MODELO PARA CONSTRUÇÃO DE AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE ORIENTADOS A DOMÍNIO

Káthia Marçal de Oliveira

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profa. Ana Regina Cavalcanti da Rocha, D. Sc.

Prof. Guilherme Horta Travassos, D. Sc.

Prof. Álvaro Rabelo Alves Junior, L.D

Prof. André Monat, D.Sc.

Prof. Crediné Silva de Menezes, D.Sc.

Prof. Walcélio Melo, PhD.

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 1999

OLIVEIRA, KATHIA MARCAL DE

Modelo para Construção de Ambiente de Desenvolvimento de Software Orientado a Domínio [Rio de Janeiro] 1999

IX, 222 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas, 1999)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Ambiente de Desenvolvimento de Software
2. Orientação a Domínio
3. Ontologia

I. COPPE/UFRJ II. Título (série)

Aos meus verdadeiros amigos
que contribuíram direta
ou indiretamente com
este trabalho.

Agradecimentos

À professora Ana Regina, que, mais que orientadora e amiga, é a grande responsável pela minha formação, guiando meus passos e me fazendo crer que todo desafio é importante para o crescimento profissional e pessoal.

Ao professor Álvaro Rabelo, por proporcionar um ambiente rico para experimentação das minhas idéias, estimular o meu trabalho e por ser exigente e crítico para que eu sempre obtivesse o melhor resultado possível.

Ao professor Guilherme Travassos, que mesmo à distância foi um orientador amigo e presente.

Ao professor Crediné, pelas discussões e idéias para este trabalho, e ao amigo, que me apoiou em vários momentos, ensinando-me que a vida, como a pesquisa, é sempre uma grande descoberta.

Aos professores Walcélio Melo e André Monat, por participarem da minha banca, contribuindo com comentários significativos para o enriquecimento do meu trabalho.

Ao Dr. Ximenes, cuja colaboração foi indispensável para a realização deste trabalho.

Ao professor Stan Matwin, pela acolhida na Universidade de Ottawa e pelas inúmeras sugestões a este trabalho.

A minha mãe por sofrer, torcer e viver cada minuto necessário para realização deste trabalho, sempre me trazendo força e fé.

Ao meu pai por entender e, acima de tudo, incentivar tudo que eu desejo realizar, mostrando-me constantemente o quanto me ama.

Aos meus irmãos: Renival, o eterno cúmplice dos meus sonhos, sempre me incentivando a seguir em frente; Júnior, pelo contínuo apoio ao caminho da pesquisa e da especialização; e Cacinha, que, mesmo sem entender minha ausência, soube incentivar e torcer pelo meu sucesso.

A minha prima, Aninha, por ser mais que uma amiga, ser uma irmã.

A Aninha, Karina e Cláudia (G), sem as quais não poderia concluir esse trabalho, dividindo não só as conquistas, mas os momentos mais difíceis de realização desta tese.

A meu irmão ontológico Ricardo Falbo, pelo prazer de descobrir juntos as maravilhas do conhecimento e pelo apoio amigo durante todo o meu trabalho.

A equipe orientada a domínio (Alessandro, Gleison, Fabio, Gustavo, Cátia e Luis Felipe), que transformaram este trabalho em mais que um projeto pessoal, mas em um projeto de equipe, cujo objetivo principal é o crescimento de cada um.

A Vera, pela amizade e carinho durante este trabalho.

Ao pessoal da FBC e da Rio-Sul, pelo apoio e, principalmente, pela amizade.

Aos meus amigos da COPPE e da Universidade de Ottawa.

Aos professores e funcionários do Programa.

A Ana Paula, por sua presença sempre simpática e disponível para todos os alunos.

Ao CNPQ, pelo apoio financeiro.

E, finalmente, a Nícolas, pela prova de amor que me deu durante os últimos anos deste trabalho, entendendo minhas ausências, suportando minhas angústias e me ajudando em todos os aspectos emocionais e de trabalho, contribuindo diretamente para o sucesso alcançado.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

MODELO PARA CONSTRUÇÃO DE AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE ORIENTADOS A DOMÍNIO

Káthia Marçal de Oliveira

Outubro/1999

Orientadores: Ana Regina Cavalcanti da Rocha
Guilherme Horta Travassos

Programa: Engenharia de Sistemas e Computação

Durante o desenvolvimento de software, os desenvolvedores tem que lidar com diferentes atividades não triviais. A mais críticas dessa atividades é provavelmente a identificação correta dos requisitos do sistema e sua descrição. Esta atividade é ainda mais difícil quando os desenvolvedores não conhecem o domínio ou não tem nenhuma experiência em desenvolver software para aquele domínio. Nós defendemos que o uso do conhecimento do domínio durante o desenvolvimento de software pode tornar esse processo mais fácil e melhorar a produtividade.

Para apoiar essa idéia nós definimos Ambientes de Desenvolvimento de Software Orientado a Domínio (ADSOD). Estes ambientes tornam disponível o conhecimento sobre o domínio numa representação simbólica utilizando ontologias do domínio e a identificação de possíveis tarefas realizadas no domínio em questão. Ferramentas específicas do domínio introduzidas no ambiente definem o uso desse conhecimento em um processo de software bem definido.

Esta tese apresenta um modelo para construção de ADSOD baseado nessas características. É apresentado também um ADSOD para o Domínio de Cardiologia construído usando este modelo.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A FRAMEWPRK FOR THE CONSTRUCTION OF DOMAIN-ORIENTED SOFTWARE DEVELOPMENT ENVIRONMENTS

Káthia Marçal de Oliveira

October/1999

Advisors: Ana Regina Cavalcanti da Rocha
Guilherme Horta Travassos

Department: Computer Science and System Engineering

Throughout software development, a software team has to deal with several non-trivial activities. The most critical is probably the correct identification and description of what the software system must accomplish (requirements). This is particularly hard when the software team does not have enough knowledge about the problem domain and no expertise developing software for that domain. We argue that the use of domain knowledge during the software development can render this process easier and increase the productivity.

To reinforce and support this assumption we defined Domain-Oriented Software Development Environment (DOSDE). This environment makes available domain knowledge in a symbolic representation by using domain ontology and an identification of potential tasks related to the domain. Domain-specific tools introduced into the software development environment define the use of this knowledge during a well-defined software process. This thesis presents a framework to construct a DOSDE that follows these features. It also presents a DOSDE for the cardiology domain built using this framework.

Sumário

Capítulo 1

Introdução

Neste capítulo são apresentados a motivação e os objetivos da pesquisa realizada nesse trabalho, o histórico dessa pesquisa e a organização dos próximos capítulos.

1.1 Motivação

Com a disseminação dos computadores em todas as áreas comerciais, industriais e de pesquisas, cresce a necessidade de desenvolver software que atenda a diferentes demandas e requisitos. Diferentes linguagens de programação surgiram sendo vistas como a solução ideal em cada momento. O enfoque saiu da programação para a modelagem do problema implicando no surgimento de diferentes métodos de análise e projeto de software. Os problemas tecnológicos foram sendo superados com o desenvolvimento de máquinas cada vez mais poderosas e rápidas. O processamento passou de centralizado para distribuído. E as informações passaram a ser acessíveis em qualquer local e em máquinas de diferentes configurações com o advento da Web. No entanto, o desenvolvimento de software é ainda um contínuo desafio. E nesse desafio um dos principais objetivos tem sido promover a qualidade e a produtividade.

Muitas pesquisas tem se direcionado na busca de identificar quais são as reais dificuldades para alcançar este objetivo e propor soluções para as mesmas. Dessa forma, o processo de desenvolvimento foi organizado e padronizado através da definição de ciclos de vida e de normas para desenvolvimento de software (NBR ISO/IEC 12207, 1998). A avaliação da qualidade foi sistematizada com critérios bem definidos (ISO/IEC 9126/NBR 13596, 1996) e com a definição de técnicas para serem usadas ao longo do desenvolvimento (PRESSMAN, 1997). Ambientes de desenvolvimento de software (ADS) foram propostos como forma de automatizar o processo de desenvolvimento e fornecer ferramentas de apoio aos desenvolvedores para

todas as atividades de desenvolvimento de software passíveis de automação, trabalhando de forma integrada num ambiente único para os engenheiros de software.

No início de 1994, a área de Engenharia de Software da COPPE/UFRJ estabeleceu uma parceria com a Unidade de Cardiologia e Cirurgia Cardiovascular/Fundação Bahiana de Cardiologia da Universidade Federal da Bahia para desenvolvimento de projetos conjuntos envolvendo o desenvolvimento de software de diferentes tipos, tais como, sistemas baseados em conhecimento, tutores, sistemas de informação hospitalar e telemedicina. Durante esses cinco anos, a diversidade de trabalhos e a alta rotatividade da equipe, pôde evidenciar um outro fator que afeta a qualidade e, principalmente, a produtividade no desenvolvimento de software: o fato da equipe de desenvolvimento trabalhar num domínio de aplicação que não conhece.

Analisando esse problema é possível perceber que esse fator é comum em várias organizações, seja com a integração de novos desenvolvedores a equipes de sistemas em desenvolvimento, ou com as empresas externas contratadas face à terceirização dos serviços, abordagem cada vez mais utilizada para o desenvolvimento de software nas organizações. Com a diversidade dos trabalhos, os desenvolvedores de empresas contratadas precisam conhecer um domínio diferente e, geralmente, com conhecimento disseminando entre várias pessoas ou sistemas da empresa contratante. Os profissionais da empresa contratante, por sua vez, estão envolvidos em suas próprias atividades o que leva a não disponibilidade do tempo necessário para fornecer aos contratados o conhecimento do domínio.

Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD), conforme definidos nesse trabalho, surgiram dessa constatação, propondo um apoio no entendimento do domínio para desenvolvedores que não têm familiaridade ou experiência em realizar trabalhos no domínio considerado. ADSOD são definidos tendo como base os tradicionais ambientes de desenvolvimento de software surgidos na década de 80, mas incorporando um novo fator: o conhecimento de um domínio específico.

1.2 Histórico da Pesquisa

O tema deste trabalho foi idealizado na UCCV/FBC ao longo do desenvolvimento dos diferentes projetos e mais particularmente de nosso trabalho no projeto SEC, um sistema especialista em cardiologia para diagnóstico de infarto agudo do miocárdio (RABELO *et al.*, 1997). Logo que a equipe de Engenharia de Software da

COPPE/UFRJ começou a trabalhar no projeto, foi percebida a necessidade do entendimento de conceitos do domínio como um todo, independente dos aspectos referentes a como é feito o diagnóstico e o que deveria ser considerado no sistema a ser desenvolvido. Dessa forma, os cardiologistas que participaram do projeto prepararam uma aula, para explicar o que significava o estudo do coração (cardiologia), o que era considerado importante nesse estudo e que tipos de tarefas são realizadas nesse estudo. Para desenvolver o sistema foi estabelecido um processo de desenvolvimento específico para sistemas baseados em conhecimento (WERNECK, 1995, WERNECK *et al.*, 1995, WERNECK *et al.*, 1997) com marcos e pontos de controle bem definidos para se realizar avaliação da qualidade (OLIVEIRA, 1995, OLIVEIRA *et al.*, 1995, OLIVEIRA *et al.*, 1996a) considerando atributos particulares desse tipo de sistemas e técnicas da literatura. No entanto, ao longo de todo o desenvolvimento percebeu-se a contínua necessidade do estudo do domínio e o esclarecimento de conceitos mais básicos de cardiologia para garantir a boa realização de entrevistas e análise de requisitos. Com os sucessivos projetos e as diferentes equipes de desenvolvimento a situação foi a mesma.

A partir dessa experiência, foi identificado que o processo de desenvolvimento deveria considerar esse constante estudo do domínio pois esta, como qualquer outra atividade, requer planejamento e tempo impactando no prazo dos projetos. Além disso, percebeu-se a importância de organizar o conhecimento do domínio de forma bem estruturada e fácil de ser entendida servindo como fonte de conhecimento para os desenvolvedores de software. Dessa forma iniciou-se a pesquisa em ADSOD. Essas idéias foram inicialmente apresentadas em congressos de informática médica (OLIVEIRA *et al.*, 1996b, 1996c) e como tema de tese de doutorado no Workshop de Teses em Engenharia de Software (OLIVEIRA, 1996a).

A partir daí iniciamos o trabalho de pesquisa para a identificação de como representar o domínio e como prover o apoio orientado ao domínio para os desenvolvedores de software. Após um primeiro estudo sobre aquisição e representação do conhecimento (OLIVEIRA *et al.*, 1996d), iniciamos o trabalho em ontologias e definimos a primeira versão de uma ontologia para cardiologia (OLIVEIRA e MENEZES, 1997). Nesse contexto, participamos de um simpósio específico sobre organização do conhecimento realizado na Universidade de Stanford (Estados Unidos) onde apresentamos um poster ("Towards an Ontology for the Construction of a Domain-Oriented Software Development Environment for Cardiology") e realizamos doutorado sanduíche de setembro de 1997 a junho de 1998 na Universidade de Ottawa

(Canadá) onde trabalhamos com o professor Stan Matwin e Doug Skuce. A evolução do trabalho, até esse momento, foi apresentada no consórcio internacional de teses de doutorado da conferência CAiSE (OLIVEIRA *et al.*, 1998a). Uma experiência de uso do conhecimento na construção de uma ferramenta específica para o domínio de cardiologia (KED – Um Editor de Conhecimento para Cardiologia) foi apresentada no Simpósio Brasileiro de Engenharia de Software (OLIVEIRA e MATWIN, 1998).

Retornando ao Brasil a pesquisa se direcionou para a definição das características do ambiente e sua construção no que se refere à definição do conhecimento no ambiente (OLIVEIRA *et al.*, 1999a), ao uso desse conhecimento através de atividades no processo de desenvolvimento de software (OLIVEIRA *et al.*, 1999b), à preocupação com a avaliação da qualidade dos produtos de software desenvolvidos em um domínio (OLIVEIRA *et al.*, 1999c, CERQUEIRA *et al.*, 1999) e à construção do ambiente CORDIS, um ADSOD específico para cardiologia (OLIVEIRA *et al.*, 1999d, 1999e).

1.3 Objetivo da Tese

O objetivo desse trabalho é, portanto, definir os principais fundamentos para definição e construção de um ADSOD que considere o uso do conhecimento do domínio durante o desenvolvimento de software.

Nesse sentido, a pesquisa inicialmente se direcionou para a definição de como o conhecimento do domínio deveria ser organizado e disponibilizado no ambiente. A resposta a esta questão veio de um estudo recentemente bastante explorado na área de Inteligência Artificial (IA): o uso de ontologias. Conceito provindo da Filosofia que define como uma explicação sistemática da existência, ou seja, o que significa existir, ontologia tem sido utilizada na comunidade de IA como uma especificação explícita de uma conceituação (GRUBER, 1995). Em outras palavras, é a definição de conceitos e suas relações, propriedades e restrições expressas formalmente.

Ontologia tem sido utilizada para diferentes propósitos. Nesse trabalho, os diferentes usos de ontologia foram organizados dentro do processo de desenvolvimento com o objetivo de guiar o trabalho do desenvolvedor através dessa descrição explícita do domínio. Devido a esse aspecto e à característica intrínseca dos ADS de serem uma automação do processo de desenvolvimento, foi realizada uma pesquisa sobre como definir o processo de desenvolvimento a ser utilizado de forma a embutir no mesmo, o estudo do domínio e as atividades particulares do ambiente de trabalho no qual o

sistema será desenvolvido. Dessa forma, diferentes normas para processo de software (ISO/IEC 12207 e SPICE (EMAM *et al.*, 1998)), foram estudadas e consideradas.

Definida a organização do conhecimento e o seu uso no processo de desenvolvimento, foi, então, estabelecido um conjunto de requisitos para a construção de ADSOD e para seu uso durante o desenvolvimento de software. Para permitir a construção de ADSOD, esses requisitos foram acrescentados à Estação TABA, definindo extensões específicas para criar uma estrutura possível de ser utilizada na definição e instanciação de ambientes para diferentes domínios. A Estação TABA (ROCHA *et al.*, 1990) é um projeto de larga escala realizado na COPPE/UFRJ e que visa a construção de uma estação de trabalho configurável para apoiar o engenheiro de software no desenvolvimento de produtos de software para diferentes domínios de aplicação.

A pesquisa sobre as características de diferentes domínios foi, inicialmente, realizadas por WERNECK (1990) que definiu uma taxonomia de domínios de aplicação. Entretanto, essa idéia de atenção às características do domínio no ambiente de desenvolvimento de software não foi, nesse momento, implementada na Estação TABA pois os trabalhos tomaram outra direção: criar uma infra-estrutura tecnológica adequada para o desenvolvimento de software segundo diferentes paradigmas (como sistemas baseados em conhecimento, orientado a objetos e baseados em reutilização) a partir da disponibilização de ferramentas para as diferentes etapas do desenvolvimento (TRAVASSOS, 1994, WERNECK, 1995, WERNER *et al.*, 1997, FALBO, 1998).

Essa tese volta à questão da particularização de ambientes de desenvolvimento de software para domínios específicos, considerando não apenas o suporte tecnológico mas também a integração do conhecimento do domínio nesses ambientes. A partir de agora, a Estação TABA é capaz de instanciar ambientes considerando não apenas o processo de software e as tecnologias de desenvolvimento, mas também o domínio da aplicação.

Para experimentar essas idéias foi construído, CORDIS, um ADSOD específico para o domínio de Cardiologia. O conhecimento do domínio foi elicitado e organizado, o processo de desenvolvimento específico foi definido e ferramentas foram implementadas.

1.4 Organização da Tese

Essa tese é composta de seis capítulos além dessa introdução.

No capítulo 2 são apresentadas as principais características de ambientes de desenvolvimento de software no que se refere à sua definição, arquitetura e evolução. Nesse capítulo é, também, apresentada a Estação TABA no que se refere a seus objetivos, funcionalidade, trabalhos já realizados e ambientes instanciados.

No capítulo 3 é enfocado o tema da orientação ao domínio. São apresentadas pesquisas na área de engenharia de software e, mais detalhadamente, pesquisas da área de inteligência artificial no que se refere ao estudo de ontologias. São, também, apresentados projetos que consideram o domínio de diferentes formas e com diferentes objetivos.

No capítulo 4 são apresentadas as principais características dos Ambientes de Desenvolvimento de Software Orientados ao Domínio: o modelo do conhecimento do domínio utilizado, seu processo de construção, as ferramentas importantes para sua definição e uso, seus usuários e os aspectos de orientação ao domínio.

No capítulo 5, são apresentadas as extensões realizadas na Estação TABA para permitir a construção desses ambientes englobando a definição dos novos requisitos acrescentados à Estação, o modelo modificado, a arquitetura definida e as ferramentas implementadas.

No capítulo 6, é apresentado o ambiente CORDIS, um ADSOD para Cardiologia, descrevendo-se todos os aspectos necessários para a sua definição e construção. Um ambiente específico usando essas definições é instanciado através da Estação TABA tornando-se operacional para experimentos de uso.

Finalmente, no capítulo 7, são apresentadas as conclusões e contribuições deste trabalho, bem como as perspectivas para futuras pesquisas.

Capítulo 2

Ambientes de Desenvolvimento de Software

Este capítulo apresenta as principais características de ambientes de desenvolvimento de software no que se refere à sua conceituação, construção, tecnologias utilizadas e experiências realizadas. É desenvolvida, ainda, uma avaliação do desenvolvimento e uso desses ambientes buscando ressaltar a importância e necessidade da orientação ao domínio. Finalmente é apresentada a Estação TABA e os principais trabalhos realizados neste contexto.

2.1 Histórico e Características

Já se passaram três décadas desde o surgimento do termo Ambiente de Desenvolvimento de Software (ADS) e do início da pesquisa para busca do apoio automatizado ao desenvolvimento de produtos de software de forma integrada e controlada. Diferentes pesquisas foram realizadas, indo desde a utilização de ferramentas isoladas (como compiladores e depuradores), consideradas como a primeira geração dos ADSs, até as definições e construções de estruturas que permitissem a integração, portabilidade e interoperabilidade de ferramentas (BROWN, 1992, PENEDO, 1993), culminada com o surgimento dos ambientes centrados em processo que defendem a necessidade da integração de ferramentas incorporadas a um processo de desenvolvimento de software específico da organização (GARG e JAZAYERI, 1995).

Ambiente de Desenvolvimento de Software (ADS) é um sistema computacional que provê suporte para o desenvolvimento, reparo e melhorias em software e para o gerenciamento e controle destas atividades; contendo uma base de dados central e um conjunto de ferramentas de apoio. A base de dados central atua como um repositório para todas as informações relacionadas ao projeto ao longo do seu ciclo de vida e as ferramentas oferecem apoio para as várias atividades técnicas e gerenciais passíveis de automação que devem ser realizadas no projeto (MOURA, 1992). ADS envolve o apoio

a atividades individuais e ao trabalho em grupo, ao gerenciamento de projeto, ao aumento da qualidade geral dos produtos e ao aumento da produtividade, permitindo ao engenheiro de software acompanhar o trabalho e medir, através de informações obtidas ao longo do desenvolvimento, a evolução dos trabalhos (TRAVASSOS, 1994).

As estruturas de organização de ADSs propõem diferentes componentes para compor os ambientes e um conjunto de funcionalidades para prover o apoio desejado aos desenvolvedores de software de forma integrada. Esse é o caso, por exemplo da estrutura hierárquica em camadas de PENEDO (1993) que considera cinco níveis: *(i)* o nível físico de hardware e sistema operacional; *(ii)* a camada de estruturação com gerenciadores de objetos, interface e do ambiente; *(iii)* a camada de suporte integrado e serviços comuns; *(iv)* a camada para construção e uso de ferramentas; e, finalmente *(v)* o nível de interface com o usuário permitindo adaptações para os projetos específicos. A arquitetura em camadas vem sendo utilizada em projetos concretos para construção de ambientes (CONRADI *et al.,* 1994; RANDALL e ETT, 1995).

Nesta mesma linha, exceto pelo aspecto hierárquico, BROWN (1992, 1993) definiu uma estrutura a partir das funcionalidades (ou serviços) contidas em seus componentes, buscando integração de dados, de gerência de tarefas e dos aspectos de interação com o usuário a partir da comunicação entre os serviços. Esses serviços englobam serviços de interface com o usuário, serviços de gerenciamento de tarefas, serviços de armazenamento e integração de dados e serviços de mensagens. Além disso, são definidos locais específicos para integração de novas ferramentas externas ao ambiente.

Integração de ferramentas é um fator muito importante no contexto de ADS buscando a melhor forma de interagir com o ambiente e estabelecer a abrangência em que deve ser feito o tratamento das informações dentro do ambiente. Para isso, a arquitetura do ambiente deve permitir que as ferramentas possam cooperar umas com as outras, que as ferramentas possam ser conectadas ao ambiente e que o ambiente forneça suporte metodológico ao desenvolvimento (TRAVASSOS, 1994). A forma mais usada, embora não muito flexível, de integrar ferramentas é a partir da utilização de modelos de dados comuns, armazenados em uma base de dados e compartilhados pelas ferramentas do ambiente. Os diferentes projetos encontrados na literatura técnica propõem subsistemas específicos, técnicas e mecanismos para tratar essa questão (CONRADI *et al.,* 1994; RANDALL e ETT, 1995. BANDINELLI *et al.*, 1996).

