALBO *et al.*, 2002b) FALBO, R.A., GUIZZARDI, G., NATALI, A.C.C., BERTOLLO, G., RUY, F.B., MIAN, P.G., "*Towards Semantic Software Engineering Environments*". Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE'2002, Ischia, Italy, 2002.

(FALBO *et al.*, 2003) 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, 1124-1135, La Plata, Argentina, Outubro 2003.

(FALBO, 2004) FALBO, R.A., "*Experiences in Using a Method for Building Domain Ontologies*". Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering, SEKE'2004, pp. 474-477, International Workshop on Ontology In Action, OIA'2004, Banff, Alberta, Canada, June 2004.

(FALBO *et al.*, 2004a) FALBO, R.A., ARANTES, D.O., NATALI, A.C.C., "*Integrating Knowledge Management and Groupware in a Software Development Environment*". 5th International Conference on Practical Aspects of Knowledge Management, Vienna, Austria, 2004.

(FALBO *et al.*, 2004b) FALBO, R.A., RUY, F.B., BERTOLLO, G., TOGNERI, D.F., "*Learning How to Manage Risks Using Organizational Knowledge*". Proceedings of the 6th International Workshop on

Advances in Learning Software Organizations, LSO'2004, pp. 7-18, Banff, Canada, June 2004.

(FALBO *et al.*, 2004c) FALBO, R.A., RUY, F.B., PEZZIN, J., DAL MORO, R., "*Ontologias e Ambientes de Desenvolvimento de Software Semânticos*". 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, JIISIC'2004, Vol. I, 277-292, Madrid, Spain, November 2004.

(FALBO *et al.*, 2005a) FALBO, R.A., PEZZIN, J., SCHWAMBACH, M.M., "*A Multi-Agent System for Knowledge Delivery in a Software Engineering Environment*". 17th International Conference on Software Engineering and Knowledge Engineering, SEKE'2005, p. 253 - 258, Taipei, China, July 2005.

(FALBO *et al.*, 2005b) FALBO, R.A., RUY, F.B., DAL MORO, R., "*Using Ontologies to Add Semantics to Software Engineering Environments*". 17th International Conference on Software Engineering and Knowledge Engineering, SEKE'2005, 151-156, Taipei, China, July 2005.

(FININ *et al.*, 1995) FININ, T., LABROU, Y., MAYFIELD, J., "*KQML as an agent communication language*". September, 1995. Disponível em http://www.cs.umbc.edu/~finin/papers/.

(FISCHER *et al.*, 2001) FISCHER, G., OSTWALD, J., "Knowledge *Management: Problems, Promises, and Challenges*". *IEEE Intelligent Systems*, v. 16, n. 1 (Jan/Fev), pp. 60-72 2001.

(FISCHER, 1996) FISCHER, G., "*Seeding, Evolutionary and Reseeding: capturing and evolving knowledge in domain-oriented design environments*". In: Sutcliffe, A., Benyon, B. Van Assche (eds) - IFIP 8. 1/13. Joint Working Conference – Domain-Knowledge for Interactive System Design, pp 1-16, Geneva, Switzerland, May, 1996.

(FLEURY *et al.*, 2000) FLEURY, A., FLEURY, M.T.L., "*Estratégias Empresariais e Formação de Competências – um Quebra-cabeças*

Caleidoscópico da Indústria Brasileira". Atlas. São Paulo, 2000.

(FOURO, 2002) FOURO, A.M., "*Apoio à Construção de Base de Dados de Pesquisa em Ambientes de Desenvolvimento de Software Orientados a Domínio*". Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil, 2002.

(FOX *et al.*, 1996) FOX, M.S., BARBUCEANU, M., GRUNINGER, M., "*An Organization Ontology for Enterprise Modelling: Preliminary Concepts for Linking Structure and Behaviour*". Computers in Industry, v. 29, pp. 123-134, 1996.

(FUGGETTA, 2000) FUGGETTA, A., "*Software Process: a Roadmap*". In: Proceedings of the Conference on the Future of Software Engineering - International Conference on Software Engineering, 25-34, Limerick, Ireland, 2000.