Estendendo os aspectos de integração proposto por BROWN (1992), TRAVASSOS (1994) propôs a inclusão de um componente para incorporar mecanismos para o

armazenamento e utilização de conhecimento descrito e adquirido ao longo do processo de desenvolvimento. O componente de conhecimento é, então, considerado como fator essencial para garantir a integração de ferramentas por manter o conhecimento sobre métodos, processos e domínios de aplicação úteis para as ferramentas. A proposta de TRAVASSOS (1994) baseou-se na idéia de que um ADS está inserido no contexto de um produto de software e é o responsável, com seu conjunto de funcionalidades e informações associadas, por traduzir, de acordo com a necessidade do usuário, uma representação do mundo real para o computacional através de um conjunto de componentes que consideram desde o aspecto do hardware (sistema computacional) até a interação e a comunicação com os usuários (componente de interface e interação) além da gerência de processos, armazenamento de dados e desenvolvimento de produtos em geral. Posteriormente, o componente de conhecimento foi melhor definido numa arquitetura genérica, denominada servidores de conhecimento, que permite a inclusão de fonte diversificada de conhecimento que seja útil à construção de ferramentas (FALBO, 1998, FALBO *et al.*, 1999a). Esses trabalhos foram realizados no contexto do Projeto TABA (ROCHA *et al.*, 1990). A arquitetura da Estação TABA será descrita em detalhes na seção 2.3

Embora num contexto mais restrito e utilizado em ferramentas específicas, a integração do conhecimento em ADS já tinha sido explorada desde 1988 no trabalho de PUNCELLO (1988) com a proposta de assistentes inteligentes. Vários trabalhos surgiram nesta área com diferentes objetivos, como por exemplo: apoio à verificação de alterações de projeto para melhorar a qualidade (FREITAS, 1998), apoio ao projeto de interface (BOLCER, 1995), apoio ao planejamento de projetos (MADACHY, 1995, TOTH, 1995, SHEPPERD *et al.*, 1997, CHATZOGLOU *et al.*, 1998), apoio à medição e avaliação da qualidade (LIU, 1998, SELBY *et al.*, 1991), entre outros. Nos últimos anos, essa assistência passou a ser mais utilizada para domínios particulares e em contextos de desenvolvimento específicos, como sistemas de informação orientados a objetos (GOMAA *et al.*, 1996) e sistemas baseados em conhecimento (SCHREIBER *et al.*, 1995).

Com a definição de ADS e a preocupação em torná-los efetivos, uma outra linha de pesquisa que começou a se destacar foi arquitetura de software, na busca de uma melhor definição de como os diferentes elementos, a partir do qual as aplicações são construídas, são organizados, relacionados e descritos para permitir sua composição ou simples uso no desenvolvimento de uma aplicação (SHAW e GARLAN, 1996).

Arquitetura de software estabelece esses aspectos sendo definida como uma estrutura de componentes do sistema, seus inter-relacionamentos, princípios e diretrizes que governam os projetos e sua evolução ao longo do tempo (CLEMENTS, 1996).

Diferentes estilos arquiteturais foram propostos para cada família de aplicações podendo ser combinados ou especializados para gerar novos estilos (SHAW e GARLAN, 1996, BRAGA e TRAVASSOS, 1998). Esses estilos podem estar diretamente associados ao nível arquitetural (como sistemas cliente-servidor, *pipe-filter* e arquitetura em camadas); ao método e notação de projetos específicos (como organizações orientadas a objetos e fluxo de dados) e a classes de sistemas (como compiladores e padrões de projeto orientado a objetos). Algumas das principais pesquisas nessa área tem enfocado a modelagem e estruturação dessas arquiteturas com o objetivo de facilitar a reutilização no desenvolvimento de software (GARLAN *et al.*, 1994, TAYLOR *et al.*, 1995, DAVIS e WILIANS, 1997, ROSSAK *et al.*, 1997, ROBBINS e REDMILES, 1996, ROBBINS *et al.*, 1997).

2.2 Ambientes de Desenvolvimento Centrados em Processo

Software não deve ser desenvolvido de forma *ad hoc* e sim seguindo um conjunto de atividades bem definido e ordenado. Este conjunto ordenado de atividades, mais os recursos utilizados e produzidos formam um processo de software. Um processo de software envolve ainda um conjunto de ferramentas e técnicas para apoio à realização das atividades (PFLEEGER, 1998).

O processo de software pode ser descrito de diferentes formas sendo geralmente organizado em um modelo de processo que contém suas principais características. Um modelo de processo é, portanto, uma representação de um processo (GARG e JAZAYERI, 1995). Muitos modelos de processos tem sido propostos na literatura: como o tradicional modelo cascata, os modelos de prototipagem e o modelo espiral (PRESSMAN 1997, PFLEEGER 1998). Na teoria, muito dos modelos de processo são semelhantes mas na prática não o são. Segundo PFLEEGER (1998) um processo de software deve ser definido para as situações específicas em que se deseja que seja utilizado. Com a modelagem de um processo específico e, consequentemente, a análise dos seus sub-processos e atividades, é estabelecido um entendimento comum na equipe sobre as atividades, recursos e restrições no desenvolvimento de software, além de esclarecido o que deve ser feito e os objetivos do desenvolvimento (GARG e JAZAYERI, 1995).

O reconhecimento da necessidade de uma modelagem explícita do processo de software tem tido uma profunda influência nas pesquisas em engenharia de software em diferentes direções. A *International Standard Organization* (ISO) estabeleceu uma norma padrão para processo de software (ISO/IEC 12207, 1995) propondo um *framework* com terminologia bem definida e contendo processos, atividades e tarefas que devem ser aplicadas durante a aquisição, fornecimento, serviço, desenvolvimento, operação e manutenção de software. A norma descreve a arquitetura de um processo em geral, mas não especifica em detalhes como implementar ou desempenhar estas atividades, nem descreve nome, formato ou conteúdo da documentação a ser gerada, o que deve ser definido pelo usuário, ou seja, a organização que irá utilizá-lo. O *Software Engineering Institute* (SEI) propôs o *Capability Maturity Model* (CMM) (PAULK *et al.*, 1995) através do qual pode-se avaliar o processo de software de uma organização, identificando o seu nível de maturidade e, assim, indicar a melhoria sistemática do processo através da definição de áreas chaves para se alcançar o próximo nível. Na mesma linha, um esforço internacional de diferentes pesquisadores propôs o SPICE (*Software Process Improvement and Capability dEtermination*) (EMAM *et al,* 1998) com o objetivo de desenvolver um padrão de avaliação que fosse aplicável tanto para melhoria do processo quanto para determinação da capacidade de uma organização, aplicável a qualquer domínio, necessidades de negócio ou tamanho da organização, sendo independente da estrutura da organização, modelos de ciclos de vida, tecnologias ou métodos de software. O SPICE está sendo reorganizado e será estabelecido como norma ISO/IEC 15504, sendo redefinidos nomes e conceitos para que seja perfeitamente compatível com a ISO/IEC 12207.

Segundo RADER e BROWN (1995), um processo para ser efetivo deve utilizar diferentes ferramentas e tecnologias para promover consistência e completude, e deve prover assistência automatizada para a maioria das tarefas necessárias pelo processo de automatização do processo de software.

Ambientes de desenvolvimento de software centrados em processo (PCSEE-*Process-Centered Software Engineering Environment)* exploram uma definição explícita do processo de software que especifica as atividades, funções e tarefas dos desenvolvedores de software e define como controlar ferramentas de desenvolvimento de software (AMBRIOLA *et al.*, 1997). PCSEE podem ser vistos como a automação de um processo de software.

Automação de processo de software significa guiar a sequência das atividades definidas, gerenciar os produtos que estão sendo desenvolvidos, executar ferramentas necessárias para a realização das atividades, permitir a comunicação entre as pessoas, colher dados de métricas automaticamente, reduzir erros humanos e prover controle do projeto à medida em que este vai sendo executado (CHRISTIE, 1995).

Uma das principais características dos PCSEE é a modelagem de processos podendo ser utilizados diferentes paradigmas, como redes de Petri (BARDINELLI *et al.*, 1995a) e sistemas de regras (CONRADI *et al.*, 1994). PCSEE deve, ainda, permitir a alteração do processo na medida em que ele vai sendo controlado pelo usuário dado que durante o desenvolvimento de um produto pode ser necessário realizar modificações no processo de software, como por exemplo antecipar ou não executar uma atividade. Um processo de software que não possa ser alterado durante a sua execução torna-se inadequado para a construção de um produto de software por sua rigidez (ARAUJO, 1998).

O apoio automatizado de processos de software através de um PCSEE é extremamente útil numa organização por disciplinar o desenvolvimento aumentando a produtividade de grupos, por coordenar as atividades de grandes equipes oferecendo a situação corrente do desenvolvimento e registrando a relação entre os membros da equipe, e, por organizar e disponibilizar documentação não só do processo mas de todos os produtos de software gerados (GARG e JAZAYERI, 1995).

2.3 Projetos Realizados

Desde a definição de ADS e, principalmente, com o surgimento dos ambientes centrados em processo, muitas experiências de construção foram realizadas tanto em ambientes acadêmicos como de empresas. Essas experiências tem como aspecto comum a construção da infra-estrutura necessária para ADS e propostas para integração de ferramentas com o objetivo de fornecer o suporte desejado.

Cada experiência procurou se concentrar em uma característica específica propondo soluções ou alternativas para viabilização da mesma. De forma geral, algumas das principais características encontradas nos ADS apresentados na literatura são:

- apoio à modelagem de processos, como o SPADE (BANDINELLI *et al.*, 1995, BANDINELLI *et al.*, 1996), STATEMATE (HARVEL e NAAMAD, 1996), Process Weaver (CHRISTIE, 1995), SPMS (BANDINELLI *et al.*, 1995) e EPOS (CONRADI *et al.*, 1994);

- apoio às fases de codificação e depuração de código fonte, como por exemplo, o SNIFF+ (SNIFF+, 1999);
- configuração de interface aberta para permitir a integração de ferramentas externas, como o (SNIFF+, 1999), o SPADE (BANDINELLI *et al.*, 1996), o EPOS (CONRADI *et al.*, 1994).
- apoio a gerência de configuração e versões, como por exemplo, o PALAS-X, (BERNAS, 1995) e o EPOS (CONRADI *et al.*, 1994);
- apoio ao desenvolvimento orientado a objetos, como o PALAS-X (BERNAS, 1995) e o PIROL (GROTH *et. al.*, 1995);
- apoio à reutilização de software, como o STARS (RANDALL e ETT, 1995)., o RESOFT (CHENG *et al.*, 1994), Reboot (CONRADI e KARLSSON, 1995);
- apoio à colaboração, como o C-SPE, (GRUNDY *et al.*, 1995), CPCE (LONCHAMP, 1995); e à cooperação, como por exemplo o SPADE (BANDINELLI *et al.*, 1996), o Beyond-Sniff (BISCHOFBERGER *et al.*, 1995), o P-Root&COO (CANALS *et al.*, 1995).

A Estação TABA, apresentada na seção seguinte, procurou integrar essas diferentes características numa arquitetura robusta de um meta-ambiente que permitisse a definição de ADS específicos para experimentação de diferentes tecnologias. Nesse sentido foi desenvolvido um ambiente para desenvolvimento de sistemas baseados em conhecimento (WERNECK, 1995) e com apoio a reutilização (WERNER *et al.*, 1997)

Atualmente, as pesquisas tem se preocupado em definir essas características para domínios particulares com o objetivo de oferecer apoio mais específico para os problemas do domínio em questão. Esses projetos serão detalhados no próximo capítulo.

2.4 A Estação TABA

O projeto TABA foi criado com a preocupação de auxiliar na definição, implementação e execução de Ambientes de Desenvolvimento de Software (ADS). Esta idéia surgiu da constatação de que domínios de aplicação diferentes tem características diferentes e que estas devem incidir nos ambientes de desenvolvimento através dos quais os engenheiros de software desenvolvem aplicações. Para atender este objetivo a Estação TABA possui quatro funções (ROCHA *et al.*, 1990):

(a) auxiliar o engenheiro de software na especificação e instanciação do ambiente mais adequado ao desenvolvimento de um produto específico;

(b) auxiliar o engenheiro de software a implementar as ferramentas necessárias ao ambiente definido;

(c) permitir os desenvolvedores de software o uso da estação através do ambiente desenvolvido, e,

(d) permitir a execução do software na própria estação por ele configurada.

Estas funções são atendidas através de um conjunto de serviços agrupados em quatro ambientes distintos:

- um ambiente especificador e instanciador de ADSs (meta-ambiente TABA), responsável pela função (a);
- um ambiente de construção de ferramenta, responsável pela função (b) e pela incorporação dessas ferramentas ao meta-ambiente TABA;
- um ambiente de desenvolvimento (ADS instanciado) responsável pela função (c), que foi especificado e instanciado pelo meta-ambiente, e,
- um ambiente de execução responsável pela função (d) sendo basicamente o local onde o software pode ser executado.

Para atender a essas funcionalidades os seguintes requisitos foram identificados como requisitos gerais da Estação TABA (TRAVASSOS, 1994):

- **ser configurável:** a Estação TABA deve poder ser configurada para diferentes domínios de aplicação de forma a atender às especificidades do desenvolvimento de software para estes domínios;
- **possuir interface consistente:** a Estação TABA deve possuir mecanismos de interface com o usuário que permitam uma utilização consistente de seus recursos e ferramentas;
- **possuir mecanismo de integração:** a Estação TABA deve permitir, e facilitar, a integração de ferramentas, sejam elas desenvolvidas com a tecnologia utilizada no próprio Projeto TABA (ferramentas internas), ou então, desenvolvidas por outros (ferramentas externas);
- **apoiar a construção de novas ferramentas:** a Estação TABA deve possuir funcionalidades que permitam ao Engenheiro de Software construir, ou adaptar, novas ferramentas para a Estação;
- **possuir conhecimento sobre processo e métodos de desenvolvimento:** a Estação TABA deve possuir conhecimento sobre o processo de desenvolvimento de software, as várias alternativas de modelos para ciclo de vida e sobre os métodos

possíveis de serem utilizados nas várias atividades do processo, bem como sobre a adequabilidade de sua aplicação em diferentes contextos;

- **possuir assistência inteligente ao usuário:** tanto a nível de assistência para o uso da Estação como um todo, quanto dos ADS específicos e das ferramentas disponíveis;
- **possuir um modelo de armazenamento de dados comum,** as informações devem possuir uma forma de representação tal que possibilite às ferramentas compartilhar, e utilizar, estas informações de forma natural e consistente;
- **possuir suporte a reutilização,** possuindo mecanismos que possibilitem a reutilização tanto a nível de código, quanto de especificação e projeto.

Os ADSs configurados pelo meta-ambiente TABA possuem os seguintes requisitos:

- **possuir suporte ao controle e gerenciamento de versões:** é responsabilidade do ambiente controlar e gerenciar as modificações realizadas nos produtos de software construídos ao longo do processo de desenvolvimento, mantendo os documentos gerados disponíveis para os usuários em suas diferentes versões;
- **possuir suporte para gerenciamento de todas as atividades ao longo do processo de desenvolvimento**: a Estação TABA deve controlar e gerenciar o processo de desenvolvimento através da verificação do andamento do trabalho, controle da execução de atividades relacionadas e encadeadas. Este controle de gerenciamento não deve permitir que a ordem de execução das tarefas seja trocada, ou mesmo modificada, sem que haja uma intervenção do engenheiro de software;
- **possuir suporte para medição do produto:** o ADS armazena, ao longo do processo de desenvolvimento, informações relevantes que permitem fazer medidas sobre os produtos do processo de desenvolvimento. Estas métricas devem estar disponíveis para projetos futuros, possibilitando aperfeiçoamentos no processo de desenvolvimento e, consequentemente, melhorias na qualidade dos produtos.
- **apoiar o trabalho cooperativo:** o desenvolvimento de um produto de software, principalmente quando se trata de desenvolvimento em larga escala, se dá através do trabalho de equipes. A coordenação e interação das pessoas envolvidas são necessárias para que as atividades possam ser realizadas corretamente. Como a comunicação entre a equipe ocorre frequentemente ao longo do processo de desenvolvimento, o ambiente deve definir protocolos de comunicação que possibilitem este relacionamento;

- **possuir interfaces customizáveis;** o ADS deve fornecer meios para que seu usuário possa ajustar a interface apresentada às suas preferências pessoais;
- **possuir suporte para avaliação do produto,** a estação TABA deve oferecer suporte para medição e avaliação da qualidade dos produtos nela desenvolvidos.

2.4.1 Estrutura e Integração

A estrutura da Estação TABA (TRAVASSOS, 1994) foi definida como um conjunto de componentes integrados que possuem controle sobre a sua existência, suas informações, estados e funcionalidades básicas associados. Esta definição baseou-se na idéia de que um ADS está inserido no contexto de um produto de software e é o responsável, com seu conjunto de funcionalidades e informações associadas, por traduzir, de acordo com a necessidade do usuário, uma representação do mundo real para o computacional. Para atender a seus requisitos, a Estação TABA possui os seguintes componentes (Figura 2.1):

- **Sistema Computacional**, representado por uma dimensão de hardware e outra do sistema operacional;
- **componente Interface com o Usuário**, responsável pela gerência e controle da interação do usuário com o ADS, garantindo a consistência da apresentação das ferramentas;
- **componente Cooperação,** que trata da comunicação entre os usuários e suas necessidades de interação com outros membros da equipe, possuindo definições de protocolos de comunicação que devem ser utilizados no ADS e auxiliando o componente de Interface com o Usuário no sentido de prover funcionalidades e recursos que o estendam de forma a suportar a interface de grupo;
- **componente Controle dos Processos,** responsável pelo controle e gerenciamento do processo de desenvolvimento através da identificação de atividades, gerência da execução das ferramentas e controle dos papéis e atividades dos usuários;
- **componente Reutilização**, que provê o ADS de mecanismos que possibilitam a reutilização de trabalhos anteriores, a nível da especificação do ADS, de componentes para a construção de ferramentas, bem como de componentes de todo tipo para a construção da aplicação;
- **componente Suporte Inteligente,** que fornece inteligência global ao ambiente, assistindo o usuário na utilização do meta-ambiente e dos ADS, através de assistentes inteligentes;

- **componente Conhecimento**, que incorpora mecanismos para o armazenamento e a utilização de conhecimento, que podem ser utilizados tanto pelo meta-ambiente, quanto pelos ADSs instanciados;
- **componente Repositório Comum**, responsável pelo controle, gerenciamento e armazenamento dos objetos manuseados pelo ambiente e meta-ambiente, e pela consistência e integridade das informações, e,
- **componente Controle de Versões**, que controla e gerencia versões de documentos e itens de software produzidos.

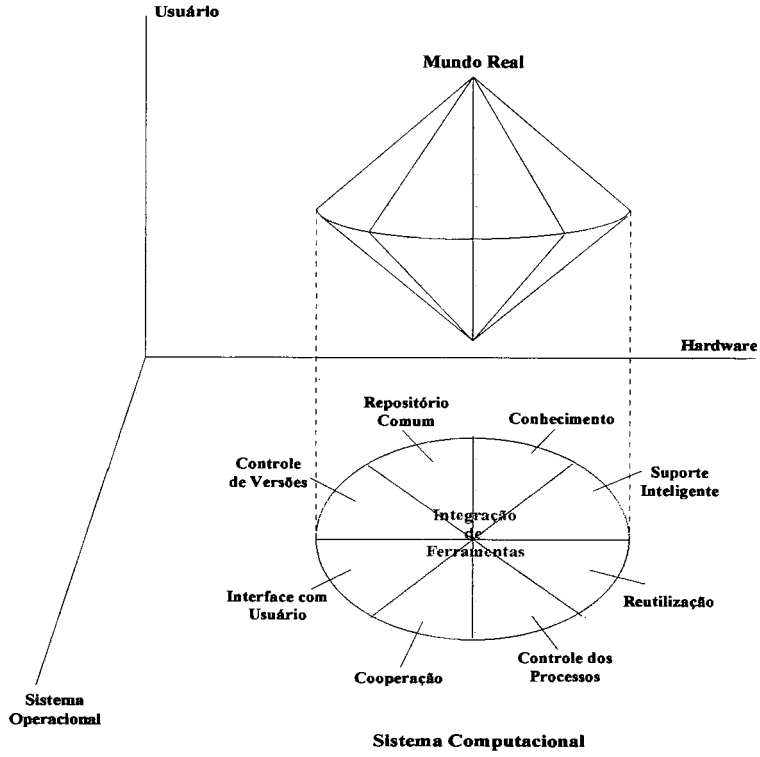

Figura 2.1 - Estrutura de um ADS na Estação TABA (TRAVASSOS, 1994).

A utilização, em conjunto, destes componentes permite a integração de ferramentas ao ambiente e provê recursos para a incorporação de ferramentas externas. A utilização desses componentes define, ainda, a filosofia de integração da Estação TABA e dos seus ambientes instanciados através de quatro tipos de integração (TRAVASSOS, 1994):

- **integração de apresentação e interação,** que possibilita a homogeneização das formas de apresentação das informações e das técnicas de interação com o usuário;

- **integração de gerenciamento e controle**, que provê, para ferramentas e para o ambiente, serviços e funcionalidades que permitam o funcionamento de forma organizada ao longo do processo de desenvolvimento;
- **integração de dados**, que estabelece a forma como as ferramentas da Estação realizarão o tratamento das informações, e,
- **integração do conhecimento**, que torna possível os serviços básicos de armazenamento, gerenciamento e utilização do conhecimento descrito e adquirido ao longo do processo de desenvolvimento.

A integração do conhecimento tem sido constante objeto de estudo no contexto da Estação TABA, sofrendo bastantes modificações desde sua concepção até o momento atual. Nesse sentido, FALBO (1998) propôs a utilização de servidores do conhecimento no qual o conhecimento é modelado para reuso e disponibilizado em um conjunto de componentes que podem ser usados por várias ferramentas a serem desenvolvidas. Esses componentes são construídos a partir de ontologias (FALBO *et al.*, 1999a) que descrevem o conhecimento do domínio de interesse e de modelos de tarefas (BREUKER e VAN DE VELDE., 1994) que descrevem o conhecimento genérico de tarefas aplicáveis a vários domínios.

Um Servidor de Conhecimento é, portanto, uma infra-estrutura de conhecimento sobre um universo de discurso que provê um vocabulário comum, baseado em ontologias, com interpretação definida dos termos no universo de discurso e, um conjunto de máquinas de inferência customizadas para os tipos de problema mais comumente encontrados no universo de discurso em questão. O objetivo dos Servidores de Conhecimento é prover esta infra-estrutura comum, para o desenvolvimento de um conjunto de sistemas baseados em conhecimento em um universo de discurso.

Para permitir o reuso de conhecimento de tarefa, o Servidor de Conhecimento provê *templates* de resolvedores genéricos de problema. Ao invés de prover apenas sistemas de representação e suas máquinas genéricas de inferência, um Servidor de Conhecimento deve prover uma biblioteca de resolvedores genéricos de problemas, passíveis de serem instanciados e adaptados para aplicações particulares.

Quanto ao reuso de conhecimento do domínio, é desejável que a base de conhecimento do Servidor seja modular e baseada em ontologias. Ontologias e suas instanciações podem ser implementadas em módulos de conhecimento, de modo que a base de conhecimento de uma aplicação que se comprometa com uma ou várias

ontologias venha a ser a conjunção dos módulos de conhecimento correspondentes, mais o conhecimento específico da aplicação particular.