(GALLOTA, 2000) GALLOTA, C., "*Netuno: Um Ambiente de Desenvolvimento de Software Orientado ao Domínio de Acústica Submarina*". Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil, 2000.

(GARVIN, 2000) GARVIN, D.A., "*Learning in action: a guide to putting the learning organization to work*". USA: Harvard Business School Press, 2000.

(GRUHN, 2002) GRUHN, V., "*Process-Centered Software Engineering Environments – A Brief History and Future Challenges*". In: Annals of Software Engineering v. 14, 363-382, Netherlands, 2002.

(GUARINO, 1998) GUARINO, N., "*Formal Ontology and Information Systems*". In: Formal Ontologies in Information Systems, N. Guarino (Ed.), IOS Press, 1998.

(HARRISON *et al.*, 2000) HARRISON, W., OSSHER, H., TARR, P., "*Software Engineering Tools and Environments: A Roadmap*". In:

Proceedings of the Conference on the Future of Software Engineering - International Conference on Software Engineering, 261-277, Limerick, Ireland, 2000.

(HOLANDA, 1999) HOLANDA, A.B., "*Novo Aurélio Século XXI - O Dicionário da Língua Portuguesa*". Nova Fronteira,1999.

(HOLZ *et al.*, 2001) HOLZ, H., KÖNNECKER, A., MAURER, F., "*Task-Specific Knowledge Management in a Process-Centred SEE*". In: Advances in Learning Software Organizations, v. 2176, Lecture Notes in Computer Science, Springer, pp. 163-177, 2001.

(JURISTO *et al.*, 2002) JURISTO, N., MORENO, A.M., "*Reliable Knowledge for Software Development*". IEEE Software, v. 19, n. 5, pp. 98-99, October 2002.

(KUCZA, *et al.*, 2001) KUCZA, T., NÄTTINEN, M., PARVIAINEN, P., "*Improving Knowledge Management in Software Reuse Process*". Proceedings of 3th International Conference on Product Focused Software Process Improvement, v. 2188, Lecture Notes in Computer Science, Springer-Verlag, pp. 141-152, 2001.

(LEÃO *et al.*, 2004) LEÃO, P.R.C., OLIVEIRA, K.M., MORESI, E.A.D., "*Ontologia de Competências Profissionais em Tecnologia da Informação"*. Simpósio Brasileiro de Qualidade de Software, Workshop de Gerência de Conhecimento, Brasília, 2004.

(LIMA, *et al.*, 2000) LIMA, K.V.C., ROCHA, A.R.C., TRAVASSOS, G.H., "*Ambientes de Desenvolvimento de Software Orientados à Organização"*. COPPE/UFRJ. Rio de Janeiro, RJ. Abril de 2000.

(LIMA *et al.*, 2002) LIMA, K.V.C., SANTOS, G., TRAVASSOS, G., Rocha, A.R., "*Melhoria de Processos de Software e Evolução de Ambientes de Desenvolvimento de Software com base no Conhecimento do Domínio e na Cultura Organizacional"*. Simpósio Brasileiro de Qualidade de Software, Gramado, Brasil, 2002.

(LIMA, 2004) LIMA, K.V.C., "*Definição e Contrução de Ambientes de Desenvolvimento de Software Orientados a Organização*". Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, 2004.

(MARKKULA, 1999) MARKKULA, M., "*Knowledge Management in Software Engineering Projects*". Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, June, 1999.

(MARTIN, 2002) MARTIN, J.L., "*Organizational Culture: Mapping the Terrain*". London: Sage Publications, 2002.

(MASLOW, 2000) MASLOW, A.H., "*Maslow no Gerenciamento*". Qualitymark. Rio de Janeiro, 2000.

(MAURER *et al.*, 2002) MAURER, F., HOLZ, H., "*Integrating Process Support and Knowledge Management for Virtual Software Development Teams*". Annals of Software Engineering, v. 14, n. 1-4, pp. 145-168, 2002.

(MCCOMB, 2004) MCCOMB, D., "*Semantic in Business Systems: The Savvy Manager's Guide*". Morgan Kaufmann Publishers, 2004.

(MEEHAN *et al.*, 2002) MEEHAN, B., RICHARDSON, I., "*Identification of Software Process Knowledge Management*". Software Process Improvement and Practice, v. 7, n. 2, pp. 47-55, June 2002.

(MEGGINSON *et al.*, 1986) MEGGINSON, L., MOSLEY, D., PIETRI, P., "*Fundamentos do Modelo Organizacional*". Administração: Conceitos e Aplicações, capítulo 8, Editora Harbra, 1986.

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

(MIAN *et al.*, 2003) MIAN, P.G., FALBO, R.A., "*Supporting Ontology Development with ODEd*". Journal of the Brazilian Computer Science, vol. 9, nº. 2, 57-76, November, 2003.

(NARDI *et al.*, 2006) NARDI, J. C.; FALBO, R.A., "*Uma Ontologia de Requisitos de Software*". IX Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software, La Plata, Argentina, Abril 2006.

(NATALI, 2003) NATALI, A.C.C., "*Uma Infra-estrutura para Gerência de Conhecimento em um Ambiente de Desenvolvimento de Software*". Dissertação de Mestrado, Mestrado em Informática, UFES, Vitória, 2003.

(NATALI *et al.*, 2003) NATALI, A.C.C., FALBO, R.A., "*Gerência de Conhecimento em ODE*". Anais do XVII Simpósio Brasileiro de Engenharia de Software, 270-285, Manaus, Brasil, Outubro, 2003.

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

(NUNES, 2005) NUNES, B.V., "*Integrando Gerência de Configuração de Software, Documentação e Gerência de Conhecimento em um Ambiente de Desenvolvimento de Software*". Dissertação de Mestrado, UFES, Vitória, Brasil, 2005.

(O'LEARY, 1998) O'LEARY, D.E., "*Enterprise Knowledge Management*". IEEE Computer Magazine, March, 1998.

(OLIVEIRA, 1999) OLIVEIRA, K.M., "*Modelo para Construção de Ambientes de Desenvolvimento Orientados a Domínio*" Tese de DSc., COPPE/UFRJ, Rio de Janeiro, Brasil, Out 1999.

(OLIVEIRA *et al.*, 2000) OLIVEIRA, K.M., TRAVASSOS, G.H., MENEZES, C.S., ROCHA, A.R.C., "*Ambientes de Desenvolvimento de Software Orientados a Domínio*". Anais do XIV Simpósio Brasileiro de Engenharia de Software, 2000.

(OLIVEIRA *et al.*, 2004) OLIVEIRA, K.M., ZLOT, F., ROCHA, A.R.C., TRAVASSOS, G.H., GALOTTA, C., MENESES, C.S., "*Domain-oriented Software Development Environment*". The Journal of Systems and Software 72, 145-161, 2004.

(PEZZIN, 2004) PEZZIN, J., "*AgeODE: Uma Infra-estrutura para Apoiar a Construção de Agentes para Atuarem em um Ambiente de Desenvolvimento de Software*". Dissertação de Mestrado, Mestrado em Informática, UFES, Vitória, Outubro 2004.

(PEZZIN *et al.*, 2004) PEZZIN, J., FALBO, R.A., "*AgeODE: Uma Infra-estrutura para Apoiar a Construção de Agentes para Atuarem no Ambiente de Desenvolvimento de Software ODE*". 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, JIISIC'2004, Madrid, Spain, November 2004.

(PFLEEGER, 1999) PFLEEGER, S., "*Albert Einstein and Empirical Software Engineering*". IEEE Computer, v. 32, n. 10, pp. 32-38, Oct. 1999.

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

(PRAHALAD *et al.*, 1990) PRAHALAD, C.K., HAMEL, G., "*The Core Competence of the Corporation*". Harvard Business Review, v. 38, n. 3, pp. 79-91, May/Jun. 1990.