Desta forma, a arquitetura de um Servidor de Conhecimento é composta de (Figura 2.2):

- uma base de conhecimento modular, onde cada módulo de conhecimento (MC) contém um corpo de conhecimento reutilizável, construído com base em uma ontologia, e,
- a máquina de inferência do sistema de representação adotado, associada a um conjunto de templates de resolvedores de problemas, para especializá-la para os tipos de problema mais freqüentemente encontrados no universo de discurso.

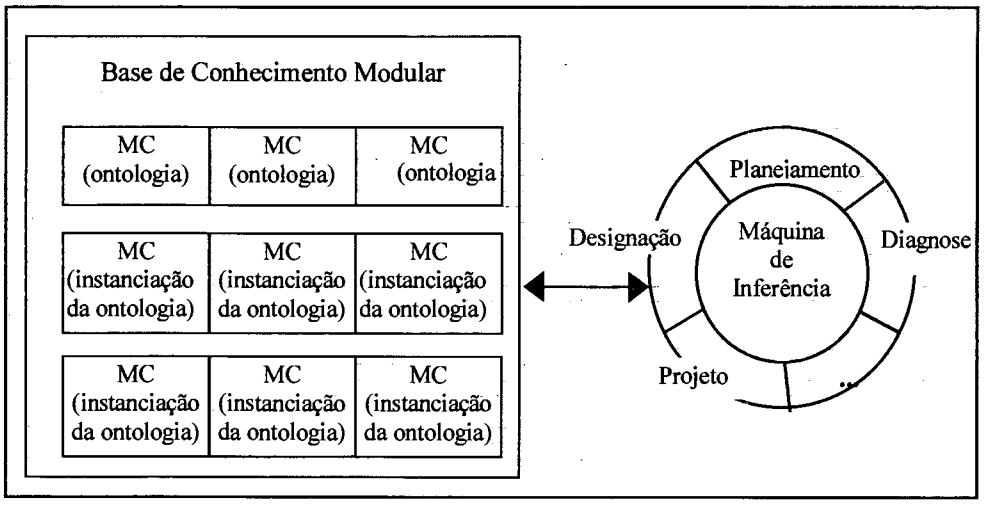

Figura 2.2 - A Arquitetura Geral de Servidores de Conhecimento (FALBO, 1998)

A base de conhecimento modular é essencialmente uma biblioteca de ontologias, pois seu critério de modularização é dado pelo uso de ontologias. Cada ontologia é implementada em um módulo de conhecimento na linguagem do sistema de representação, assim como cada uma de suas instanciações.

A máquina de inferência é aquela provida pelo sistema de representação de conhecimento adotado. Os templates de resolvedores de problemas, por sua vez, implementam modelos de tarefa genéricos para os tipos de problemas que ocorrem com freqüência no universo de discurso do Servidor de Conhecimento.

Os módulos de conhecimento e os templates de resolvedores de tipos de problema guardam estreita relação entre si: os papéis de conhecimento considerados em um

template de resolvedor de problema serão preenchidos com o conhecimento de domínio descrito nos módulos de conhecimento.

No que se refere à tecnologia de representação utilizada para construção desses componentes, a Estação TABA provê uma classe de representação de conhecimento que permite a definição de diferentes tecnologias de representação: rede neural e sistemas lógicos, englobando frames, sistema de programação em lógica e rede semântica. Atualmente, tem-se implementada a classe de sistema de programação em lógica usando Prolog através de uma máquina Prolog externa. A comunicação entre a Estação TABA e esta máquina externa foi feita a partir de um mecanismo de interfaceamento utilizando a linguagem C.

2.4.2 Instanciação de Ambientes de Desenvolvimento de Software

A configuração de um ADS a ser instanciado na Estação TABA é feita a partir da definição de um processo de desenvolvimento que se caracteriza pela descrição de uma sequência de atividades, suas ferramentas de apoio, produtos de software gerados e recursos consumidos. Diferentes trabalhos foram realizados para apoio à definição e execução de processos. Nesse sentido, nos dois últimos anos, três trabalhos atuaram de forma direta e complementar garantindo não só a definição como o acompanhamento e alteração do processo durante a sua execução:, a alteração do processo baseado em descrições padrões proposta por VASCONCELOS (1997), a máquina de processo proposta por ARAÚJO (1998) e a integração do conhecimento permitindo assistência inteligente na definição de processo, proposta por FALBO (1998).

VASCONCELOS (1997) propôs a utilização de padrões para organizar as descrições de processos. As descrições de processo, são então representadas não apenas nas classes de conhecimento, mas , também, nas classes que representam os padrões. Esses padrões servem como elementos de organização e extensão do conhecimento sobre processos, podendo ser usado para processos e atividades. Dessa forma, uma descrição de processo, a partir de padrões, é estruturada a través de um padrão de processo e diversos padrões de atividades.

ARAÚJO (1998) estabeleceu uma estrutura para definição do processo de forma a suportar essa flexibilização, apoiando adaptações a serem feitas durante o desenvolvimento. Estas adaptações referem-se à inclusão ou exclusão de atividades, mudanças em seus relacionamentos e alterações em seus recursos, artefatos e hierarquia de atividades. As alterações permitidas não afetam a vida passada de execução do

processo, ou seja, as mudanças são válidas a partir do ponto corrente de execução do processo.

FALBO (1998) construiu um Servidor de Conhecimento de Processo no qual uma ontologia sobre processo de desenvolvimento de software foi elaborada alterando a estrutura da descrição do conhecimento sobre processo para incluir todos os detalhes sobre o conhecimento relevante sobre processo identificado na ontologia. Esta descrição de conhecimento é utilizada na ferramenta Assist-Pro (FALBO, 1999b), contendo o conhecimento sobre vários modelos de ciclo de vida. Assist-Pro auxilia o engenheiro de software na descrição do processo específico do ambiente a ser instanciado. Para isso, a ferramenta entrevista o engenheiro de software procurando identificar características do projeto a ser desenvolvido. É, então, sugerido o(s) ciclo(s) de vida que mais se adequam ao problema e o engenheiro de software pode descrever o processo de desenvolvimento específico para o projeto. A partir dessa descrição será definido o modelo à medida que o processo vai sendo instanciado conforme definido por ARAUJO (1998).

Assim, a ontologia de processo (FALBO, 1998) serve como fonte de conhecimento para a descrição de um processo específico que é instanciado pela máquina de processo (ARAUJO, 1998) e, que por sua vez, é acompanhado e adapatado durante execução por ferramentas específicas para alterações no processo de desenvolvimento feitas de forma a não causar inconsistências no modelo existente (VASCONCELOS, 1997).

Dois ambientes já foram definidos e implementados na Estação TABA: Orixás (WERNECK, 1995) e Memphis (WERNER *et al.*, 1997).

Orixás (WERNECK, 1995) foi o primeiro ambiente instanciado utilizando o modelo de integração da Estação TABA. Este ambiente tem como objetivo apoiar o desenvolvimento de Sistemas Baseados em Conhecimento a partir da definição de um processo de desenvolvimento bem definido, do uso sistemático de ferramentas e de procedimentos gerenciais e de controle da qualidade. Foi definido um modelo de ciclo de vida específico e uma extensão para o método de modelagem KADS (HICKMAN *et al.*, 1992) para a fase de projeto. Foi definido ainda um conjunto de ferramentas para apoiar as atividades de construção, avaliação do produto e gerência do processo de desenvolvimento

O ambiente Memphis (WERNER *et al*, 1997) tem como objetivo o apoio ao desenvolvimento de software baseado em reutilização. Memphis é um ambiente instanciado TABA e, portanto, está baseado em um modelo de processo de desenvolvimento de software baseado em reutilização (ROCHA *et al.*, 1996) sendo

apoiado pelo uso sistemático de ferramentas que possibilitam a aplicação do método adotado, e de procedimentos gerenciais de controle da qualidade e de reutilização adequados.

2.5 Considerações Finais

A perspectiva do apoio automatizado para o desenvolvimento de software proposta pelos ADS na década de 80 estimulou a pesquisa em diferentes linhas da computação desde o simples apoio às atividades de programação, até aspectos mais complexos de gerência e desenvolvimento de software. No entanto, ADS na sua integridade ainda não são uma realidade com ampla utilização comercial.

Os problemas são os mesmos apontados por BROWN(1993) em 1993: falta de documentação do custo-benefício do ADS, com referência a problemas encontrados e benefícios alcançados; tamanho dos ADS, que para gerenciar vários aspectos do desenvolvimento de software se tornam grandes e complexos; alto custo de construção e manutenção, e, a inexistência de ambientes configuráveis que possam ser customizados para diferentes organizações, projetos e pessoas.

Com isso, soluções simplificadas foram disponibilizadas através de ferramentas CASE (*Computer-Aided Software Engineering*) que apoiam algumas atividades do desenvolvimento de software, geralmente, as atividades de análise, projeto e codificação. Pesquisas apresentadas no *International Workshop on Computer-Aided Software Engineering* em 1995 (HARDY *et al.*, 1995, KROGSTIE, 1995) indicaram que o número de ferramentas CASE utilizadas nas empresas duplicou em quatro anos e que a tendência era aumentar muito mais. No entanto, na mesma conferência CHURCH e MATTEWS (1995) mostraram que, apesar do grande número de ferramentas no mercado, nenhuma se mostrava ainda completamente adequada quando se buscavam características como facilidade de uso, verificação de consistência, geração de documentação, código e integração às atividades do ciclo de vida do projeto.

BROWN (1993) também acertou quando apostou que maior enfoque deveria ser dado para ambientes orientados a processo, mas que deveria se ter um melhor entendimento do processo de desenvolvimento de software antes da sua automação. Embora vários ambientes tenham sido construídos, muito poucos são experimentados em contextos industriais. Segundo AMBRIOLLA *et al.* (1997) isso se deve, principalmente, ao fato de que muitas organizações ainda são imaturas no que se refere ao uso e melhoria de processo, e justifica fazendo uma correlação entre a maturidade de

uma organização e a tecnologia de processo. Nessa análise, organizações no nível 1 e 2 do CMM estão mais interessadas em linguagens para especificação de requisitos e avaliação, procurando explorar diferentes alternativas antes de estabelecer um processo efetivo de desenvolvimento de software sendo praticamente impossível a modelagem de processos em ambientes automatizados. Organizações acima do nível 3 já tem seus processos descritos e detalhados (mesmo que informalmente) e poderiam utilizar linguagens de implementação de processo para explorar o uso de ambientes centrados em processo. Contudo, muito trabalho experimental ainda é necessário para que isso seja possível.

Disponibilizar ambientes genéricos é, realmente, um esforço bastante oneroso. O benefício desses ambientes pode não ser aceitável. A utilização de ferramentas incompatíveis com as estratégias de desenvolvimento e com a própria organização pode implicar em mudanças drásticas, não desejáveis, na gerência e desenvolvimento de software (HENNINGER, 1996). Para que se tenha melhores resultados, apoio específicos para uma organização e para um determinado contexto devem ser propostos e implementados. Nesse sentido, diferentes pesquisas vem sendo realizadas para a definição e construção de ADS que considerem domínios específicos. A orientação ao domínio passou a ser amplamente discutida e investigada. Com a orientação ao domínio desenvolvedores de software podem trabalhar diretamente com o domínio do problema e ter assistência específica no contexto do seu trabalho.

Esse tese apresenta uma experiência nessa linha, procurando integrar aspectos de inteligência artificial nos ambientes de desenvolvimento de software tradicionais como uma forma de trabalhar com o domínio no apoio ao desenvolvimento.

No próximo capítulo, serão apresentadas as principais questões relacionadas à orientação ao domínio, assim como os principais projetos na área.

Capítulo 3

Orientação a Domínio no Desenvolvimento de Software

Este capítulo apresenta conceitos, características, metodologias de desenvolvimento e projetos concretos que enfocam orientação a domínio no desenvolvimento de software. Uma atenção especial é dada para a conceituação de ontologias pelo fato de serem utilizadas como aspecto principal para a orientação a domínio proposta nessa tese.

3.1 Orientação a Domínio

Orientação ao domínio é um esforço dos pesquisadores de, em vez de se preocupar com soluções genéricas, atender as características particulares de um determinado contexto. O enfoque no domínio vem sendo explorado por diferentes grupos de pesquisa tanto na área de Engenharia de Software como na área de Inteligência Artificial. Nesta seção, serão apresentadas a origem e a evolução da orientação a domínio enfocando os principais projetos na área.

3.1.1 Evolução da Orientação a Domínio

Domínio e orientação a domínio são conceitos importantes que norteiam qualquer princípio para atividades ou objetos do mundo. Crianças, por exemplo, são educadas aprendendo sobre diferentes domínios e se tornam profissionais especializados em um domínio específico. Da mesma forma, sistemas envolvendo software são desenvolvidos em um contexto particular, determinado a partir de vários fatores como a organização para a qual está sendo desenvolvido, o problema que busca atender e mesmo a tecnologia que utiliza.

A percepção da importância do domínio no desenvolvimento de projetos de software, bem como da necessidade de identificação do domínio específico e orientação do trabalho com o uso de informações do domínio foi feita por PRIETO-DÍAZ (1987) quando apresentou os primeiros trabalhos sobre análise de domínio. Desde então, vários

trabalhos tem sido realizados enfocando domínios específicos. Esses trabalhos consideram o domínio de diferentes formas, como por exemplo, uma área para a qual se aplica o desenvolvimento de sistemas, uma família de sistemas, um problema de interação entre pessoas, etc.

Não importa exatamente qual o enfoque dado, e sim a direção de trabalho, construindo modelos e estruturas que apoiem a sua implementação e uso no desenvolvimento de software. Análise de domínio tem agora mais de uma década de existência. Definida para procurar organizar informações de um domínio de forma a torná-las reutilizáveis, a análise de domínio tem sido continuamente explorada por pesquisadores de Engenharia de Software. Dentro dessas pesquisas foram identificados cinco níveis de reutilização de informações no desenvolvimento de software (MILI *et al.*, 1995): *(i)* o conhecimento ambiental, que refere-se ao conhecimento sobre transferência de tecnologias e sobre os sistemas em uso; *(ii)* o conhecimento externo, que refere-se ao conhecimento do processo de desenvolvimento utilizado e sobre o domínio ou área de aplicação para o qual os sistemas são desenvolvidos; *(iii)* as arquiteturas funcionais, que referem-se à especificação de funções e objetos de dados; *(iv)* as estruturas lógicas que consistem de processos e arquiteturas de dados, e, *(v)* os fragmentos de código. De certa forma, essa classificação serviu de guia para a definição e estruturação de informações nos diferentes trabalhos de orientação a domínio, que procuraram organizar as informações do domínio em um conjunto de itens que pudessem ser reutilizados ao se descrever e construir software.

Uma das definições mais aceitas e citadas sobre análise de domínio é a proposta por PRIETO-DÍAZ (1990) que define análise de domínio como um processo no qual a informação utilizada no desenvolvimento de software é identificada, capturada e organizada com o propósito de torná-la reutilizável na criação de novos produtos. Essa informação capturada refere-se a todos os produtos de software gerados durante o desenvolvimento e que possam ser reutilizados incluindo requisitos, modelos, código, casos de teste, comentários, projetos e outros (VALÉRIO *et al.*, 1999).

Segundo GAMBHIR (1997) análise de domínio é geralmente realizada através do exame de vários sistemas existentes com o objetivo de determinar similaridades entre os mesmos produzindo um conjunto de requisitos de alto nível que descreve as necessidades do usuários. Todos os sistemas de um *domínio do problema* são, então, generalizados em um modelo, chamado *modelo do domínio*, com maior nível de abstração do que o modelo produzido pela análise de sistemas (PRIETO-DÍAZ, 1987).

O domínio do problema é uma área de aplicação, um campo para o qual sistemas são desenvolvidos (como, por exemplo, sistemas de reserva de passagens aéreas, sistemas de controle e comunicação, planilhas, etc.) (ARANGO e PRIETO-DÍAZ, 1991) e na qual todos os sistemas contidos compartilham um conjunto de funcionalidades ou dados (GAMBHIR, 1997). *Modelo do domínio* é o produto gerado pela análise de domínio e deve conter informações sobre três aspectos do domínio do problema (ARANGO E PRIETO-DÍAZ, 1991): *(i)* conceitos para capacitar a especificação do sistema dentro do domínio, *(ii)* planos para descrever como mapear especificações em código, e, *(iii)* razões para a especificação de conceitos, suas relações e a relação com seus plano de implementação (ARANGO E PRIETO-DÍAZ, 1991).

O trabalho com análise de domínio baseia-se nas seguintes crenças: *(i)* a necessidade de uma especificação para um domínio de informações reutilizáveis, que refere-se à identificação, aquisição e representação das informações dentro de um domínio; *(ii)* o nível de coesão dos problemas do domínio, que torna possível a captura do conhecimento necessário para a solução de diversos problemas, sob a forma de um conjunto finito e pequeno de descrições reutilizáveis, e, *(iii)* a estabilidade dos problemas do domínio, que torna possível diminuir o custo da captura e representação da informação através da reutilização durante longos períodos e em diversas situações e lugares (ARANGO E PRIETO-DÍAZ, 1991).

A demanda de organização e tecnologia do sistema de reutilização não fazem parte da análise de domínio, mas são complementares ao processo de converter um conhecimento comum em bibliotecas reutilizáveis. A engenharia do domínio refere-se ao conjunto de atividades para alcançar esta conversão de forma controlada e confiável (ARANGO E PRIETO-DÍAZ, 1991) e, de forma geral, é composta por três atividades: análise de domínio, especificação da infra-estrutura e projeto da infraestrutura. A ***análise do domínio*** envolve dois aspectos: a análise conceitual, que se refere à identificação e aquisição da informação requerida para especificar sistemas dentro do domínio, e, a análise construtiva, que se refere à identificação e aquisição da informação requerida para implementar essas especificações. Na ***especificação da infra-estrutura*** são feitas a seleção e a organização da informação reutilizável no modelo para ajustá-la aos padrões do ambiente, resultando numa arquitetura para informação reutilizável. E, o ***projeto da infra-estrutura***, em que são realizados o projeto e a implementação dos componentes resultantes do processo de especificação.

Segundo FAVARO (1997) a engenharia do domínio foi dificultada durante alguns anos pela falta de tecnologias adequadas para registrar os resultados da análise de domínio, o que foi conseguido com o surgimento e uso dos *frameworks* orientados a objetos (WIRFS-BROCK e JOHNSON, 1990, MOSER e NIERSTRASZ, 1996) e os padrões de modelagem (GAMMA *et al.*, 1995).

Muitos métodos para a análise de domínio surgiram definindo passos para adquirir, organizar e avaliar as informações. Alguns desse métodos são: o método comum proposto por ARANGO (1994) definido a partir da análise de similaridades de outros métodos; o método FODA (*Feature-Oriented Domain Analysis*) (COHEN, 1994) que propõe o uso de *features* que são características do domínio que sejam visíveis para o usuário, o método proposto por GOMMA e KERSCHBERG (1995) incorporado em um ciclo de vida evolutivo do domínio (GOMMA *et al.*, 1996); o método proposto por JACOBSON *et al.* (1997) que define uma arquitetura de modelos para sistematizar o processo de desenvolvimento baseado em reuso; o método SODA (*Sodália Object-Oriented Domain-Analysis)* (LISSONI e STELLUCI, 1997) proposto pela empresa de telecomunicações Sodália como extensão ao método proposto por JACOBSON *et al.* (1997) para possibilitar a representação da variabilidade de requisitos em uma família de sistemas, e, o método FAST (WEISS e LAI, 1999), uma evolução de Synthesis (CAMPBELL *et al.*, 1990), que procura identificar as similaridades de uma família de produtos (sistemas) pertencentes a um determinado domínio de aplicação e suas variações, definindo uma linha de produção (*product line*) capaz de suportar o desenvolvimento do produto básico e suas diferentes configurações. Todos os métodos propõem a definição e o uso de um ou mais modelos, utilizando diferentes formas de representação, como por exemplo, esquema de facetas, glossários, modelos entidade-relacionamento, taxonomia de funções e de objetos, modelos hirárquicos e diagramas orientados a objetos.

3.1.2 Projetos Orientados a Domínio

Com a disseminação das pesquisas em ADS, diferentes grupos fizeram propostas visando oferecer apoio no desenvolvimento de software considerando características particulares do domínio para o qual os ADS são definidos. Como todo desenvolvimento utilizando ADS, estes trabalhos se preocupam com a definição de processos de desenvolvimento, utilização de modelos e métodos, suporte ao armazenamento de dados e uso de ferramentas. No entanto, o principal enfoque é a orientação a domínio. As

principais linhas de pesquisa referem-se à utilização de arquiteturas específicas para o domínio, abordagem baseada em conhecimento para apoiar o reuso, enfoque no apoio ao entendimento do problema e o uso de definição formal para a documentação. A seguir descrevemos algumas destas pesquisas.

➢ **Ambientes baseados em Arquiteturas Específicas do Domínio**

Arquiteturas de software específicas do domínio (DSSA- *Domain-Specific Software Architecture)* (TRACZ, 1994, TAYLOR *et al.*, 1995) é um projeto de larga escala que utiliza a abordagem de desenvolvimento de software baseado em componentes. DSSA é definido como uma arquitetura de referência acrescentada de um modelo de domínio e requisitos de referência. A ***arquitetura de referência*** é uma arquitetura de software genérica para uma família de aplicações. Esta arquitetura é projetada para ser utilizada por várias aplicações semelhantes dentro de um domínio e depende de um estilo arquitetural específico. O ***modelo de domínio*** é a representação dos elementos do domínio e da relação entre eles. O domínio é considerado um problema ou uma área de tarefa na qual muitas aplicações semelhantes serão desenvolvidas. O principal objetivo do modelo de domínio é padronizar a terminologia do domínio e fornecer a base para descrições padronizadas de problemas específicos a serem resolvidos no domínio. Finalmente, os ***requisitos de referência*** são requisitos comportamentais genéricos para aplicações dentro de um domínio. Requisitos de uma aplicação específica podem ser estabelecidos através da parametrização ou modificação desses requisitos de referência.

Um processo de engenharia do domínio é utilizado para gerar as arquiteturas específicas do domínio (TRACZ *et al.*, 1993, TAYLOR *et al.*, 1995*)* consistindo de cinco fases principais. As três primeiras fases correspondem à análise do domínio, sendo definido o escopo do domínio, os requisitos e conceitos específicos do domínio e as restrições de implementação no que se refere a software (linguagens utilizadas) e hardware (plataformas utilizadas). As duas últimas fases correspondem à definição e projeto da arquitetura propriamente dita, sendo desenvolvidos modelos/arquiteturas do domínio e produzidos componentes reutilizáveis. Com os modelos definidos, uma aplicação pode ser desenvolvida por composição, particularizando os componentes para satisfazer os requisitos do projeto específico.

Diferentes iniciativas tem sido realizadas considerando arquiteturas específicas do domínio, como o DASDE (*DSSA-Application Development Software Environment*) definido para apoiar o desenvolvimento de aplicações no domínio de aviação, mais

especificamente para o gerenciamento e controle distribuído de veículos de artilharia (TERRY *et al.*, 1994); o ambiente para desenvolvimento de sistemas inteligentes adaptativos (como sistemas de robótica e monitoração) (HAYES-ROTH *et al.*, 1995), e DOSAEE (*Domain-Oriented Software Analysis and Engineering Environment*) (SPS, 1996) que apoia a criação, análise e reuso de arquiteturas específicas do domínio.

➢ **Ambientes baseados em Conhecimento para apoio à Reutilização**

Ambientes baseados em conhecimento para apoio à engenharia de software (KBSE - *Knowledge-Based Software Engineering*) (GOMMA *et al.*, 1994, GOMMA *et al.*, 1996) tem como objetivo apoiar o reuso de software nas fases de especificação, projeto e codificação. O ambiente se caracteriza por ser independente do domínio da aplicação podendo ser configurado para qualquer domínio. KBSE usa um ciclo de vida orientado à reutilização para evolução do modelo de domínio e um método de modelagem do domínio desenvolvido no contexto do próprio projeto (GOMMA e KERSCHBERG, 1995). A modelagem do domínio refere-se ao desenvolvimento de requisitos, especificação e arquitetura reutilizável, para uma família de sistemas em um domínio de aplicação. A abordagem baseada em conhecimento refere-se à utilização de uma ferramenta de elicitação de requisitos que possui regras para geração da especificação do sistema a partir do modelo do domínio definido para a família em questão. Este ambiente tem sido usado em diferentes domínios, tais como controle e comando de satélites por estações terrestres da NASA e manufaturas.

➢ **Ambientes baseados em Conhecimento para apoiar o Entendimento do Problema**

No desenvolvimento de sistemas os diferentes participantes do processo tem, frequentemente, que ensinar uns aos outros. A comunicação e colaboração entre todos os participantes são, portanto, essenciais. Os usuários ou especialistas no domínio entendem a prática, ou seja, o que o sistema deve fazer. Os desenvolvedores conhecem a tecnologia, ou seja, como o sistema deve fazer. Dessa forma, o máximo possível de conhecimento deve ser trocado entre os participantes com o objetivo de uma educação mútua e um entendimento compartilhado em relação à tarefa da aplicação a ser desenvolvida. A partir deste problema FISCHER *et al.* (1992a, 1992b) propôs ambientes para a realização de projetos orientados a domínio (DODE - *Domain-Oriented Design Environment*).

DODE (FISCHER, 1994, 1996) são sistemas complexos que apoiam as atividades de projeto (*design*) em domínios específicos. DODE tem como objetivo apoiar as atividades *upstream* (transição do entendimento do problema para a etapa de especificação). Segundo FISCHER (1994), as pesquisas em engenharia de software, em geral, preocupam-se mais com a transição da especificação para a implementação (atividades *downstream*) do que com o entendimento do problema, sendo que estas duas atividades são complementares. Estes ambientes são caracterizados pelos seguintes componentes: abstrações e objetos do domínio capturados em um modelo e disponibilizados em uma paleta (ex: fogão no ambiente para projeto de cozinhas); restrições de projeto definidas em um conjunto de regras (ex: um fogão não deve ser colocado ao lado de uma geladeira); argumentação para as regras de projeto (FISCHER *et al.*, 1993); catálogo de projetos realizados para permitir o estudo baseado em casos e reuso; lista de verificação para guiar a construção do artefato desejado e um componente de simulação para mostrar o comportamento de objetos e casos dentro do domínio.

Uma característica importante do DODE é o aspecto evolutivo do ambiente apoiado pela definição de um processo de construção e uso do ambiente em que o conhecimento é inicialmente modelado e continuamente evoluído e reorganizado (FISCHER *et al.* 1994, 1995, FISCHER, 1996). Este processo considera três fases básicas: a fase de semear, na qual projetistas e desenvolvedores instanciam a arquitetura do DODE para o domínio específico; a fase de crescimento evolutivo, na qual os projetistas acrescentam informação à medida em que usam o ambiente; e a fase de re-semear onde as informações são reorganizadas para serem reutilizadas em outros projetos.

DODE tem sido desenvolvido para diferentes domínios como, por exemplo, o projeto de cozinhas (FISCHER *et al.*, 1993, FISCHER, 1994), projeto de interface com o usuário (FISCHER, 1994), projeto de redes de computadores (FISCHER *et al.*, 1992a, FISCHER, 1996), projeto de interface com o usuário baseada em telefone (voz) (FISCHER *et al.*, 1995, REPENNING e SUMMER, 1995), projeto de desenhos gráficos (EISENBERG e FISCHER, 1994), projetos para autoria de informações multimídia (NAKAKOJI *et al.*, 1996) e ambientes de aprendizado baseados em jogos (EDEN *et al.*, 1996).

➢ **Ambientes Baseados em Formalização**

Ambientes baseados em formalização preocupam-se com a conversão do conhecimento informal fornecido pelo usuário em especificações formais que possam ser processadas pelo computador. Os trabalhos nessa área vão de simples projetos de pesquisa (SHIPMAN III e MCCALL, 1994) até projetos em larga escala como é o caso do AMPHION (LOWRY e BAALEN, 1997) desenvolvido para auxiliar no desenvolvimento de sistemas de projetos espaciais da NASA.

O AMPHION é um ambiente baseado em conhecimento que capacita o usuário a definir um problema de uma forma abstrata, utilizando vocabulário específico do domínio e recursos gráficos e que, automaticamente, gera um programa que consiste de chamadas para bibliotecas de componentes e que implementa a solução para a especificação do problema. Para isso, são utilizadas teorias declarativas do domínio, especificações formais e síntese dedutiva de programas. AMPHION provê uma arquitetura independente do domínio que pode ser instanciada para qualquer domínio desejado, já tendo sido experimentado com os domínios de geometria do sistema solar, dinâmica de fluidos e navegação espacial.

De forma geral todas essas propostas de ambientes preocupam-se com o reuso, a automação de especificações e o entendimento do domínio. Esse último aspecto é bastante relevante no contexto dessa tese por enfocar a necessidade do entendimento do domínio em si, antes da sua utilização para desenvolvimento de sistemas ou para verificação de sistemas existentes. FISCHER (1994, 1996) propõe-se a apoiar esse entendimento do domínio disponibilizando o conhecimento pertinente em representações formais (regras de produção utilizadas em inteligência artificial) para apoiar a atividade de projeto. Conhecimento do domínio, organizado sobre esse enfoque, pode apoiar a interação dos desenvolvedores com o domínio do problema em vez de com componentes implementacionais. Esse conhecimento refere-se ao segundo nível de reutilização apresentado anteriormente e que, portanto, se modelado e construído é passível de reutilização.

Buscando o reuso de bases de conhecimento em sistemas baseados em conhecimento, pesquisadores da área de Inteligência Artificial (IA) utilizam Ontologias, um conceito vindo da Filosofia, para modelar e organizar esse conhecimento que é representado em uma linguagem formal. Ontologias têm sido amplamente exploradas pela comunidade de IA como um fator chave para garantir o reuso e o

compartilhamento do conhecimento. Nesse trabalho, Ontologia é utilizada como a principal fonte para organização e recuperação de informação orientada a domínio fornecendo apoio no entendimento do domínio e do problema e, consequentemente, tornando-se útil no desenvolvimento de software. Dessa forma, dedicamos o resto desse capitulo a descrever os conceitos, características e benefícios dessa tecnologia.

3.2 Ontologia

O termo ontologia vem da filosofia significando uma explicação sistemática da existência, ou seja, o que significa "existir". O termo tem sido amplamente utilizado em Inteligência Artificial nos últimos anos e surgiu da percepção que de nada valia a preocupação com mecanismos de representação do conhecimento (regras, frames, redes neurais, lógica *fuzzy, etc.*) se não existisse um bom conteúdo e organização sobre o conhecimento do domínio em que se deseja trabalhar (CHANDRASEKARAN *et al*, 1999). Diferentes definições para o termo ontologia tem sido propostas. Algumas delas são:

(a) ontologia é uma especificação explícita de uma conceituação (GRUBER, 1993, 1995)

(b) ontologia é uma descrição parcial e explícita de uma conceituação (GUARINO e GIARRETA, 1995)

(c) ontologia é uma teoria sobre um domínio que especifica um vocabulário de entidades, classes, propriedades, predicados e funções; e um conjunto de relações que necessariamente amarram esses vocabulário (FIKES e FARQUHAR, 1999).

A primeira definição é importante por servir de marco para o entendimento de ontologias e das definições que se seguiram. Segundo GRUBER, em IA o que existe é o que pode ser representado. Uma ontologia consiste, portanto, de um vocabulário representacional com definições precisas do significado dos termos desse vocabulário mais um conjunto formal de axiomas que restringe a interpretação e o uso desses termos. Os axiomas especificam a semântica de termos na ontologia e restrições sobre sua interpretação. Geralmente, são inicialmente escritos em linguagem natural e, posteriormente, formalizado na linguagem de formalização específica. Os axiomas definem restrições impostas pela forma de estruturação dos conceitos (axiomas epistemológicos), e, principalmente, descrevem restrições de significação impostas no domínio (axiomas ontológicos). Segundo FALBO (1998) além desses axiomas, existem

os axiomas de consolidação responsáveis por representar a coerência, ou integridade, das informações existentes.

GUARINO e GIARRETA (1995) propuseram "enfraquecer" a definição de GRUBER substituindo "especificação" por "descrição parcial" (definição b) pois consideram que o grau de especificação de uma conceituação de uma linguagem utilizada para uma base de conhecimento depende do propósito desejado para a ontologia. Um dos propósitos é ser um sinônimo de teoria ontológica, ou seja, um conjunto de axiomas (fórmulas) que são consideradas sempre verdadeiras independente dos valores particulares para os mesmos, sendo, portanto, compartilhável entre diferentes agentes. Nesse sentido a ontologia se aproxima de uma conceituação independente e que pode ser utilizada para estabelecer o compartilhamento de uma base de conhecimento particular. Um segundo propósito é da ontologia ser um conjunto de compromissos ontológicos (restrições rígidas) no qual é desenvolvida de acordo com inferências particulares projetadas para serem compartilhadas por usuários que concordam previamente com a conceituação especificada.

Uma discussão bastante comum quando se fala sobre ontologia é a questão de que o nível de conhecimento da conceituação é dependente do domínio e tarefa particulares para a qual é projetada (VAN HEIJST *et al.*, 1997). A justificativa para essa questão é o chamado problema da interação, no qual é definido que a representação do conhecimento é fortemente afetada pela natureza do problema e a estratégia de inferência utilizada para resolver o problema. VAN HEIJST *et al.* (1997) defendem que a tarefa da aplicação determina qual o conhecimento que deve ser codificado. Além disso o conhecimento deve ser codificado de forma que a estratégia de inferência utilizada possa funcionar de forma eficiente. Dessa forma, os autores acreditam que esta interação deve estar explicitamente definida.

GUARINO (1997a) critica essa definição afirmando que uma parte específica do conhecimento pode, de certa forma, ser relevante para uma tarefa mas que isso não significa que esse conhecimento é peculiar e particular para esta tarefa. Além disso, afirma que a reutilização entre várias tarefas e métodos deve sempre ser buscada e que isso implica ser cada vez mais independente da tarefa. Recentemente (CHANDRASEKARAN *et al.*, 1999) defenderam essa posição argumentando que apesar de ontologias serem independentes de tarefas, quando se define uma ontologia é natural enfocar alguns aspectos da realidade em detrimento de outros, dependendo do

que se deseja como uso dessa ontologia. No entanto, isso não significa que esses conceitos são dependentes da tarefa.

Finalmente, a última definição é importante por procurar englobar todos os conceitos definidos pela ontologia sendo, portanto, mais completa. Esses conceitos são explicitamente definidos e representados de forma que possam ser compartilhados e possuem restrições expressas através de axiomas que explicitamente restringem os significados dos termos (como descrito anteriormente). Esses conceitos podem incluir estruturas conceituais para modelagem do conhecimento do domínio, protocolos específicos do conteúdo para comunicação e inter-operação entre agentes e compromissos sobre uma representação de teorias de um domínio particular (USCHOLD e GRUNINGER, 1996). Essa é, portanto, a definição adotada neste trabalho, mas considerando os aspectos de que essa ontologia é uma descrição parcial projetada para ser compartilhada dentro de uma comunidade que concorda com a sua definição, e para o fim específico de desenvolvimento de software em um dado domínio.

3.2.1 Classificação de Ontologias

Assim como existem várias definições sobre ontologias, existem também diversas classificações. De acordo com VAN HEIJST *et al.* (1997) e VAN HEIJST (1995), uma ontologia pode ser classificada de acordo com duas dimensões: a quantidade e o tipo de estrutura da conceituação, e o assunto da conceituação.

Na primeira dimensão, são definidas três categorias de ontologias:

- ***ontologias terminológicas*** que especificam os termos que são usados para representar o conhecimento no domínio de discurso;
- ***ontologias de informação*** que especificam a estrutura de registro de um banco de dados, como por exemplo a esquematização ("schemata") de banco de dados, e,
- ***ontologias de modelagem do conhecimento,*** que especificam a conceituação do conhecimento. Comparadas com ontologias de informação, estas ontologias têm sempre uma rica estrutura interna e estão, frequentemente, ajustadas a um uso específico do conhecimento que descrevem.

Na segunda dimensão, pode-se distinguir quatro categorias:

- ***ontologias genéricas***, que definem conceitos genéricos entre as várias áreas, por exemplo, conceitos como estado, evento, processo, ação, componente, etc;

- ***ontologias de domínio,*** que expressam conceituações específicas para domínios particulares, colocando restrições na estrutura e conteúdo do domínio do conhecimento responsável por descrever os fatos de um determinado domínio sendo vistas como uma especialização dos conceitos da ontologia genérica;
- ***ontologias de aplicação,*** que contém todas as definições necessárias para modelar o conhecimento requerido de um domínio específico, sendo uma combinação dos conceitos tirados das ontologias de domínio e das ontologias genéricas e que pode conter extensões para a tarefa e método específico, e,
- ***ontologias de representação*** que explicam as conceituações utilizadas nos formalismos de representação do conhecimento, sendo neutras em relação às entidades do mundo (ou seja, o domínio). Ontologias de domínio e ontologias genéricas são descritas usando as primitivas providas pelas ontologias de representação.

Considerando que a primeira dimensão definida por vanHeijst não é clara e que não existe razão para diferenciar ontologias com base na quantidade e tipo de estrutura de sua conceituação, GUARINO (1997b) propôs classificar ontologias de acordo com as seguintes dimensões: o ***nível de detalhe*** da ontologia e *o* ***nível de dependência*** da ontologia em relação a uma tarefa específica ou ponto de vista.

Na primeira dimensão, uma ontologia muito detalhada consegue especificar o significado pretendido de um vocabulário (conceituação), e portanto, pode ser usada para estabelecer consenso sobre o compartilhamento desse vocabulário ou de uma base de conhecimento que usa esse vocabulário. Por outro lado, uma ontologia muito simples pode ser desenvolvida de acordo com características particulares de forma a ser compartilhada entre usuários que, previamente, concordam com a conceituação.

Na segunda dimensão, GUARINO (1998) considera ontologias de alto-nível (*top-level*), ontologias de domínio, ontologias de aplicação e ontologias de tarefas (Figura 3.1).

As três primeiras correspondem respectivamente as ontologias genéricas, de domínio e aplicação definidas por vanHeijst sendo enfatizado que a ontologia de domínio refere-se ao vocabulário de um domínio genérico como medicina ou automóveis. As ontologias de tarefas descrevem uma tarefa ou atividade genérica (como por exemplo diagnóstico) especializando os termos introduzidos na ontologia de alto-nível. Finalmente, Guarino considera as ontologias de representação com um tipo separado por serem uma ontologia de meta-nível que descreve uma classificação das

primitivas sobre a linguagem de representação (como conceitos, atributos, relações, etc.).

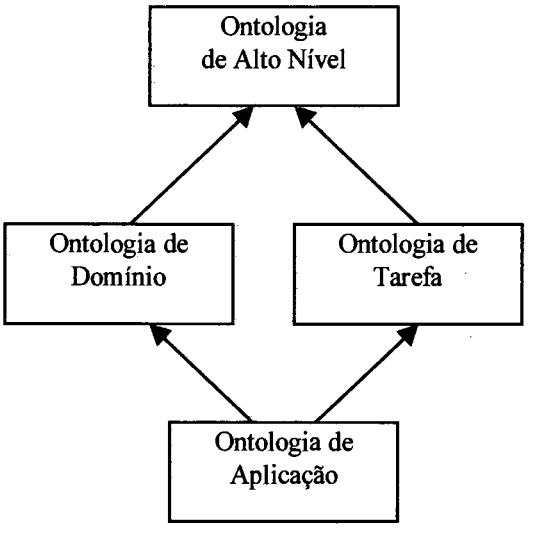

Figura 3.1 – Tipos de Ontologias (GUARINO, 1998)

Uma classificação numa linha diferente das anteriores, é apresentada por USCHOLD (1996). Nesta classificação uma ontologia pode variar em relação: *(i)* ao propósito para o qual é definida; *(ii)* quanto a natureza do assunto que a ontologia está caracterizando, e, *(iii)* quanto ao grau de formalidade com que o vocabulário é criado e seu significado é definido. Esta variação, implicitamente, dá origem a diferentes tipos de ontologias. No que se refere ao propósito uma ontologia pode ser definida para diferentes utilizações, o que é discutido em detalhes na próxima seção. A natureza do assunto classifica ontologias do domínio, de tarefas e de representação definidas anteriormente. No que se refere à formalidade, uma ontologia pode ser de três tipos:

- ***altamente informal***, expressa em linguagem natural (como glossários),
- ***estruturada informal***, expressa em uma forma restrita e estruturada de linguagem natural,
- ***semi-formal***, expressa formalmente em uma linguagem definida, e,
- ***rigorosamente formal***, em que os termos são meticulosamente definidos com semântica formal, teoremas e provas de propriedades como, por exemplo, completude.

Neste trabalho é adotada a classificação de GUARINO (1998) por separar, claramente, o domínio das tarefas em tipos de ontologias distintas mostrando que a combinação das duas são particulares para uma aplicação. A classificação proposta por USCHOLD (1996), no entanto, acaba sendo natural quando se define uma ontologia já

que é projetada para uma ou mais utilizações específicas, dentro de uma área e alguma linguagem de representação que combina linguagem natural e formal.

3.2.2 Utilização de Ontologias

Ontologias tem sido desenvolvidas em diferentes domínios e com diferentes propósitos (FRIDMAN e HAFNER, 1997, LÓPEZ *et al.*, 1999, VALENTE *et al.*, 1999, RECTOR e HORROCCKS, 1997, SMITH e BECKER, 1997). De forma geral, a maioria das ontologias buscam alguma forma de reuso, variando no que se refere à utilização com um pequeno grupo de pessoas e dentro do contexto de uma variedade de aplicações, ou se o objetivo é o reuso numa grande comunidade (USCHOLD e GRUNINGER, 1996). Alguns projetos usam ontologias, principalmente, para estruturação da base de conhecimento. Outros propõem ontologias para serem utilizadas como parte da base de conhecimento.

De forma geral, o uso de ontologias se classifica em uma das seguintes categorias (USCHOLD e GRUNINGER, 1996): *(i)* comunicação entre as pessoas e organização, *(ii)* interoperabilidade entre sistemas, e, *(iii)* benefícios na engenharia de sistemas baseados em conhecimento ou não.

No que diz respeito à ***comunicação***, ontologias reduzem confusões terminológicas e conceituais dentro da organização. Dessa forma, ontologias capacitam o entendimento compartilhado e a comunicação entre pessoas com diferentes necessidades e pontos de vista particulares sobre o seu trabalho. No contexto de sistemas de software integrados de grande escala, diferentes pessoas precisam ter um entendimento comum do sistema e seus objetivos. Através de ontologias pode ser criado um modelo normativo do sistema criando uma semântica para o mesmo. Ontologias podem, ainda, serem usadas para criar uma rede das relações chaves dentro do sistema permitindo às pessoas explorarem e navegarem através dessa rede para entender os relacionamentos internos do sistema.

Interoperabilidade refere-se à necessidade de diferentes usuários trocarem dados ou utilizarem diferentes ferramentas. Ontologias podem ser usadas como uma inter-lingua para apoiar a tradução entre diferentes linguagens e representações. Dessa forma para n linguagens, em vez da necessidade de criar um tradutor para cada par de linguagens (o que significa uma grandeza da ordem de n^2 tradutores) seria, apenas, necessário um tradutor da linguagem para a ontologia, que funciona como interlingua, e desta para a linguagem (ou seja, $2n$ tradutores). Além disso, para que ferramentas e repositórios de dados possam ser compartilhados e reutilizados deve-se levar em consideração a

natureza da relação entre os usuários (se será uma interoperabilidade entre sistemas internos em uma organização ou com sistemas externos), a necessidade de integrar ontologias em diferentes domínios para apoiar alguma tarefa específica e, ainda, a necessidade de integrar diferentes ontologias dentro de um mesmo domínio. Essa é uma área de difícil pesquisa com alguns experimentos realizados no contexto de sistemas baseados em conhecimento (SWARTOUT *et al.*, 1994, NETCHES, 1994).

Ontologias podem ser benéficas na especificação, confiabilidade e reutilização durante a ***engenharia de sistemas***. Na *especificação*, ontologias podem ser utilizadas para promover o entendimento compartilhado de um problema ou tarefa, facilitando o processo de identificação dos requisitos do sistema e o entendimento das relações entre os componentes do sistema, o que é extremamente importante em sistemas que envolvem uma equipe trabalhando em diferentes domínios. Ontologias podem prover, ainda, uma especificação declarativa de um software que permite aos desenvolvedores raciocinarem quais são os objetivos do sistema em vez de como é a sua funcionalidade. No que se refere a *confiabilidade*, ontologias podem servir para verificação manual do projeto em relação à sua especificação. Ontologias formais permitem uma verificação semi-automatizada do sistema de acordo com suas especificações declarativas e podem ser utilizadas para tornar explícitos os acordos entre diferentes componentes de software facilitando sua integração. Ontologias formais tem sido utilizadas, ainda, para verificação automática de erros em especificações executáveis no que se refere a descrições do domínio (KALFOGLOU e ROBERTSON, 1999). Finalmente, para serem efetivas, ontologias devem apoiar a *reutilização* de forma que se possa importar e exportar módulos entre diferentes sistemas. Caracterizando classes de domínios e tarefas dentro destes domínios, ontologias prover uma estrutura de trabalho para determinar que aspectos de uma ontologia são reutilizáveis entre diferentes domínios e tarefas. Isso significa que ontologias podem prover uma biblioteca para reutilização de objetos para domínios e problemas de modelagem. Trabalhos, nesse sentido, referem-se à construção de uma biblioteca de ontologias que possam ser reutilizadas e adaptadas para diferentes classes de problemas e ambientes (NETCHES *et al.*, 1991).

No que se refere ao desenvolvimento de sistemas baseados em conhecimento, uma ontologia pode ser vista como uma interface entre a base de conhecimento e o mundo externo, tendo quatro utilizações gerais (ABU-HANNA e JANSWEIJER, 1994): *compartilhamento*, pois para que as aplicações possam compartilhar bases de conhecimento é necessário concordarem na mesma definição de termos o que é

determinado pela ontologia; *aquisição de conhecimento*, no sentido de que como a ontologia inclui terminologias específicas para a aplicação, podem ser utilizadas em ferramentas de aquisição que trabalham diretamente com especialistas do domínio, o que efetivamente evita erros no conhecimento adquirido e o conteúdo do conhecimento pode ser aplicado; *organização do conhecimento*, permitindo que as bases de conhecimento sejam melhores acessadas, modificadas e reutilizadas e no *processo de raciocínio* já que a ontologia pode ser utilizada junto com a base de conhecimento como um modelo que uma tarefa pode considerar durante sua execução. Além disso, ontologias são essenciais para sistemas de gerência de conhecimento (O'LEARY, 1998) que buscam o controle formal do conhecimento para facilitar criação, acesso e reuso do conhecimento.