(PRESSMAN, 2004) PRESSMAN, R.S., "*Software Engineering: A Practitioner's Approach*". 6th Edition, New York: McGraw-Hill, 2004.

(PROTÉGÉ, 2006) PROTÉGÉ. "*Protégé – Ontology Editor and Knowledge Acquisition System*". http://protege.stanford.edu, May 2006.

(RASMUS, 1995) RASMUS, D.W., "*Ruling Classes: The Heart of Knowledge-based Systems*". Object Magazine, July 1995.

(ROCHA *et al.*, 1990) ROCHA, A.R.C., AGUIAR, T.C., SOUZA, J.M., 1990, "*Taba: A Heuristic Workstation for Software development*".

Proceedings of COMPEURO 90, Tel Aviv, Israel, Maio de 1990.

(RUS *et al.*, 2002) RUS, I., LINDVALL, M., "*Knowledge Management in Software Engineering*". IEEE Software, v.19, n. 3, pp. 26-38, May/Jun. 2002.

(RUY, 2003) 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.

(RUY *et al.*, 2003) RUY, F.B., BERTOLLO, G., FALBO, R.A. "*Uma Ferramenta Baseada em Conhecimento para Apoiar a Definição de Processos de Software em Níveis*". Anais da X Sessão de Ferramentas do Simpósio Brasileiro de Engenharia de Software - SBES'2003, pp.1-6, Manaus - Amazonas, Outubro 2003.

(RUY *et al.*, 2004) RUY, F.B., BERTOLLO, G., FALBO, R.A., "*Knowledge-based Support to Process Integration in ODE*". CLEI Electronic Journal, Volume 7, Number 1, June 2004.

(SCHWAMBACH, 2004) SCHWAMBACH, M.M., "*OplA: Uma Metodologia para Desenvolvimento de Sistemas Orientados a Objetos e Agentes*". Dissertação de Mestrado, Vitória, Brasil, 2004.

(SOUZA *et al.*, 2003) SOUZA. V.E.S., FALBO. R.A., "*Construindo Axiomas e Avaliando Ontologias em ODEd*". Anais da X Sessão de Ferramentas do Simpósio Brasileiro de Engenharia de Software, 7-12, Manaus, Brasil, Outubro, 2003.

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

(STAAB *et al.*, 2001) STAAB, S., STUDER, R., SCHNURR, H.P., SURE, Y., "*Knowledge Processes and Ontologies*". IEEE Intelligent Systems, vol. 16, No. 1, January/February, 2001.

(TAKEDA *et al.*, 1995) TAKEDA, H., LINO, K., ISHIDA, T., "*Agent Organization and Communication with Multiple Ontologies*". International Journal of Cooperative Information Systems – IJCIS, No. 4, pp. 321-337, 1995.

(THOMAS *et al.*, 1992) THOMAS, I., NEJMEH, B.A., "*Definitions of Tool Integration for Environments"*. IEEE Software, March, 1992.

(TIWANA, 2000) TIWANA, A., "*The Knowledge Management Toolkit: Practical Techniques for Building a Knowledge Management System*". New York: Prentice Hall PTR, 2000.

(TRAVASSOS, 1994) TRAVASSOS, G.H., "*O Modelo de Integração de Ferramentas da Estação TABA*". Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, 1994.

(USCHOLD *et al.*, 1998) USCHOLD, M., KING, M., MORALEE, S., *et al.*, "*The Enterprise Ontology*". The Knowledge Engineering Review, v.13, n. 1, Special Issue on Putting Ontologies to Use (eds. Mike Uschold e Austin Tate), 1998.

(WOOLDRIDGE *et al.*, 2000) WOOLDRIDGE, M., JENNINGS, N. R., KINNY, D., "*The Gaia Methodology for Agent-Oriented Analysis and Design*". Journal of Autonomous Agents and Multi-Agent Systems, 2000.

Livros Grátis

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

Milhares de Livros para Download:

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