No entanto, uma das principais e mais abordadas funções das ontologias refere-se ao compartilhamento e reutilização da base de conhecimento (GRUBER, 1991, NECHES *et al.*, 1991, GRUBER, 1995, VALENTE, 1995, VALENTE *et al.* 1999). O objetivo é evitar o esforço e o custo elevado no desenvolvimento de sistemas, que normalmente requerem a construção de novas bases de conhecimento.

Aspectos de compartilhamento e reuso do conhecimento serão melhor detalhados no final desse capítulo.

3.2.3 Engenharia de Ontologias

A engenharia de ontologias é um processo que envolve disciplina e organização, sendo considerada mais do que uma ciência, uma arte, pois não existe nenhuma definição e padronização de ciclos de vida, métodos e técnicas que orientem o desenvolvimento de ontologias (GÓMEZ-PÉREZ, 1996, LÓPEZ *et al.*, 1999).

A engenharia de ontologias é uma tarefa que deve ser cuidadosa e bem conduzida através de princípios adequados que permitam sua melhor avaliação. Se as definições ontológicas e o ambiente de software não tiverem sido suficientemente avaliados, a comunicação entre as pessoas pode não ser conseguida da melhor forma, porque as ontologias provêm um canal através do qual questões são resolvidas (GÓMEZ-PÉREZ, 1995).

Semelhante ao desenvolvimento de sistemas, a definição de ontologias deve considerar critérios específicos e um processo de avaliação que envolva verificação e validação. A Tabela 3.1 apresenta alguns dos critérios mais importantes citados na literatura para o projeto e avaliação de ontologias.

Tabela 3.1 – Critérios para o projeto e avaliação de ontologias

Critério	Descrição
Clareza (GRUBER, 1995, FOX *et al.*, 1995)	Uma ontologia deve comunicar efetivamente o significado planejado dos termos definidos, as definições devem ser objetivas, documentadas em linguagem natural e independentes do contexto social ou computacional o que é conseguido através de formalismos.
Consistência (GRUBER, 1995, FOX *et al.*, 1995, GÓMEX-PÉREZ *et al.*, 1995)	Uma ontologia deve garantir consistência com sua definição, tanto no que se refere à especificação dos axiomas lógicos quanto aos conceitos informais, como os comentários em linguagem natural. Uma ontologia é, semanticamente, consistente se e somente se suas definições são semanticamente consistentes e uma dada definição é semanticamente consistente se e somente se *(i) os* significados tanto das definições formais quanto informais são consistentes com o mundo real e consistentes umas com a outras, e, *(ii)* se as definições não são sentenças contraditórias que podem ser inferidas usando outras definições e axiomas que pertençam ou não à mesma ontologia.
Extensibilidade (GRUBER, 1995, FOX *et al.*, 1995, GÓMEZ-PÉREZ *et al.*, 1995)	Uma ontologia deve poder ser estendida para englobar novos conceitos com base no vocabulário existente. Garantir a capacidade de extensão refere-se ao esforço necessário para se incluir novas definições sem alterar o conjunto de propriedades já avaliadas na ontologia.
Minímo "bias" de codificação (GRUBER, 1995, FOX *et al.*, 1995)	Uma conceituação deve ser especificada sem a dependência de uma codificação simbólica específica, procurando não escolher uma representação somente por causa de uma determinada implementação ou codificação.
Requerer um conjunto mínimo de compromissos ontológicos (GRUBER, 1995, FOX *et al.*, 1995)	Uma ontologia deve ser suficiente para apoiar as atividades pretendidas de compartilhamento do conhecimento. Isto é conseguido através de uma especificação de uma teoria, definição somente de termos que são essenciais para comunicação do conhecimento consistente com esta teoria, e definição do mínimo de restrições possíveis sobre o mundo que está sendo modelado.
Generalidade (FOX *et al* 1995)	Uma ontologia deve ser capaz de ser compartilhada entre diferente atividades como projeto e análise.
Suporte(FOX *et al* 1995)	Uma ontologia deve apoiar o desenvolvimento de grandes aplicações
Completude (GÓMEZ-PÉREZ *et al.*, 1995)	A completude de uma definição em uma ontologia depende do nível de granularidade da ontologia. Garantir a completude de uma definição implica em garantir a completude tanto da definição formal quanto da informal. Para garantir que uma definição formal é completa, deve-se determinar se a definição está de acordo com seu critério estrutural para uma definição completa, se o domínio e um grupo de relações e funções estão precisamente e exatamente bem delimitados; se a generalização/especialização de uma determinada classe representa exata e precisamente uma determinada classe do mundo real; e, finalmente, verificar a existência de um conjunto completo de atributos em cada definição. Uma definição informal escrita em linguagem natural é completa se e somente se expressa o mesmo conhecimento pretendido em uma definição formal.
Concisão (GÓMEZ-PÉREZ *et al.*, 1995)	Uma ontologia deve capturar apenas a informação necessária e suas definições devem ser concisas. Uma definição em uma ontologia é concisa se são evitadas, o quando possível, redundâncias formais e informais. (Explicação em linguagem natural das explicações e exemplos não são considerados conhecimento redundante das definições formais.)
Robustez (GÓMEZ-PÉREZ *et al*, 1995)	Uma ontologia deve ser robusta de forma que pequenas mudanças não afetem o conjunto de definições já avaliadas.

O objetivo da avaliação de uma ontologia é detectar a ausência de propriedades nas definições. A avaliação pode ser dividida em verificação e validação (GÓMEZ-PÉREZ et al, 1995).

Verificação significa garantir a correção da ontologia, seu ambiente de software associado e documentação. Os passos dessa verificação incluem: *verificação da estrutura ou arquitetura da ontologia*, cujo objetivo é investigar se as definições estão sendo construídas seguindo os critérios de projeto do ambiente na qual estão incluídas (por exemplo, Ontolingua); *verificar a sintaxe das definições*, para detectar o mais cedo possível estruturas sintaticamente incorretas e/ou palavras chaves erradas (sem significado) em definições formais, palavras chaves sem documentação, estrutura das definições formais, ausência de "loops" entre as definições, etc.; e, *verificar o conteúdo das definições*, para detectar o que a ontologia define, não define ou define incorretamente, e o que pode ser inferido, não pode ser inferido ou é inferido incorretamente, com o objetivo de identificar a falta de conhecimento e erros nas definições, o que significa verificar a ontologia de acordo com os critérios estabelecidos.

A validação garante que a ontologia, seu ambiente de software e documentação correspondem ao que se propõem. Validar uma ontologia significa, então, garantir que as definições da ontologia realmente definem o mundo real para o qual esta foi projetada, se as definições são necessárias e suficientes para representar as tarefas e suas soluções por diferentes usuários.

3.2.3.1 Métodos e Linguagens para Desenvolvimento de Ontologias

As propostas de métodos apresentadas na literatura são, normalmente, decorrentes da necessidade de definição de ontologias em algum projeto específico. Cada grupo segue um conjunto de princípios, critérios de projeto e fases para o desenvolvimento de ontologias. As principais propostas apresentadas na literatura são mostradas na Tabela 3.2.

Comparando as três propostas percebe-se que embora apresentem fases de desenvolvimento diferentes, o desenvolvimento de uma ontologia considera basicamente os seguintes objetivos em cada fase:

- ***definição do propósito***, ou seja, determinação da motivação para construção de uma ontologia, definindo qual será sua utilização (compartilhamento, reutilização, organização, estruturação do conhecimento)

- ***construção***, onde as definições da ontologia são capturadas, formalizadas em alguma linguagem específica e integradas com outras ontologias existentes
- ***avaliação*** da ontologia definida de acordo com os requisitos especificados; e,
- ***documentação*** dos conceitos e axiomas identificados.

Documentação e avaliação são, geralmente, realizadas em todas as fases. Outro aspecto importante a observar é que em cada uma dessas propostas pode ser observada uma característica básica (última coluna da Tabela 3.2).

Tabela 3.2 – Metodologias para Construção de Ontologias

Proposta	Projeto	Métodos de Desenvolvimento		Característica básica
		Fases	Objetivo específico	
USCHOLD e KING (1995)		Identificação do propósito	Definição do propósito	Elicitação
		Construção	Construção	
		Avaliação	Avaliação	
		Documentação	Documentação	
GRUNINGER e FOX (1995)	TOVE (modelagem de empresa)	Cenários de motivação	Definição do propósito	Formalização
		Questões informais de competência	Definição do propósito	
		Lógica de 1ª Ordem: terminologia	Construção	
		Questões formais de competência	Construção	
		Lógica de 1a ordem: axiomas	Construção	
		Teoremas de completude	Construção	
GÓMEZ-PÉREZ *et al.* (1996) e FERNÁNDEZ *et al.* (1997) LOPEZ *et al.* (1999)	Elementos químicos	Especificação	Definição do propósito	Documentação
		Conceituação	Construção	
		Formalização	Construção	
		Integração	Construção	
		Implementação	Construção	
		Manutenção	Construção	

A proposta do USCHOLD e KING (1995) tem como característica básica uma definição de como deve ser feita a captura da ontologia. Inicialmente, deve ser feita uma *definição do escopo*, através, de uma reunião de "brainstorming" para levantar todos os termos e frases potencialmente relevantes e do agrupamento desses termos nas áreas de trabalhos, que vão sendo trabalhadas em ordem começando pelas áreas que tem maior sobreposição semântica (quando houver) e identificando, para cada uma das áreas, os termos básicos, específicos e abstratos, nessa ordem. Em seguida, é feita uma *produção de definições*, na qual todas as definições em todas as áreas de trabalhos são completadas, o que requer um considerável grau de esforço para concordância nas definições e termos para os conceitos, tendo-se que manipular cuidadosamente com os

termos ambíguos que normalmente devem ser eliminados. Além disso, devem ser feitas definições textuais em linguagem natural para tudo o que foi definido, deve ser garantida a consistência com termos já em uso, indicados os relacionamentos entre os termos e dados exemplos onde apropriado. Deve-se, então, fazer uma *revisão* crítica das definições mantendo um histórico do conjunto de alterações realizadas, e, finalmente é definida uma meta-ontologia usando as definições em linguagem natural como uma especificação de requisitos implícita que sirva de base para o estágio de codificação.

A característica básica do projeto de GRUNINGER e FOX (1995) é o fato de, através de uma formalização mais rigorosa, orientar o projeto e avaliação de ontologias. Nesse sentido, o propósito da ontologia é definido inicialmente em questões informais de competência que são posteriormente formalizadas, especificadas em sentenças de primeira ordem, criando um vínculo com os axiomas da terminologia. São definidos, ainda, os teoremas de completude, nos quais é especificado, em sentenças de primeira ordem, o conjunto de condições sob o qual as soluções para o problema são completas.

GOMÉZ-PÉREZ *et al.* (1996), FERNANDEZ *et al.* (1997) e LÓPEZ *et al.* (1999) procuraram enfatizar a necessidade de documentação de todas as atividades de desenvolvimento de uma ontologia e um conjunto de atividades organizadas em um ciclo de vida que determine em que momento são realizadas as atividades conforme mostra a Figura 3.2. Antes de construir a ontologia devem ser *planejadas* todas as tarefas a serem realizadas, como serão organizadas, quanto tempo será necessário e que recursos que devem ser utilizados. O primeiro passo na construção da ontologia é então a definição do propósito e escopo (*especificação*), seguido de uma *conceituação* do domínio considerado. O modelo conceitual produzido é *formalizado* em uma representação lógica de descrição ou uma representação orientada a "frame". Deve-se *integrar* a ontologia, o máximo possível, com ontologias já existentes, *implementar* na linguagem formal escolhida e, em qualquer momento, poder incluir ou modificar definições na ontologia (*manutenção*). Todos os estágios devem ser apoiados pelas atividades de aquisição do conhecimento, documentação e avaliação. Finalmente, FERNÁNDEZ *et al* (1997), LÖPEZ *et al.* (1999) propõem que o ciclo de vida que melhor se adequa para o processo de desenvolvimento de ontologias é o ciclo de vida evolutivo dado que a ontologia cresce de acordo com a necessidade modificando, adicionando ou removendo definições em qualquer momento.

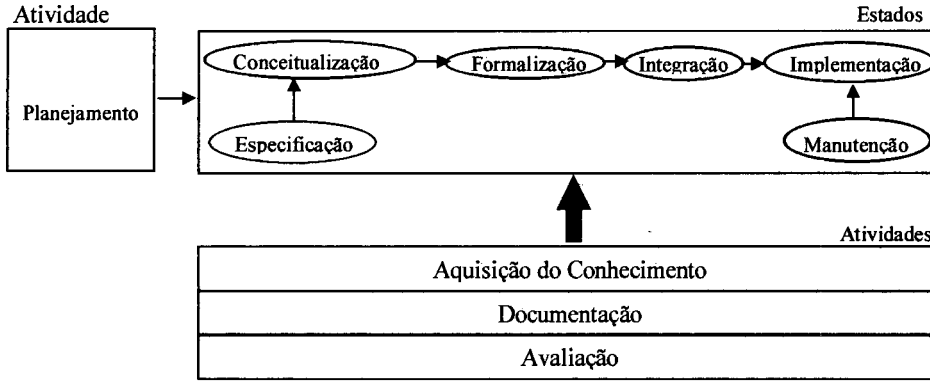

Figura 3.2 - Estados e Atividades (FERNÁNDEZ *et al.*, 1997)

Vários documentos são produzidos desde uma especificação de requisitos da ontologia a ser elaborada até o detalhamento completo da ontologia envolvendo um conjunto de documentos como por exemplo: glossário de termos (conceitos, instâncias, verbos e propriedades), dicionário de dados, árvores de classificação de conceitos e atributos, e, tabelas de descrição de constantes e fórmulas.

As linguagens de representação são um ponto importante para operacionalizar a ontologia, ou seja, permitir a sua efetiva utilização. Segundo VALENTE (1995) qualquer linguagem de representação do conhecimento formal, ou mesmo informal, pode ser usada para expressar ontologias. No entanto, segundo GUARINO (1995) uma linguagem formal só é adequada ontologicamente se, no nível sintático, possuir granularidade e capacidade suficientes para expressar os postulados de significação de suas próprias primitivas ou se, no nível semântico, for possível dar uma interpretação lógica formal para suas primitivas básicas.

Existem muitas linguagens candidatas à representação de ontologias. Algumas das mais citadas são:

- Lógica de Primeira Ordem - muito utilizada para representar ontologias, por ser uma linguagem geral, bem conhecida, expressiva e que adiciona poucos compromissos ontológicos (VALENTE, 1995);
- KIF ("Knowledge Interchange Format") - linguagem formal construída para trabalhar como uma "interlíngua", isto é, um meio para trocar conhecimento entre bases construídas em diferentes linguagens (GRUBER, 1992);
- Ontolingua - sistema que provê um mecanismo para escrever ontologias em um formato canônico (e, portanto, formal), baseado na linguagem KIF com extensões para se definir classe e relações, e para organizar o conhecimento em uma hierarquia centrada em objetos com herança (GRUBER, 1992, 1993);

- CML ("Conceptual Modelling Language") - uma linguagem semi-formal (proposta como um formalismo de representação dentro da metodologia CommomKADS) através da qual uma ontologia é definida através da especificação de conceitos, atributos, expressões, estruturas e relações, utilizando, ainda, uma representação gráfica (SCHREIBER *et al.*, 1994); e,
- LINGO (Linguagem Gráfica para Ontologias) – uma linguagem gráfica que tem sua representação formal explicitamente definida em lógica de 1ª Ordem (FALBO, 1998).

3.2.4 Ontologia e Base de Conhecimento

Ontologia e base de conhecimento são muitas vezes referidas como tendo mesmo significado porque ambas representam e capturam conhecimento sobre conceitos e suas relações dentro de uma linguagem para um domínio concreto. No entanto, algumas diferenças podem ser observadas (GÓMEZ-PÉREZ *et al.*, 1995). A generalidade da informação é a diferença mais importante entre ontologias e base de conhecimento. As definições de ontologias são mais gerais do que o conhecimento de uma base de conhecimento, de forma que estas podem ser compartilhadas entre diferentes sistemas, devendo ser independentes do sistema que irá compartilhar ou reutilizar. Além disso, ontologias geralmente não tem métodos de raciocínio que processam sobre suas definições como é o caso da base de conhecimento. Outro aspecto é que a linguagem através da qual ontologia é mais expressiva, declarativa, independente do domínio e semanticamente bem definida do que as tecnologias de representação do conhecimento.

Segundo GUARINO (1997b) a diferença entre ontologia e base de conhecimento está relacionada ao propósito de uma ontologia que é uma base de conhecimento particular descrevendo fatos assumidos sempre como verdade por uma comunidade de usuários em virtude de um significado em comum acordo sobre o vocabulário utilizado. Uma base de conhecimento genérica, no entanto, pode também descrever fatos e assertivas relacionadas a um estado particular de valores. Dentro de uma base de conhecimento genérica pode ser distinguido dois componentes: uma ontologia que contém informação independente do estado de valores e o núcleo da base de conhecimento contendo informação dependente do estado de valores.

Um forma de diferenciar ontologias e base de conhecimento é considerar, por exemplo, um tipo de ontologia e o conhecimento que estará na base de conhecimento. Assim, no caso de aplicações, a afirmação de que existe relação de causa entre estados e

que esta relação não é simétrica (*se A causa B, B não pode causar A*) faz parte da ontologia de aplicação e, no entanto, a afirmação de que isquemia causa necrose faz parte do conhecimento da aplicação. Da mesma forma, o conhecimento do domínio refere-se a estados particulares dentro do domínio específico (por exemplo, dor no peito é uma manifestação de arterosclerose) e ontologia do domínio impõe restrições na estrutura de expressões do conhecimento do domínio (para o exemplo, doenças têm achados como manifestações) (VAN HEIJST, 1995). Ou seja, uma ontologia provê um conjunto de conceitos e termos para descrever algum domínio enquanto a base de conhecimento usa estes termos para representar o que é verdadeiro sobre o mundo real ou hipotético (SWARTOUT e TATE, 1999). Segundo MUSEN (1998) uma base de conhecimento pode ser vista como uma instanciação de uma ontologia, que significa preencher as descrições dos conceitos da ontologia detalhando a aplicação que estiver sendo construída.

Compartilhamento e reuso são, geralmente, discutidos em conjunto por serem muito semelhantes e por, muitas vezes, não estarem claras as diferenças entre esses conceitos (MIZOGUCHI, 1994). Compartilhar conhecimento pode ser definido como o uso por múltiplos agentes de software de uma parte ou de todo o conhecimento de uma certa base de conhecimento construída por outro agente de software. Reutilizar conhecimento seria o uso por um agente de software da base de conhecimento (em parte ou no todo) construída por outro agente com fins diferentes do seu. Existem duas formas básicas de compartilhamento e reuso: através do uso direto do conhecimento de uma base de conhecimento ou, indiretamente, através da comunicação entre agentes de software no qual um agente solicita a outro para resolver determinado problema de acordo com protocolos de comunicação pré-estabelecidos.

Segundo LENAT (1995a) o compartilhamento do conhecimento em programas pode ser visto de forma simples como o compartilhamento de termos e dos significados da maioria desses termos; a capacidade de discutir e resolver conflitos quando eles ocorrem e o uso de um domínio de teorias do mundo real de senso comum.

Existem basicamente três tipos de reutilização e compartilhamento: através da cópia de um módulo de uma biblioteca e sua reimplementação, através da inclusão da especificação fonte em fase de projeto, ou através da chamada de módulos em tempo de execução. Experiências em sistemas convencionais têm mostrado que cada uma dessas formas são difíceis de serem colocadas em prática, pois reuso significa uma compatibilidade do novo sistema com o que já existe, o que as vezes é bastante difícil.

No caso de reuso e compartilhamento de conhecimento, algumas dificuldades significativas são encontradas (NETCHES *et al.*, 1991, SWARTOUT *et al.*, 1994).

A primeira dificuldade refere-se à existência de uma grande variedade de representação do conhecimento e o fato de que o conhecimento expresso em um formalismo não pode diretamente ser incorporado em outro. Além disso, não existe uma linguagem de representação que seja melhor para capturar todo tipo de conhecimento o que torna essa diversidade inevitável. Compartilhamento e reuso envolvem, portanto, a tradução de uma representação em outra. Ontologias podem ser utilizadas como uma linguagem para facilitar essa tradução. MIZOGUCHI (1994) enfatiza que, mesmo com a existência de tradutores, o conhecimento codificado para um certo problema não pode ser utilizado por outro sem modificação e que, muitas vezes, essa modificação se torna tão trabalhosa que é melhor construir a base de conhecimento do nada (*from scratch*).

Mesmo utilizando uma mesma representação nos diferentes sistemas, surge outra dificuldade: a profusão de dialetos entre as linguagens. Algumas diferenças entre os dialetos são substantivas, outras são triviais, mas ambas impedem o compartilhamento e o reuso.

Uma terceira dificuldade surge se considerarmos a possibilidade de compartilhamento entre sistemas com diferentes bases de conhecimento e não através do compartilhamento de uma base de conhecimento comum. Para isso, seria necessário definir meios de comunicação entre os sistemas através dos quais estes compartilhariam conhecimento usando consultas entre eles quando necessários. No entanto, não existem protocolos definidos para essa comunicação entre sistemas (como seriam as consultas, como deve ser dado a resposta e como utilizar a resposta), nem protocolos padronizados para prover interoperabilidade entre sistemas baseados em conhecimento e sistemas convencionais.

Finalmente, mesmo que dois sistemas usem a mesma linguagem de representação e tenham bons meios de comunicação entre eles, compartilhamento do conhecimento pode ser impossível devido à não utilização de uma terminologia comum do domínio. O maior enfoque na pesquisa em ontologias é a busca de uma solução para esta questão. Desenvolver conjuntos compartilhados de terminologias explicitamente definidas pode remover significativas diferenças no nível do conhecimento.

3.3 Projetos baseados em Ontologias

O uso de ontologias como forma de promover compartilhamento e reuso é uma área que ainda está em fase inicial de pesquisa. Os projetos nessa área buscam a definição do conhecimento do domínio, através da explicitação formal do conhecimento em ontologias do domínio, para ser utilizado no desenvolvimento de sistemas baseados em conhecimento. Os principais projetos nesse sentido são o *ARPA Knowledge Sharing Initiativte* e o projeto ESPRIT Kactus e o *DARPA Joint Forces Air Component Commander.*

O projeto *ARPA Knowledge Sharing Initiative* (NETCHES *et al.,* 1991, SWARTOUT *et al.,* 1994, NETCHES, 1994) é realizado na Universidade de Stanford (*Knowledge Systems Laboratory*) e tem como objetivo identificar como o conhecimento de uma aplicação individual, baseada em conhecimento, pode se tornar disponível para outra aplicação, ou seja, como bases de conhecimento de sistemas individuais podem ser compartilhadas e reutilizadas. Para alcancar este objetivo foi desenvolvida uma linguagem para troca de conhecimento com o objetivo de servir como uma interlingua, definindo protocolos de comunicação entre sistemas através da criação de uma linguagem de consulta e a criação de bases de conhecimento reutilizáveis e compartilhavéis. Pesquisadores tem se dedicado a testar o compartilhamento do conhecimento através da utilização de ontologias comuns como uma forma de superar a falta de consenso entre as bases de conhecimento no que se refere ao vocabulário e interpretações semânticas em modelos de domínio. O projeto defende a idéia da construção de sistemas baseados em conhecimento através da composição de componentes reutilizáveis disponíveis em uma biblioteca que serve como repositório de ontologias, bases de conhecimento, ferramentas e sistemas baseados em conhecimento.

O projeto Kactus (SCHREIBER *et al.*, 1995) tem como objetivo o desenvolvimento de métodos e ferramentas para reuso do conhecimento sobre sistemas técnicos durante o seu ciclo de vida, de forma que a mesma base de conhecimento pode ser utilizada para projeto, diagnóstico, operação, manutenção, reprojeto e instrução (tarefas típicas de sistemas baseados em conhecimento). Com este objetivo são definidas ontologias do domínio que são reutilizadas pelas diferentes aplicações. Ontologias são formuladas como um ponto de vista de meta-nível sobre um conjunto de possíveis teorias do domínio. Além disso, o projeto usa uma linguagem específica que faz uma clara distinção entre os conhecimentos do domínio e controle.

Outro projeto baseado em ontologia, mas com um objetivo mais ambicioso, é o projeto CYC (LENAT e GUHA, 1990, GUHA e LENAT, 1994, LENAT, 1995b) que tem como objetivo a definição de uma grande base de conhecimento de senso comum que seria utilizada pelas próximas gerações de software. Por senso comum, entende-se aqui, uma grande quantidade de conhecimento útil para a maioria das tarefas do mundo real (coisas como tempo, espaço, causalidade, eventos, capacidades humanas, etc). O projeto se preocupou com três aspectos: a definição de uma linguagem de representação específica, a construção da máquina de inferência com heurísticas específicas e a construção da base de conhecimento propriamente dita. Para construção da base de conhecimento foi definida uma ontologia que organiza os conceitos em categorias (ou coleções) numa hierarquia de generalização/especialização, onde uma categoria pode ter mais de uma generalização direta. Os pesquisadores se concentraram em construir a base de conhecimento manualmente. Nesse sentido, foram incluídos mecanismos de contextos para especificar uma tarefa de solução de problema específica, focalizar uma parte específica da base de conhecimento durante a solução do problema ou uma teoria geral de algum tópico (chamadas de microteorias). Cada microteoria captura alguma solução adequada para alguma área ontológica: áreas gerais (como tempo, substâncias, agentes e causalidade) ou áreas específicas (como manufatura, psicologia, refeições, etc.). Alguns protótipos foram desenvolvidos mas o esforço principal do grupo de pesquisadores foi na construção dessa grande base de conhecimento.

O projeto *DARPA Joint Forces Air Component Commander* (JFACC) (VALENTE *et al.*, 1999) tem como objetivo utilizar uma grande ontologia para planejamento de operações militares aéreas para facilitar a interoperabilidade e a comunicação entre sistemas com terminologia comum; promover o compartilhamento e reuso entre sistemas (em particular, integrar os esforços de aquisição e modelagem do conhecimento); e, criar um repositório sobre conhecimento geral sobre planejamento de operações militares aéreas para ser utilizado por um amplo programa de pesquisa em diferentes aplicações incluindo sistemas tradicionais (não baseados em conhecimento). A ontologia do JFACC tem em torno de 1500 entidades, entre conceitos, relações e instancias englobando informações sobre planos de operações militares aéreas, mísseis, bombas, tipos de aviões, unidade de forças militares, etc. Três aplicações já foram desenvolvidas utilizando essa ontologia: o *Strategy Development Assistant*, um assistente para auxiliar planejadores a decompor seus objetivos em sub-objetivos seguindo a teoria de planejamento de operações militares aéreas; *Inspect*, projetado para

criticar os planejamentos inseridos pelo usuário a partir de uma biblioteca de erros e problemas, e, *Mastermind*, um editor de objetivos de operações militares aéreas que usa uma abordagem de gramática de casos. Os novos experimentos, em realização neste momento, tem como objetivo substituir os esquemas de banco de dados usuais e orientados a objetos pela ontologia, de forma a resolver o *gap* entre sistemas tradicionais e sistemas baseados em conhecimento.

Ontologias também tem sido utilizadas para construção e reuso de componentes de sistemas baseados em conhecimento pelo benefício da separação entre tarefas, domínio e métodos. OUSSALAH e MESSAADIA (1999), por exemplo, usam ontologias para especificar uma linguagem de componentes descritivos de forma que o usuário possa, rapidamente, entender e encontrar os componentes que melhor se adequem a aplicações específicas. Isso é realizado criando um nível de ontologias para a biblioteca de componentes que explicita os diferentes conceitos, tarefas, domínio, métodos de solução de problemas e relações entre os conceitos. MOTTA *et al.* (1999) propõe uma biblioteca de resolvedores de problema para sistemas baseados em conhecimento onde ontologias são utilizadas para caracterizar a terminologia associadas a tarefas e métodos de solução de problemas, facilitando a busca e o reuso dos componentes.

Finalmente, um trabalho de grande importância é o sistema UMLS (*Unified Medical Language System*) da Biblioteca Nacional de Medicina do Instituto Nacional de Saúde dos Estados Unidos (UMLS, 1998). Este sistema foi projetado para facilitar a busca e troca de informações eletrônicas entre diferentes sistemas da área biomédica através da definição de uma terminologia comum. As fontes de interesse incluem: descrições de literatura biomédica, registros eletrônicos, banco de dados de fatos, sistemas baseados em conhecimento, e diretórios de pessoas e organizações. Para isso, o UMLS possui uma ontologia de 135 conceitos médicos organizados em uma rede semântica com 51 relações diferentes e um grande número de instâncias para esses conceitos (FRIDMAN e HAFNER, 1997). UMLS tem sido utilizado por diferentes aplicações da área biomédica incluindo banco de dados médico na internet[1] e o próprio servidor de conhecimento[2] do UMLS. No entanto, BODENREIDER *et al.* (1998) concluíram numa recente avaliação do UMLS que embora o UMLS seja uma boa fonte de conhecimento, as categorias de conceitos definidas são muito amplas precisando serem refinadas ou mesma serem definidas novas categorias para a aplicação específica.

[1] http://igm.nlm.nih.gov
[2] http://umlsks.nlm.nih.gov

Além disso, indicam a necessidade de um versão internacional completa do UMLS para que possa ser realmente utilizado entre sistemas de diferentes instituições de diferentes nacionalidades.

Percebe-se, portanto, apesar de em estágio de uso ainda inicial, o uso de ontologias tem trazido novas direções para a interação entre sistemas e usuários e entre os próprios sistemas. FIKES e FARQUHAR (1999) vislumbram um futuro no qual servidores de ontologia (semelhante aos servidores de banco de dados distribuídos) se tornarão altamente comuns apoiando o amplo acesso e reuso de ontologias e acelerando a construção de sistemas ricos em conhecimento.

3.4 Considerações Finais

O enfoque ao domínio passou a ser visto como um fator essencial no desenvolvimento de sistemas e no apoio automatizado para essa atividade. Muitas pesquisas têm procurado a integração de informações específicas de um domínio para apoiar o desenvolvimento de sistemas naquele domínio. Pesquisas que usam análise de domínio têm buscado a organização de modelos do domínio colocando o foco na construção de componentes de software para reutilização, no desenvolvimento de uma aplicação dentro de um domínio. Pesquisas em ontologia tem buscado organizar o conhecimento considerando os diferentes enfoques de um problema separados, ou seja, os conceitos diretamente relacionados com o domínio propriamente dito, conceitos genéricos de qualquer domínio, conceitos sobre tarefas e sobre as próprias formas de representação. Para ambos, o produto principal é o modelo do domínio gerado representado em linguagens específicas. No caso de ontologias, esse modelo deve explicitar formalmente os conceitos e as restrições entre os conceitos provendo melhor semântica para os termos.

A maioria dos trabalhos, no entanto, pouco discute sobre a necessidade de entendimento do domínio antes de se projetarem soluções para o mesmo. A maior preocupação é em como especificar e implementar uma solução para um problema de um determinado domínio através da composição de objetos projetados anteriormente. Dessa forma, arquiteturas específicas do domínio são centradas na definição e implementação de estilos arquiteturais para uma determinada família de aplicações, o que só é aceitável quando a organização não trabalha com diferentes tipos de aplicação já que as soluções arquiteturais não serão únicas. O ambiente baseado em conhecimento proposto por GOMMA *et al.* (1996) preocupa-se com a modelagem de subsistemas e o

apoio ao reuso desses modelos. No entanto, admite que adaptações ainda terão que ser realizadas. No mesmo caminho, mas na linha de IA, os projetos utilizando ontologias buscam o reuso de bases de conhecimento.

O ambiente AMPHION oferece uma melhor alternativa de especificação para um problema de forma informal e com geração automática de código. Mas, pelo caráter extremamente formal da especificação gerada, não são aplicáveis a todos os domínios e são de difícil construção.

Os ambientes propostos por FISCHER (1996) são os únicos que tentam abordar o entendimento do problema, que este chama de atividades *upstream*, procurando definir conceitos e estabelecer regras de restrição de uso desses conceitos. No entanto, Fischer não propõe nenhuma forma bem definida de organizar esse conhecimento, bem como nenhuma forma clara de como os especialistas podem interagir com o ambiente e inserir um novo conhecimento. Além disso, DODE (*Domain Oriented Design Environment*) tem sido desenvolvido com sucesso em um número limitado de domínios cuja principal característica é o projeto visual, sendo que o desenvolvimento de software não é, particularmente, visual (SELFBRIDGE, 1994). É difícil projetar esse tipo de solução para diferentes domínios complexos que tem pouco a ver como interface e interconexões (NING, 1994).

Nessa tese, o objetivo é apoiar o entendimento do domínio disponibilizando o conhecimento do domínio independente de quais sistemas já existam na empresa e dos produtos de software que se deseja construir. Esse conhecimento deve representar as características intrínsecas do domínio, suas restrições e organização. Além disso, esse conhecimento deve ser útil na construção de ferramentas de desenvolvimento de software e deve poder ser utilizado no desenvolvimento dos próprios produtos de software. Dessa forma, optamos por utilizar ontologia para modelagem do domínio. Essa escolha se deve a várias razões: o não comprometimento da modelagem do conhecimento do domínio com qualquer aspecto que não sejam as características do próprio domínio, a possibilidade de trabalhar com conhecimento que seja processável e que possa ser utilizado em assistentes para apoio ao desenvolvimento no domínio, a boa e clara definição de como esse conhecimento deve ser explicitado, estruturado, e representado; e pela separação bem definida do que significam os conceitos de um domínio e as tarefas que são automatizadas sobre eles. Finalmente, a ontologia expressa claramente a estrutura do conhecimento de um domínio. Segundo CHANDRASEKARAN *et al.* (1999) sem ontologia, ou conceituação sobre o

conhecimento do domínio, não é possível existir vocabulário para representar o conhecimento de um domínio, devendo este ser, portanto, o primeiro passo para uma efetiva representação do conhecimento ou de um vocabulário, pois a ontologia forma o coração de qualquer sistema de representação do conhecimento para um determinado domínio.

No próximo capítulo será apresentada uma proposta de apoio ao desenvolvimento de software, denominada Ambiente de Desenvolvimento de Software Orientado a Domínio, cujo objetivo é atender ao problema fundamental no desenvolvimento de software que é o entendimento do domínio.

Capítulo 4

Ambientes de Desenvolvimento de Software Orientados a Domínio

Este capítulo conceitua Ambientes de Desenvolvimento de Software Orientado a Domínio englobando os aspectos referentes à sua construção e às facilidades de uso que devem ser providas no ambiente para apoiar o desenvolvimento de sistemas.

4.1 Definição

Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD) são ADS que apoiam o desenvolvimento de software em domínios específicos através do uso do conhecimento deste domínio durante todo o processo de desenvolvimento para auxiliar o desenvolvedor no entendimento do problema. O domínio é uma área de aplicação (como por exemplo leis, cardiologia, etc.) na qual vários produtos de software serão desenvolvidos.

A concepção desses ambientes está baseada na premissa de que uma das principais dificuldades no desenvolvimento de sistemas é o fato dos desenvolvedores de software não conhecerem, ou não estarem familiarizados com o domínio para o qual devem desenvolver um determinado sistema. Portanto, ao contrário da maioria dos projetos orientados a domínio apresentados no capítulo anterior, que visam, principalmente, a organização de componentes do domínio para facilitar o reuso, os ADSOD visam a organização do conhecimento do domínio para facilitar o entendimento do problema no desenvolvimento de sistemas.

Dessa forma, o aspecto central dos ADSOD é a introdução e uso do conhecimento do domínio no ADS. As principais questões para a definição de um ADSOD referem-se, então, a: *(i)* qual o conhecimento que deve estar disponível no ambiente, *(ii)* como este conhecimento deve estar organizado e representado, e, *(iii)* quando e como utilizar este conhecimento definido durante o desenvolvimento.

Para responder às questões *(i)* e *(ii)* foi definida a Teoria do Domínio como o modelo que deve conter o conhecimento do domínio a ser utilizado no ambiente. A Teoria do Domínio considera descrições de tarefas relacionadas aos conceitos para facilitar o entendimento do domínio. No que se refere, à questão *(iii)*, deve-se considerar o estudo e investigação do domínio nas diferentes atividades do processo de software apoiando suas realizações. Esses diferentes elementos (teoria do domínio, descrição de processo e processo de software) são incorporados ao ADSOD através de ferramentas de definição e são utilizados ao longo do desenvolvimento de um sistema específico através de ferramentas de desenvolvimento. Essas ferramentas tem como objetivo permitir a edição de uma teoria do domínio ou processo e a assistência nas atividades de desenvolvimento de software. A Teoria do Domínio, a orientação a domínio e as ferramentas de definição e desenvolvimento serão detalhadas nas seções seguintes deste capítulo.

A visão geral dos elementos básicos de um ADSOD é apresentada na Figura 4.1. Os engenheiros do ambiente definem o processo de software e a Teoria do Domínio específicas para o ADSOD a ser construído. Além disso, definem tarefas que podem ser aplicadas àquele e a outros domínios. Uma vez concluído o ambiente, os usuários de desenvolvimento através de ferramentas específicas acessam e utilizam a Teoria do Domínio em cada uma das atividades do processo de software. Nas próximas seções são detalhados cada um destes elementos.

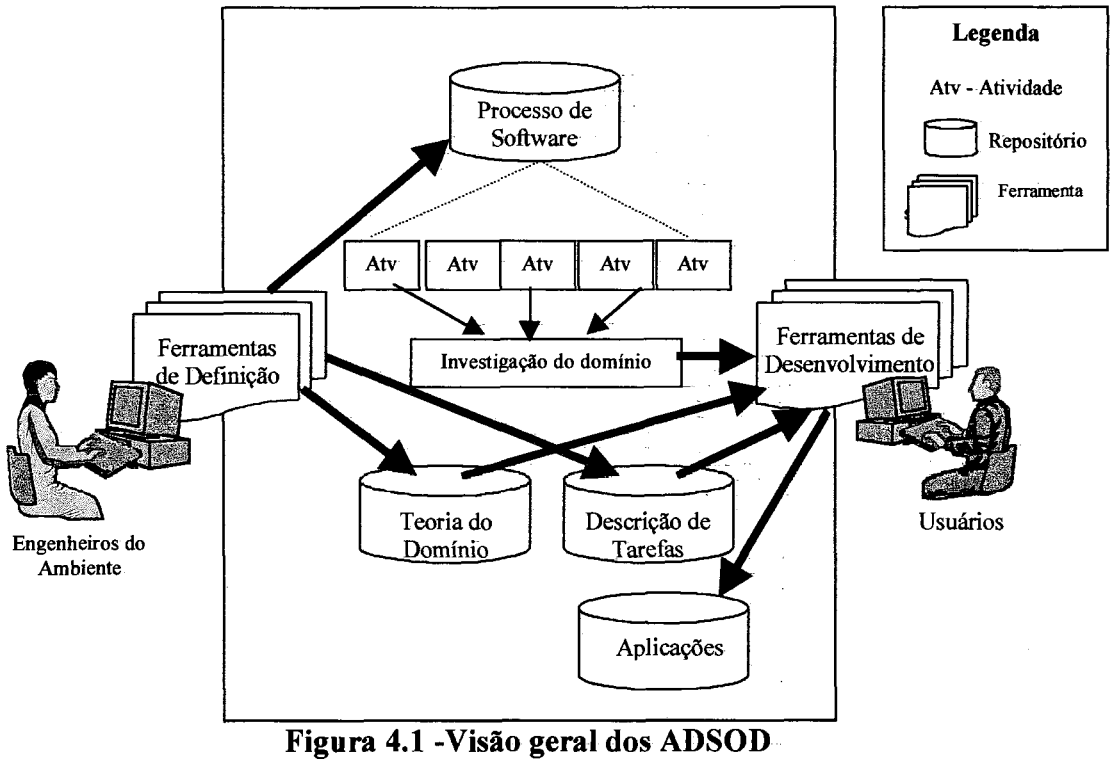

Figura 4.1 -Visão geral dos ADSOD

4.2 A Teoria do Domínio

A Teoria do Domínio é o modelo de conhecimento do domínio a ser utilizado para assistir o desenvolvedor ao longo do desenvolvimento de software. A Teoria do Domínio é composta basicamente da organização de conceitos, propriedades e restrições (axiomas) do domínio através de ontologias do domínio e do mapeamento desses conceitos com tarefas identificadas para o domínio considerado.

Identificar e definir os conceitos de um domínio pode ser um processo muito longo porque um domínio é, geralmente, muito amplo e rico em detalhes. Dessa forma, uma Teoria do Domínio deve ser dividida em sub-teorias. Cada sub-teoria considera os conceitos do domínio e a relação entre os conceitos que estão semanticamente mais relacionados e em um mesmo nível de abstração. Cada sub-teoria é uma ontologia sobre uma parte específica do domínio que se refere a um mesmo contexto dentro do domínio considerado.

A escolha do uso de ontologias do domínio para ADSOD se deve a várias razões. Primeiro, o interesse do modelo de domínio nos ADSOD está centrado na definição de conhecimento do domínio, o que implica na organização de informações sobre esse domínio que permita inferir conclusões sobre ele próprio e representar regras e conceitos do domínio específico. Segundo, ontologia pode ser vista como uma inter-língua para um domínio, estabelecendo terminologias claras que podem ser utilizadas pelas diferentes pessoas envolvidas no desenvolvimento (usuários, gerentes e desenvolvedores) para facilitar a comunicação e o entendimento comum do domínio e para garantir a interoperabilidade entre os sistemas construídos. Um terceiro aspecto é o fato de ontologias serem úteis na construção de ferramentas de elicitação e aquisição servindo como uma linguagem do domínio para ser utilizada para interação com os diferentes especialistas. Finalmente, a definição de ontologias do domínio é independente da representação e do uso em sistemas específicos. Ao definir ontologias, trabalha-se diretamente com o domínio em si independente dos módulos e sistemas que serão construídos para serem reutilizados. No que se refere a reuso, ontologias permitem a reutilização em um nível mais alto de abstração, o reuso do conhecimento. Esse reuso se dá seja a nível estrutural no que se refere ao conhecimento epistemológico ou a nível conceitual no que se refere ao conhecimento ontológico definido.

Para compor a Teoria do Domínio como um todo, são definidas, ainda, relações entre as sub-teorias. Essas relações são, na verdade, relações entre os conceitos das sub-

teorias envolvidas, sendo, portanto, também definidas e descritas através de restrições axiomáticas. A divisão da Teoria do Domínio em sub-teorias é bastante útil para reutilização de conceitos do domínio na definição de outras teorias ou no desenvolvimento de aplicações específicas.

Cada sub-teoria é, ainda, mapeada com potenciais tarefas pertinentes ao domínio. Uma tarefa é geralmente independente do domínio, sendo genérica e aplicável a diferentes domínios. Pode-se encontrar, por exemplo, sistemas de reservas para livros, carros ou qualquer outro objeto, sistemas de diagnóstico de uma doença ou de falhas em automóveis e sistemas de vendas, entre outros. Cada uma dessas tarefas possui características próprias que serão adaptadas em qualquer domínio escolhido.

Para a definição da Teoria do Domínio em um ADSOD pressupõe-se a existência e a contínua definição de um repositório de descrições dessas tarefas genéricas para serem aplicadas dentro de um domínio. Essas descrições são identificadas pelo próprio nome da tarefa que serve como uma espécie de índice para armazenamento e acesso no repositório.

Existem diferentes trabalhos na literatura que abordam a definição e organização de tarefas. Na inteligência artificial, essa idéia foi preconizada por CHANDRASEKARAN *et al.* (1993) quando propôs a existência de tarefas genéricas utilizadas nos sistemas baseados em conhecimento. Na mesma linha, o grupo do projeto ESPRIT que propôs o método KADS (HICKMAN *et al.*, 1992) organizou modelos de interpretação para um conjunto de tarefas comuns. O projeto ARPA (SWARTOUT *et al.*, 1994, NECHES, *et al.*, 1991) defende a criação de bibliotecas de ontologias de tarefas para serem reutilizadas. Uma pesquisa importante na linha de sistemas de informação é liderada por SUTCLIFFE e MAIDEN (1998) que definem a organização de conceitos sobre tarefas de tal forma que possa apoiar a elicitação de requisitos para aquela tarefa quando aplicável em determinado domínio.

A descrição de tarefas para ser utilizada no ADSOD é uma combinação de diferentes características dessas abordagens. Cada tarefa é descrita segundo um padrão específico que estabelece: *(i)* uma descrição de alto nível da tarefa; *(ii)* uma ontologia dos conceitos daquela tarefa específica, e, *(iii)* um conjunto de referências bibliográficas para as mesmas. A descrição de alto nível é feita a partir do entendimento sobre a tarefa específica e das descrições definidas por CHANDRESEKARAN *et. al* (1993) e pelo KADS. A definição de ontologias de tarefas envolve um esforço de organização dos

conceitos e definição dos seus componentes. As referências bibliográficas servem como indicação para busca de um melhor entendimento sobre aquela tarefa específica.

A identificação de tarefas e o mapeamento com os conceitos do domínio específicos (organizados em sub-teorias) são uma forma complementar de entender sobre os conceitos do domínio através do entendimento de como e para que esses conceitos são utilizados. No entanto, o fator chave da Teoria do Domínio é a definição de ontologias do domínio.

Em suma, a Teoria do Domínio contém os componentes necessários para auxiliar na orientação ao longo do desenvolvimento, explorando os conceitos e relações particulares do domínio em questão.

4.3 Engenharia do Domínio

Como a Teoria do Domínio corresponde basicamente à definição de um conjunto de ontologias, a Engenharia do Domínio considera, essencialmente, as características dos métodos para definição de ontologias.

Conforme apresentado no capítulo anterior, existem vários métodos para a engenharia de ontologias. Todos eles têm características essenciais importantes na engenharia de um domínio: os aspectos de elicitação, formalização, documentação e definição de um processo sistemático. A Engenharia do Domínio em um ADSOD nada mais é do que a combinação dessas características de forma a permitir a modelagem da Teoria do Domínio.

Assim sendo, as quatro principais etapas para construção de uma Teoria do Domínio são: *(i)* identificação do propósito; *(ii)* definição; *(iii)* formalização; e, *(iv)* codificação. Atividades de apoio estão continuamente presentes em cada uma das etapas. Essas atividades de apoio referem-se à documentação, avaliação, aquisição do conhecimento e integração com ontologias existentes, podendo ter maior ou menor influência conforme a etapa que está sendo executada. A documentação gerada é específica de cada uma das etapas sendo bastante detalhada na etapa de definição. Nesta etapa, técnicas de aquisição de conhecimento são, também, intensamente utilizadas. A atividade de avaliação deve ser realizada com mais de um especialista e deve buscar os critérios definidos no capítulo anterior. A integração com ontologias existentes pode ser uma atividade bastante comum quando existem várias sub-teorias a serem definidas. A Figura 4.2, apresenta o processo de Engenharia do Domínio, enfatizando suas diferentes etapas de desenvolvimento:

➢ **Identificação do Propósito**

Nesse passo é identificado, claramente, o propósito e uso esperado da Teoria do Domínio. Nos ADSOD esse propósito significa apoiar o desenvolvimento de sistemas em um determinado domínio. Esse objetivo deve ser caracterizado para o domínio específico, limitando o seu escopo e descrevendo o cenário de sua utilização.

A aquisição de conhecimento busca a identificação do que é o domínio. Neste

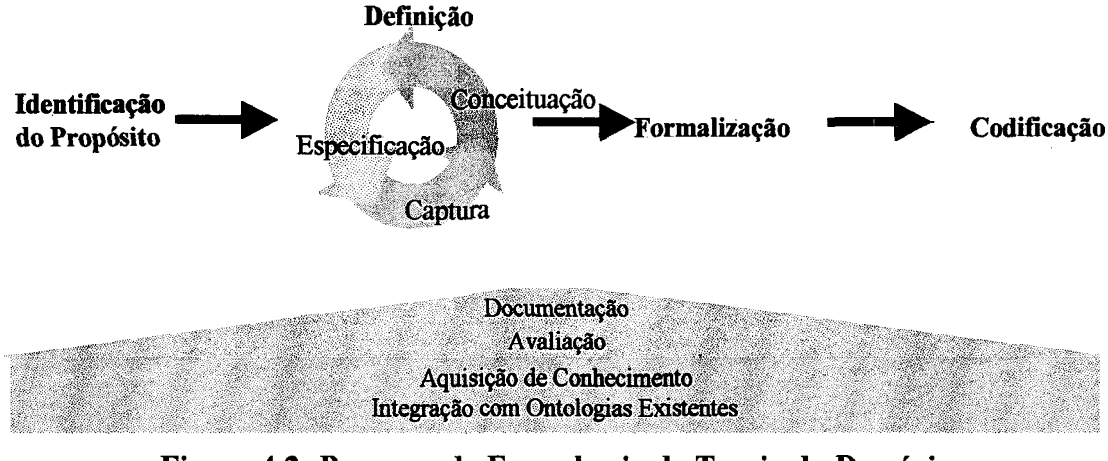

Figura 4.2 -Processo de Engenharia de Teoria do Domínio

momento, a documentação gerada deve descrever o domínio do problema e os cenários de motivação para a definição da Teoria do Domínio. Cenários de motivação são geralmente problemas ou situações (GRUNINGER e FOX, 1995) que mostram a utilidade da Teoria do Domínio.

➢ **Definição**

A definição da Teoria do Domínio é a etapa mais longa do processo e refere-se à modelagem do domínio propriamente dita. Esse passo consiste de três atividades: especificação de requisitos, captura do domínio e conceituação. A ***especificação de requisitos*** implica na definição das questões de competência do domínio, ou seja o que as ontologias do domínio deverão responder (GRUNINGER e FOX., 1995). Segundo FALBO (1998) ao se especificar um relacionamento entre as questões de competência e os cenários de motivação, está se dando uma justificativa para as ontologias e provendo um mecanismo para a sua avaliação. A ***captura do domínio*** refere-se à identificação dos conceitos, relacionamentos, propriedades e restrições do domínio através de técnicas específicas de aquisição de conhecimento. Finalmente, a *conceituação* envolve a geração de definições precisas e não ambíguas para o domínio capturado. Neste

momento, várias documentações são realizadas baseadas nas representações intermediárias definidas por GOMEZ-PEREZ *et al.* (1996) e LÖPEZ *et al.* (1999).

Não existe um ponto de início específico para a execução dessas atividades. A maioria dos métodos defendem um processo *top-down* onde primeiro são especificados os requisitos, para depois capturar o conhecimento que atenda àqueles requisitos e, por fim, modelar e documentar. No entanto, a identificação do conhecimento do domínio é difícil e cada uma dessas atividades podem não estar completas de uma só vez. Além disso, a abordagem *top-down* pode resultar na escolha e imposição de categorias de alto-nível muito arbitrárias porque não surgiram naturalmente (USCHOLD e GRUNINGER, 1996). Dessa forma, a filosofia adotada nesse passo é *middle-out,* que significa capturar os termos mais significativos do domínio, procurar entendê-los e definí-los , especificar os requisitos e retornar para uma captura e conceituação mais detalhada.

A etapa de definição requer várias iterações. Cada iteração envolve a execução dessas três atividades, seja de forma inicial ou como refinamento do que foi realizado na iteração anterior. O número de iterações pode variar de acordo com o tamanho do domínio e a experiência da equipe que está realizando a Engenharia do Domínio. No entanto, no mínimo quatro iterações são realizadas:

- Iteração 1 – identificação das sub-teorias;
- Iterações 2 a n-2 – definição e refinamento de uma sub-teoria;
- Iteração n-1 – definição de restrições axiomáticas para os conceitos e sub-teorias;
- Iteração n – identificação de tarefas e mapeamento das sub-teoiras do domínio com as tarefas.

Na primeira iteração é realizado um *brainstorming* conforme definido por USCHOLD e GRUNINGER (1996) produzindo um conjunto de termos e frases potencialmente relevantes para o domínio. A partir daí, termos que possuem alguma relação são colocados em um mesmo grupo, e novos termos vão sendo classificados em um dos grupos existentes ou formam novos grupos. Finalmente, deve-se identificar a semântica de cada grupo e a referência cruzada entre os grupos. As sub-teorias que compõem o domínio são, então, identificadas, nomeadas e descritas de acordo com os principais conceitos existentes na sub-teoria. Dessa forma, consegue-se especificar requisitos de mais alto nível, que serão atendidos por cada uma das sub-teorias identificadas.

Nas iterações de definição de sub-teorias, cada grupo de conceitos é exaustivamente explorado e detalhado. Técnicas de elicitação de conhecimento como entrevistas, pesquisa bibliografica e ordenação conceitual (ASSIS, 1992) são bastante úteis. Questões de competência específicas são determinadas. Conceitos são definidos e descritos. Atributos dos conceitos são identificados e são estabelecidas relações entre os conceitos. Cada um desses termos deve ser definido de forma não ambígua e consistente. Essas descrições devem considerar dois princípios importantes (FALBO, 1998): o princípio do vocabulário mínimo que diz respeito ao menor vocabulário a ser utilizado para definição dos conceitos, e o princípio da auto-referência, que estabelece que as definições, sempre que possível devem se referenciar a apenas definições já realizadas, procurando evitar a circularidade (USCHOLD e GRUNINGER, 1996).

A conceituação implica na definição e organização desses termos em vários documentos:

- um dicionário de dados (Tabela 4.1), com todos os conceitos, seus significados, sinônimos, exemplos de instâncias e atributos;
- organização dos conceitos de um grupo em árvores de classificação (quando pertinentes), definindo taxonomias de especialização e composição;
- dicionário de atributos (propriedades) de conceitos definindo significado, tipo e valores de domínio (Tabela 4.2), e, se necessário,
- tabela de fórmulas usadas para inferir valores numéricos relacionando os atributos.

Tabela 4.1 – Dicionário de Dados para Conceitos de uma Sub-Teoria

Conceito	Sinônimo	Descrição	Sub-Conceito	Instâncias	Atributos

Tabela 4.2 – Tabela de Atributos para Conceitos de uma Sub-Teoria

Conceito	Atributo	Descrição	Tipo	Faixa de Valores	Unidade

Um exemplo de uso dessas tabelas para documentação da Teoria do Domínio será apresentado no capítulo 6.

O formato desses documentos segue o mesmo padrão proposto pelo método para definição de ontologias proposto por GOMEZ-PEREZ *et al.* (1996). O ciclo pode ser repetido para a mesma sub-teoria com objetivo de refinar e completar as descrições obtidas.

A penúltima iteração (definição de restrições axiomáticas para os conceitos e sub-teorias) é, provavelmente, a etapa mais difícil da definição de ontologias (USCHOLD e GRUNINGER, 1996). Apenas definir os conceitos não constitui uma ontologia. Os axiomas provêm a semântica e o significado para todos os termos. Os axiomas são definidos com base nas questões de competência especificadas para cada sub-teoria e devem ser claros e objetivos. Axiomas são definidos para estabelecer restrições nas relações entre conceitos ou entre propriedades (atributos) de um conceito. Os axiomas devem, nessa etapa, ser descritos em linguagem natural para que seja possível sua validação pelos especialistas do domínio. Ao definir axiomas, novas iterações da etapa de definição podem ser necessárias para inclusão e refinamento dos conceitos das sub-teorias consideradas.

- **Formalização**

Esse passo consiste em escrever a ontologia na linguagem de representação escolhida, ou seja, definir, explicitamente, conceitos e relações através de uma linguagem formal. A escolha entre as várias linguagens existentes (ver capítulo anterior) normalmente, é definida pela disponibilidade para o grupo considerado. A maioria das linguagens, exceto a lógica de 1ª ordem, é definida especificamente no contexto de pesquisas específicas, provendo ambientes e compiladores adequados, ainda não havendo uma disponibilização comercial.

A depender da linguagem que será utilizada para formalização, todos os axiomas tem que ser explicitamente definidos pelos engenheiros do domínio, inclusive axiomas que representam especialização, composição e domínio de valores para os atributos.

- **Codificação**

Esse passo corresponde à implementação da formalização definida no passo anterior. Com isso, a Teoria do Domínio passa a estar disponível como módulos de conhecimento no ambiente para que possa ser instanciada e acessada. Nesse momento, a integração com ontologias existentes é fundamental para manter a completude e consistência da Teoria do Domínio.

4.3.1 Evolução do Domínio

A medida que diferentes aplicações vão sendo desenvolvidas no ADSOD a Teoria do Domínio pode evoluir incluindo mais conhecimento sobre o domínio. Duas formas de evolução podem ser consideradas: a instanciação e a evolução das ontologias.

Uma base de conhecimento pode ser vista como uma instanciação da ontologia (MUSEN, 1998). Nos ADSOD para todo conceito definido na Teoria do Domínio é gerada uma classe de conhecimento que é utilizada como interface para a criação de instâncias na base de conhecimento. Dessa forma, a medida em que novas aplicações são desenvolvidas, instâncias sobre os conceitos do domínio utilizados na aplicação específica são criadas como objetos da classe de conhecimento correspondente. Esses objetos são, por sua vez, armazenados na base de conhecimento do domínio formando módulos de conhecimento disponíveis no ambiente para apoio ao entendimento do domínio.

A evolução de conceitos da Teoria do Domínio, no entanto, é um pouco mais complexa. Pelo princípio da completude, uma ontologia deve ser completa para o grau de granularidade que ela se propõe. No entanto, novos conceitos podem surgir, o que torna necessário estender a ontologia. Segundo o critério de extensibilidade do projeto de ontologias, a capacidade de extensão refere-se à inclusão de novas definições sem alterar o conjunto de propriedades já avaliadas na ontologia. Dessa forma, a evolução da Teoria do Domínio (que é composta de ontologias) permite a inclusão de novos conceitos e propriedades mas não permite a alteração dos conceitos já existentes. Assim, aplicações desenvolvidas com base nos conceitos já definidos não precisarão ser alteradas mantendo a consistência e interoperabilidade entre as mesmas.

A inclusão de novos conceitos pode ser feita na própria sub-teoria correspondente, se não alterar a estrutura e organização já definidas. Isso é permitido quando os novos conceitos se relacionarem diretamente a um conceito já definido e possuem semântica fortemente associada à da sub-teoria correspondente. A melhor opção é definir os novos conceitos em novas sub-teorias e executar, novamente, os passos de definição, formalização e codificação do processo de Engenharia do Domínio sendo essencial a atividade de integração com ontologias já existentes, para definir as fronteiras e relações com conceitos definidos em outras sub-teorias.

4.4 Desenvolvimento Orientado a Domínio

Para desenvolver software de forma organizada e sistemática é necessário o uso de um processo de desenvolvimento que estabeleça um conjunto bem definido de atividades a serem realizadas (NBR ISO/IEC 12207, 1998). De forma geral, as principais atividades de desenvolvimento de software são: análise/definição do sistema, planejamento, análise de requisitos, projeto, codificação, teste e manutenção. A análise/ definição de sistemas é o primeiro contato dos engenheiros de software com o domínio no qual vão trabalhar, devendo ser estabelecido o objetivo e escopo do sistema a ser desenvolvido, definida a interface com outros sistemas, hardware e pessoas. O planejamento consiste da organização das atividades do processo de desenvolvimento de acordo com o ciclo de vida mais adequado ao projeto. A análise de requisitos é a atividade onde o sistema é realmente concebido e elaborado envolvendo o estabelecimento de requisitos de qualidade, a elicitação de requisitos de dados e funções, a modelagem funcional e de dados e a definição dos requisitos de banco de dados. A atividade de projeto é a transformação da modelagem funcional e de dados feita na análise em um modelo que possa ser codificado, compilado e testado. A partir da especificação do projeto o sistema é, então, codificado e testado. A manutenção consiste do conjunto de modificações a serem realizadas após o sistema ser implantado e pode envolver a realização das demais atividades novamente.

Todas essas atividades direta ou indiretamente são dependentes do conhecimento do domínio, seja pela necessidade do entendimento sobre o domínio e sobre a tarefa a ser automatizada na análise/definição do problema, pela verificação da complexidade do domínio para realização do planejamento, pelo uso do conhecimento do domínio para apoio a elaboração do projeto, programas e casos de teste, ou, principalmente, pelo apoio à realização de todas as sub-atividade da análise de requisitos que é a atividade principal do desenvolvimento de sistemas. No entanto, algumas atividades são mais orientadas a domínio do que outras, isto é, em algumas atividades existe uma maior necessidade do conhecimento do domínio do que em outras.

Para definir a orientação a domínio, nessas atividades, foi considerado o fato de que no processo de desenvolvimento de software cada uma das atividades é composta de várias sub-atividades. Com isso definimos uma sub-atividade específica, chamada **Investigação do Domínio,** cujo o objetivo é realizar a pesquisa e o uso do domínio necessário na atividade correspondente. A Investigação do Domínio deve ser incluída

nas várias atividades onde é importante a orientação a domínio (Figura 4.3) (como por exemplo nas atividades de análise e projeto). A Investigação do Domínio é apoiada por ferramentas específicas do domínio construídas utilizando a linguagem do domínio definida pela Teoria do Domínio. Ferramentas acessam e atualizam a base com a Teoria do Domínio de acordo com a atividade específica que estão apoiando.

Um fator importante de ser considerado, na definição de processo de desenvolvimento de software é que um processo deve, geralmente ser particularizado para cada tipo de software, para organização em que é utilizado para a equipe disponível para o desenvolvimento específico de um sistema. Diferentes atividades podem ser definidas. A orientação a domínio pode estar presente em qualquer uma das atividades definidas, sempre que se julge necessário o entendimento do domínio para a realização da mesma.

Considerando que algumas atividades sempre estão presentes em qualquer processo de software e que as principais atividades do processo de software são as apresentadas no início dessa seção, percebe-se que a maior influência refere-se às atividades de entendimento do problema, considerando-se que o bom entendimento do domínio é crucial para a qualidade do sistema produzido e para a adequação aos requisitos do usuário. As atividades com estas características são as de análise ou definição do sistema e análise de requisitos.

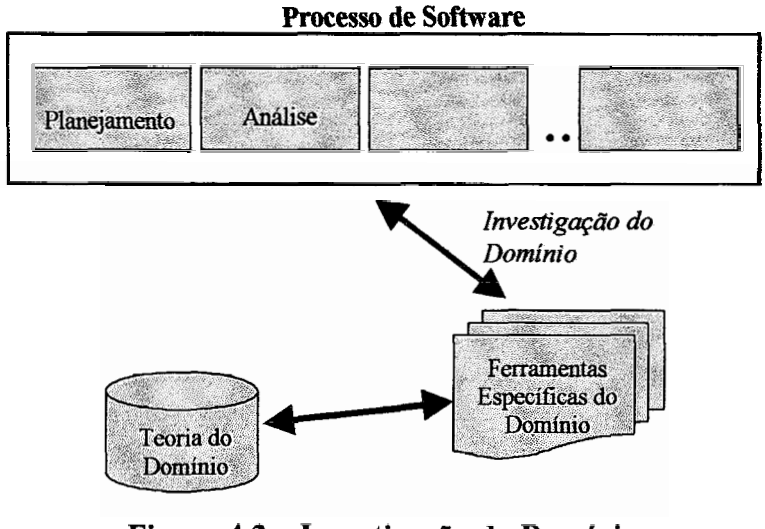

Figura 4.3 – Investigação do Domínio

Na análise ou definição do sistema, após identificado o objetivo do sistema o engenheiro de software pode pesquisar a Teoria do Domínio identificando a tarefa relacionada com o sistema a ser desenvolvido e, consequentemente, os conceitos que

podem estar relacionados com esta tarefa. Esta identificação inicial serve como estímulo para a próxima atividade pois define quais conceitos do domínio devem ser entendidos e explorados. Além disso, essa investigação é essencial para o entendimento do objetivo e da complexidade do domínio que serão fundamentais para o planejamento do projeto.

Na etapa de análise, o conhecimento do domínio é fortemente explorado e utilizado. A elicitação de requisitos (ou elicitação do conhecimento para sistemas baseados em conhecimento) é uma atividade sempre trabalhosa, e se torna mais complexa quando o engenheiro de software não tem nenhum conhecimento do domínio pois, além da identificação de quais são os requisitos do software, tem que entender cada um dos conceitos referidos pelos usuários (ou especialistas), sua importância no domínio e sua relação com outros conceitos do domínio. Nesse momento, a sub-atividade de investigação do domínio é de extrema importância, tanto na preparação para a realização desta atividade quanto na sua efetiva realização. Nos dois momentos, ferramentas específicas para o domínio podem apoiar o desenvolvedor de software facilitando o processo. Ferramentas específicas do domínio diferem das ferramentas tradicionais por utilizarem o vocabulário do domínio (um exemplo deste tipo de ferramenta é KED - um Editor de Conhecimento para Cardiologia - descrita no próximo capítulo e que apoia a elicitação de conhecimento). Na preparação para a elicitação de requisitos, ferramentas especialmente construídas para explorar o conhecimento do domínio podem mostrar os conceitos, descrições, suas relações e as características definidas na Teoria do Domínio. A Teoria do Domínio provê, ainda, o mapeamento de quais conceitos são importantes para determinadas tarefas indicando o que realmente deve ser investigado.

A Teoria do Domínio durante a elicitação funciona como um modelo inicial para os engenheiros de software. A construção da Teoria do Domínio já é, por si só, uma elicitação de conhecimento para qualquer aplicação naquele domínio. Ao desenvolver uma aplicação específica esta elicitação inicial é completada para se obter as particularidades da aplicação. Usando a Teoria do Domínio, os desenvolvedores podem identificar que conhecimento é relevante para a aplicação através do mapeamento das atividades, ou identificando os conceitos a partir de um conjunto de dados coletados em documentos referentes às tarefas na organização. Ferramentas específicas do domínio (como por exemplo, editores de conhecimento e ferramentas para elicitação de requisitos) podem ser construídas como forma de coletar novo conhecimento ou requisitos e possibilitar aos especialistas usarem diretamente o ambiente.

Na modelagem de dados do sistema, a investigação do domínio auxilia para estruturação dos dados de acordo com a organização definida na Teoria do Domínio.

Tabela 5.3 - Mapeamento entre conceitos ontológicos e modelo de dados (WAND, 1996).

Conceitos Ontológico	Construtor de Modelagem Semântica
Conceito	Entidade
Propriedade/atributo	Atributo
Esquema funcional (conjunto de conceitos)	Entidade tipo
Interação	Relacionamento
Composição	Agregação

Tabela 5.4- Mapeamento entre conceitos ontológicos e modelo de objetos

Conceitos Ontológico	Construtor de Modelagem Semântica
Conceito	Objeto
Propriedade/atributo	Atributo
Esquema funcional (conjunto de conceitos)	Classe
Interação	Comunicação/Relacionamento
Composição	Composição todo parte
Classificação	Especialização/Herança

Nesse sentido, WAND (1996) propôs uma correlação entre os conceitos ontológicos e os utilizados em modelagem de dados do tipo entidade-relacionamento (Tabela 4.3). De forma semelhante tem-se a correlação para a modelagem de objetos (PARSON e WAND, 1997) (Tabela 4.4). Além disso, a investigação do domínio permite a identificação as restrições (axiomas) entre os conceitos que definem as restrições de integridade dos dados modelados.

A definição de requisitos de banco de dados requer o conhecimento do tipo de informação do domínio para determinar requisitos de acesso e armazenamento de dados. A Teoria do Domínio pode indicar estas informações na própria descrição de cada um dos conceitos.

No que se refere às demais atividades do desenvolvimento de sistemas, a disponibilização da investigação do conhecimento do domínio através de navegadores ou assistentes é, a princípio, suficiente para uma orientação inicial. Ferramentas específicas podem ser construídas conforme o caso.

4.5 Ferramentas

O uso de ferramentas em um ADSOD é uma característica essencial dos ADS. Todas as atividades possíveis de automação devem ser apoiadas por ferramentas que trabalhem de forma integrada e transparente para seus usuários finais. No caso de ADSOD, tanto a introdução do conhecimento do domínio como o seu uso, deve ser apoiado por ferramentas específicas que considere suas particularidades. Um ADSOD deve ainda possuir ferramentas específicas do domínio, ou seja, ferramentas que utilizam a Teoria do Domínio com linguagem para iteração com o usuário. A seguir, são apresentadas um conjunto de diferentes ferramentas que são utilizadas para definição de um ADSOD quanto e para o desenvolvimento de software apoiado pelo mesmo.

4.5.1 Ferramentas de Definição

Três ferramentas são importantes para apoiar a definição de um ADSOD: um editor de Teoria do Domínio, um editor de tarefas e um assistente para definição de processo de software.

- **Editor de Tarefas**

O Editor de Tarefas serve para descrição de tarefas que embora sejam independentes do domínio, são importantes para o entendimento do uso de conceitos do domínio no desenvolvimento de sistemas. Esse editor deve permitir a descrição de tarefas a partir do padrão de descrição definido na seção 4.2.

- **Editor de Teoria do Domínio**

O Editor de Teoria do Domínio permite a definição de uma ontologia do domínio descrevendo os conceitos que caracterizam o domínio. Além disso, são identificados quais conceitos estão relacionados com quais tarefas conforme definido na seção 4.2. O Editor do Domínio deve permitir a documentação dos termos, a descrição informal e formal de axiomas, a geração dos dicionário de dados e da tabela de atributos. Além disso, deve gerar o conjunto de classes de conhecimento que devem ser utilizadas para instanciação dos conceitos à medida em que as aplicações vão sendo desenvolvidas.

- **Assistente para Definição do Processo**

O Assistente para Definição do Processo auxilia o engenheiro de software a definir o processo mais adequado ao tipo de software que se deseja desenvolver e de acordo com as características do projeto. O assistente indica em que atividades deve ser incluído a sub-atividade de investigação do domínio, e qual seu objetivo em cada momento. O engenheiro de software deve adaptar a atividade, quando necessário, para melhor adequação com o software a ser desenvolvido.

4.5.2 Ferramentas de Desenvolvimento

As atividades fundamentais do processo de desenvolvimento estão relacionadas às atividades de construção, gerência de desenvolvimento e avaliação da qualidade do produto.

As ***ferramentas para construção*** têm como objetivo auxiliar no processo de produção de um software desde a fase de especificação de requisitos gerais do sistema,

análise, projeto até sua implementação. As ferramentas de construção importantes de estarem presentes em um ADSOD são:

➢ **Assistente de Aprendizado do Domínio**

O Assistente de Aprendizado do Domínio permite a pesquisa de conceitos do domínio através da estrutura hierárquica definida na ontologia do domínio e de outros tipos de relações entre os conceitos. O desenvolvedor de software tem informações sobre o significado de conceitos, características específicas de cada conceito, situações de uso em sistemas já desenvolvidos e uma relação de tarefas que podem utilizar esses conceitos quando implementadas. O assistente permite o desenvolvedor aprender sobre conceitos do domínio respondendo a questões do tipo:

- O que é o conceito X?
- Que conceitos tem alguma relação com X?
- Como se caracteriza essas relações?
- Que tarefas usam o conceito X?
- Quais são algumas das instâncias do conceito X?
- Quais os estados do conceito X e como eles se modificam?
- Como o conceito X foi utilizado no desenvolvimento de sistemas?

➢ **Ferramenta para Elicitação Específica a Domínio**

Ferramentas para elicitação específicas ao domínio podem ser construídas utilizando a Teoria do Domínio. Ferramentas desse tipo permitem uma melhor interação com o especialista do domínio por utilizarem terminologia de uso comum para os mesmos. Ferramentas específicas do domínio podem ser construídas para diferentes tarefas. Com o uso dessa ferramenta os desenvolvedores de software tem um conjunto de informações para continuar o processo de elicitação.

➢ **Ferramenta de Apoio à Modelagem**

A Ferramenta de Apoio à Modelagem deve auxiliar o desenvolvedor a reutilizar definições e a estrutura de organização dos conceitos definidos na Teoria do Domínio. Esse apoio é baseado na correlação entre os conceitos ontológicos de modelos de dados e objetos conforme proposto por WAND (1996, 1997) (Tabelas 4.3 e 4.4).

➢ **Ferramenta para Registro do Uso de Conceitos**

Esta ferramenta é utilizada pelo desenvolvedor ao término de cada projeto. O desenvolvedor deve fazer uma breve descrição do sistema produzido, identificar e relacionar quais conceitos da Teoria do Domínio foram utilizados e como foram utilizados, e registrar exemplos de instâncias dos conceitos utilizados. Esta atividade é fundamental para o melhor entendimento do domínio que é apoiado pelo Assistente de Aprendizado do Domínio.

➢ **Ferramenta para Evolução da Teoria do Domínio**

O desenvolvimento de novas aplicações pode utilizar conceitos que ainda não foram definidos na Teoria do Domínio, já que o conhecimento não é estático e é implícito o que implica que dificilmente pode ser capturado de uma só vez. Dessa forma, é importante que o ADSOD ofereça uma ferramenta que permita ao engenheiro do domínio incluir novos conceitos, relações ou propriedades na Teoria do Domínio. A ferramenta não apoia a identificação desses novos conceitos e sim o seu registro na Teoria do Domínio. Cabe ao engenheiro do domínio analisar o sistema desenvolvido e identificar estes conceitos. Para manter a integridade com sistemas já desenvolvidos, não é permitida a alteração dos conceitos já existentes, mas apenas a inclusão dos mesmos.

Como todo ADS baseado em processo, um ADSOD deve disponibilizar **ferramentas para gerência** do processo, permitindo aos gerentes de projeto o acompanhamento das atividades que estão sendo realizadas, e aos engenheiros de software a execução dessas atividades, devendo prover ferramentas para apoio ao acompanhamento e adaptação do processo conforme porposto por ARAUJO (1998).

Finalmente, as **ferramentas para avaliação de qualidade** apoiam desde a identificação de requisitos de qualidade e o planejamento do controle de qualidade até a sua efetiva realização. No ADSOD, essas ferramentas devem considerar atributos de qualidade definidos para o tipo de software do domínio em questão. Além disso, tanto a identificação de requisitos como a avaliação da qualidade dos produtos de software deve levar em consideração o perfil dos diferentes avaliadores daquele domínio, conforme definido por BELCHIOR (1997). Essa ferramenta deve ser customizada para cada domínio específico através da inclusão de atributos de qualidade específicos.

4.6 Usuários do ADSOD

Dois grupos de usuários são importantes no contexto dos ADSODs: os engenheiros do ambiente, responsáveis pela construção do ambiente, e os usuários do ambiente, que efetivamente usam o ambiente. Engenheiros de software, engenheiros do domínio, especialistas do domínio, gerentes do projeto e desenvolvedores de software compõem esses grupos. Os engenheiros de software, engenheiros do domínio e especialistas do domínio são tanto usuários como engenheiros do ambiente.

Como engenheiros do ambiente, os engenheiros de software são responsáveis pela definição do processo de software mais adequado para determinado projeto e, também, pela inclusão da sub-atividade de investigação do domínio de acordo com o tipo de software. Os engenheiros do domínio são responsáveis pela definição e evolução da Teoria do Domínio e pela interação com especialistas do domínio para a definição e a avaliação do conhecimento a ser incluído na mesma. A definição do domínio pode ser facilitada quando o engenheiro do domínio tem alguma experiência no desenvolvimento de software para o domínio.

Como usuários, os engenheiros de software avaliam, continuamente, o processo ao longo do desenvolvimento fazendo as adaptações necessárias. Os engenheiros do domínio avaliam cada novo conhecimento adquirido no desenvolvimento de aplicações para incluí-lo na Teoria do Domínio definida. Os especialistas do domínio utilizam ferramentas específicas do domínio nas atividades de elicitação de requisitos e aquisição do conhecimento.

Desenvolvedores e gerentes de projeto atuam como usuários do ambiente utilizando as ferramentas de construção, gerência e avaliação da qualidade durante o desenvolvimento. O grupo de gerentes do projeto pode ser composto de representantes do domínio específico e de informática, sendo os responsáveis pela definição e realização do projeto.

4.7 Considerações Finais

A medida em que cresce a complexidade dos sistemas, os pesquisadores tem buscado soluções menos genéricas e mais específicas para apoiar o seu desenvolvimento. A construção de ADS com este enfoque tem aumentado nos últimos anos. ADSOD, definido neste capítulo, se enquadra nesse grupo procurando apoiar a atividade de entendimento do problema com o uso do conhecimento do domínio.

ADSOD considera a introdução do conhecimento do domínio em um ambiente utilizando ontologias como definido pelos projetos baseados em ontologias apresentados no capítulo anterior. No entanto, o enfoque é no entendimento do domínio do problema como definido por FISCHER (1996). Como os projetos DASDE e KBSEE os ADSOD se propõem a orientar o desenvolvimento considerando o domínio embora com outro enfoque. O ADSOD proposto neste trabalho, possui, portanto, similaridades com os principais projetos orientados a domínio, mas possui diferenças significativas tais como a preocupação na organização e disponibilidade do conhecimento em um ADS projetado especificamente para um determinado domínio. Isso implica em não, apenas integrar o conhecimento mas também utilizar um processo de desenvolvimento de software e ferramentas específicos para o domínio e o contexto de desenvolvimento considerados. Na Tabela 4.5 é feito um comparativo entre os principais ambientes descritos no capítulo anterior de acordo com as seguintes características: a consideração do domínio no ambiente, o objetivo básico, a existência ou não de base de conhecimento e a arquitetura utilizada para apoiar o desenvolvimento de software.

A construção dessa nova classe de ADS exige uma infra-estrutura robusta que possibilite a especificidade de um ADSOD. No próximo capítulo, será apresentado como essas questões foram resolvidas para possibilitar a construção de ADSOD.

Tabela 4.5 – ADSOD x Principais Projetos Orientados a Domínio

Projeto / Característica	DASDE	KBSE	DODE	ARPA	Kactus	ADSOD
Domínio	Modelo do domínio definido pela análise de domínio	Modelo do domínio definido pela análise de domínio	Modelo do domínio definido pela análise de domínio	Várias ontologias para diferentes domínios	Uma ontologia para um domínio específico	Uma ontologia para um domínio específico
Enfoque básico	Reuso a partir da definição de arquiteturas	Reuso para especificação de requisitos	Entendimento do domínio do problema	Reuso de bases de conhecimento por várias aplicaçoes	Reuso de bases de conhecimento por várias aplicaçoes	Entendimento do domínio do problema
Base de Conhecimento	———	Uso de base de conhecimento para apoiar o reuso da especificação de requisitos	Base de conhecimento com regras específicas para a atividade de projeto	Várias bases de conhecimento	Uma base de conhecimento comum reutilizada em diferentes aplicações baseadas em conhecimento	Uma ontologia e base instanciável para a ontologia
Arquitetura	Ambiente genérico com ferramentas para apoiar a definição do domínio e arquitetura	Ambiente Genérico para apoio a modelagem do domínio e para assistência na especificação	Ambiente específico com ferramentas para apoio ao desenho e realização de críticas	Ambiente centrado na definição de uma biblioteca com componentes gerais e um conjunto de ferramentas para diferentes domínios disponíveis numa biblioteca	———	Ambiente baseado em processo projetado especificamente para o domínio com um conjunto de ferramentas específicas ao domínio projetadas de acordo com a ontologia definida

Capítulo 5

Construção de Ambientes Orientados a Domínio na Estação TABA

Para construção de Ambientes de Desenvolvimento de Software Orientados a Domínio, definidos no capítulo anterior, está sendo utilizada a infra-estrutura da Estação TABA. Neste capítulo, são apresentadas as alterações realizadas na definição e construção de ambientes de desenvolvimento de software na Estação TABA e os novos requisitos introduzidos para se considerar o domínio na instanciação de ADS. Finalmente é descrita a implementação realizada.

5.1 Processo de Software e Construção de ADS na Estação TABA

A Estação TABA, como foi descrito no capítulo 2, permite a instanciação do ADS mais adequado ao desenvolvimento de um produto específico, a partir da definição de um processo de desenvolvimento.

O processo de desenvolvimento é um dos processos do ciclo de vida de software, segundo a norma NBR ISO/IEC 12207 (1998). Esta norma é o padrão internacional para a definição de processos e agrupa as atividades que podem ser executadas durante o ciclo de vida de software em cinco processos fundamentais (aquisição, fornecimento, desenvolvimento, operação e manutenção), oito processos de apoio (documentação, gerência de configuração, garantia da qualidade, verificação, validação, revisão conjunta, auditoria e resolução de problemas) empregados e executados quando necessário por outro processo, e, quatro processos organizacionais (gerência, infra-estrutura, melhoria e treinamento) empregados para estabelecer e implementar uma estrutura subjacente, constituída de processos do ciclo de vida e pessoal associados, e melhorar continuamente a estrutura e os processos. Cada processo do ciclo de vida é dividido em um conjunto de atividades. Cada atividade, por sua vez, é dividida em um conjunto de tarefas.

Nenhum projeto é igual ao outro e nenhuma organização é igual a outra. Para acomodar estas variações a norma ISO 12207 foi definida para um projeto genérico. Portanto, deve ser adaptada para projetos e organizações específicos. Para isto, é necessário identificar as características do ambiente do projeto que influenciarão na adaptação. Algumas das características podem ser: modelo de ciclo de vida, requisitos de sistema e do software, políticas, procedimentos e estratégias organizacionais, tamanho, criticalidade, tipos de sistemas e quantidade de pessoas e partes envolvidas (ver anexos A e B da norma ISO 12207).

Nossa experiência no desenvolvimento de projetos específicos para diferentes organizações, também, tem mostrado que, para um processo ser realmente útil e adequado em uma organização, não basta a definição de boas práticas de engenharia de software, mas, também, deve-se considerar aspectos humanos e do ambiente de trabalho presentes durante o desenvolvimento de software e particulares de uma organização. O atendimento a este requisito faz que o processo seja melhor aceito e seja mais fácil de utilizar pela equipe de desenvolvimento da organização (ROCHA *et al.*, 1999).

É preciso, portanto, identificar o modo particular de trabalhar e a cultura da organização e contemplá-los no processo. Além disso, o problema crucial presente no desenvolvimento de software em uma organização pode não ser evidente, de imediato, e precisa ser identificado para se ter um processo realmente adequado. A motivação para esta tese surgiu, justamente, da identificação do problema crucial no desenvolvimento de software em uma organização, a UCCV/FBC, onde verificamos ser o principal problema, o desconhecimento do domínio (cardiologia) por parte dos desenvolvedores de software.

No contexto deste trabalho e da Estação TABA estamos interessados, apenas, no processo de desenvolvimento, pois é este que possibilitará a instanciação de ADS na Estação. Atividades dos processos de apoio e organizacionais são relacionadas no contexto do processo de desenvolvimento e no momento do ciclo de vida que motiva a sua realização. Desta forma, a partir deste momento, quando nos referirmos ao processo de desenvolvimento de software, o chamaremos apenas de processo de software.

Como diferentes tipos de software (sistemas de informação, sistemas baseados em conhecimento, aplicações multimídia, etc.) podem ser desenvolvidos, para um mesmo domínio e em uma mesma organização, torna-se, então necessário definir diversos processos de software de modo que estes sejam adequados para a sua construção.

Definir vários processos de software para uma mesma organização pode ser muito trabalhoso e ter como resultado processos completamente distintos, o que não é adequado. Pesquisas recentes tem mostrado a necessidade da padronização dos processos de software dentro de uma organização (EMAM *et al.*, 1998, NBR ISO/IEC 12207, 1998, MAIDANTCHIK, 1999, ROCHA *et al.*, 1999). Esta padronização pode ser obtida através da definição de um Processo Padrão para a organização, que seja a base para a definição dos processos de software adequados aos diferentes tipos de software e a partir do qual processos específicos possam ser definidos.

Dessa forma, consideramos que um processo de software definido na Estação TABA para instanciar um ADS, deve considerar a definição de um Processo Padrão para a organização para a qual se deseja construir o ambiente.

Processo Padrão, segundo o SPICE[1] *(Software Process Improvement and Capability dEtermination)* (EMAM *et al.*, 1998) é a definição operacional do processo básico que guia o estabelecimento de um processo comum dentro da organização. O Processo Padrão descreve os elementos fundamentais que devem ser incorporados em qualquer processo definido e as relações entre esses elementos (como seqüência e interfaces).

Neste trabalho, Processo Padrão é utilizado com o objetivo de estabelecer um processo de desenvolvimento comum para uma organização, de forma que possa ser utilizado para a definição de processos de software e ADS específicos para projetos, no contexto da Estação TABA. A norma ISO12207 é a base para a definição de qualquer processo de software. Pode, também, ser considerado um dos modelos de maturidade CMM (*Capability Maturity Model*) (PAULK *et al.*, 1995) ou SPICE (EMAM *et al.*, 1998), conforme proposto por MACHADO e ROCHA(1999).

Para definir um Processo Padrão, a primeira questão a resolver é identificar quais são as atividades que devem ser consideradas. Segundo PFLEEGER (1998) as atividades de desenvolvimento de software, normalmente, englobam análise e definição de requisitos, projeto do sistema e dos programas, programação, testes, implantação e manutenção. Ainda, segundo PFLEEGER (1998) antes de iniciar um desenvolvimento o engenheiro de software frequentemente quer uma estimativa de prazos e custos sendo portanto necessário um conjunto de atividades para planejar e gerenciar o

[1] SPICE é um projeto que tem como objetivo desenvolver um padrão de avaliação que seja aplicável tanto para melhoria do processo quanto para determinação da capacidade de uma organização; sendo aplicável para qualquer domínio, necessidades de negócio ou tamanho da organização, sendo independente da estrutura da organização, de ciclos de vida, tecnologias ou métodos de software (EMAM *et al*, 1998)

desenvolvimento. Essa atividade está contida no processo de gerência, que faz parte dos processos organizacionais definidos na ISO/IEC 12207.

Ao definir um processo de software para uma organização, devem ser incluídas ainda atividades típicas daquela organização de processos já implantados (caso existam) ou atividades que são sempre realizadas no desenvolvimento mesmo que não estejam formalmente definidas em um processo e que podem não estar presentes no conjunto de atividades definidas na norma ISO/IEC 12207, no CMM ou no SPICE, caso estes sejam considerados.

Segundo EMAM *et al.* (1998) a definição de um Processo Padrão deve considerar os objetivos do processo; identificar as atividades, funções e responsabilidades; definir as entradas e saídas; os pontos de controle, os registros de qualidade e identificar interfaces externas e internas.

Para definição do Processo Padrão, no contexto da Estação TABA, estabelecemos que cada atividade do processo de software deve ser descrita através da: *(i)* definição de seus objetivos principais; *(ii)* definição e descrição das sub-atividades ou sub-processos a serem realizados; *(iii)* definição dos produtos gerados na atividade, e, *(iv)* identificação dos responsáveis envolvidos na realização das atividades e sub-atividades.

A partir do Processo Padrão definido para a organização, diferentes processos de software podem ser especializados considerando as características de cada tipo de software. Consideramos tipo de software um software que utiliza uma determinada tecnologia de desenvolvimento (como sistemas baseados e conhecimento, sistemas convencionais, etc.) seguindo um paradigma específico (como orientado objeto, estruturado, etc.). Neste momento, novas atividades podem, também, ser definidas e incluídas no processo especializado. Um determinado tipo de software pode ser desenvolvido através de diferentes modelos ciclos de vida e utilizando-se diferentes métodos e ferramentas. A adoção de um determinado método pode, ainda, implicar na inclusão de atividades específicas ao método. No entanto, todos os elementos básicos definidos no Processo Padrão deverão sempre estar presentes nos processos especializados.

O primeiro passo para a definição de um processo de software especializado, a partir de um processo padrão, é a escolha do modelo de ciclo de vida. A adoção de um determinado modelo de ciclo de vida impõe uma sequência para a realização das atividades de desenvolvimento e pode implicar na inclusão de atividades específicas (como por exemplo, a atividade de análise de riscos ao se adotar o ciclo de vida em

espiral). O próximo passo na especialização, é adequar a descrição de cada uma das atividades definidas no Processo Padrão às características do tipo de software a ser desenvolvido (por exemplo, no paradigma orientado a objetos a atividade de projeto considera a definição de classes e métodos enquanto no paradigma estruturado é definido um projeto de arquitetura em módulos e o projeto detalhado dos programas). Além disso, devem ser identificados os métodos e ferramentas utilizados na organização para aquele tipo de software.

Para ser utilizado em um projeto específico, o processo especializado mais adequado para um determinado tipo de software deve ser instanciado para atender às características do projeto específico. Nesse momento, deve-se considerar o tamanho e complexidade do produto, as características da equipe de desenvolvimento e a expectativa de vida útil do software e demais características do projeto.

A Figura 5.1 mostra o esquema global de definição de um processo de software a ser utilizado na Estação TABA. O processo de software instanciado final é, então, utilizado para instanciar o ADS na Estação seja ele orientado ao domínio ou não.

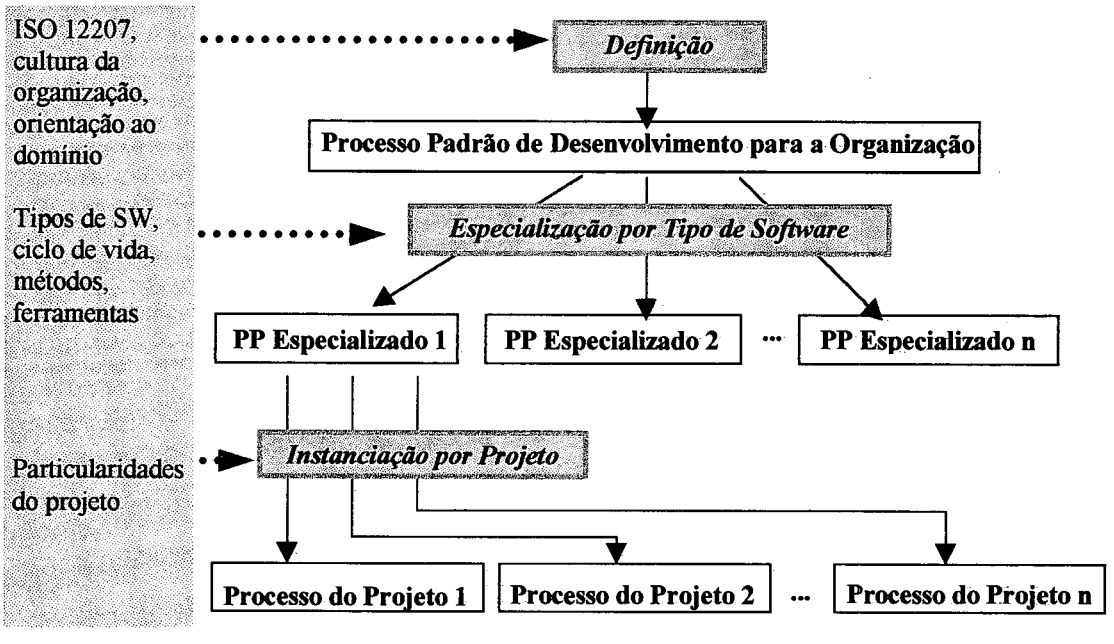

Figura 5.1 - Esquema de Definição de Processo de Software para um ADS na Estação TABA

Seguindo esse esquema, identificamos que para a construção de ADSOD a atividade de investigação do domínio definida no capítulo anterior deve ser introduzida como atividade na definição do Processo Padrão. Dessa forma, garantimos que todos os processos especializados e, posteriormente, instanciados vão considerar a orientação ao domínio no desenvolvimento do software específico (ROCHA *et al*, 1999).

5.2 Novos Requisitos e Arquitetura

A partir das características de ADSOD definidas no capítulo 4 e do modelo para definição de processos de software na Estação TABA descrito na seção anterior, novos requisitos foram acrescentados à Estação TABA para permitir a instanciação de ADSOD.

Em trabalhos anteriores (OLIVEIRA, 1996, VILLELA, 1997) foi estabelecida uma lista preliminar de requisitos para ADSOD. Esses requisitos foram refinados e criticados (OLIVEIRA *et al.*, 1998b) para uma identificação mais concisa e que atenda às definições apresentadas no capítulo 4. Dessa forma, os seguintes requisitos foram acrescentados aos requisitos gerais da Estação TABA:

- ***apoiar a definição da teoria de domínio:*** a Estação TABA deve possuir mecanismos que facilitem a definição de uma ontologia do domínio pelo engenheiro do domínio. Além disso, deve facilitar a definição de um modelo de tarefas de alto nível que identifique as tarefas realizadas no domínio, como elas se relacionam e como se relacionam com os conceitos do domínio definidos na ontologia do domínio;
- ***permitir a evolução de conceitos de domínio e tarefas***: a Estação TABA deve oferecer suporte ao engenheiro do domínio para que este descreva as tarefas a serem consideradas no ADSOD (caso esta descrição não exista) ou defina novos conceitos na ontologia do domínio;
- ***permitir a construção de ferramentas específicas do domínio***: o ambiente de construção de ferramentas da Estação TABA deve poder utilizar a Teoria do Domínio na construção de ferramentas específicas do domínio;
- ***armazenar e indexar projetos desenvolvidos no domínio de acordo com a Teoria do Domínio***: a Estação TABA deve controlar o armazenamento e indexação de projetos desenvolvidos no domínio associados aos conceitos do domínio (definidos na ontologia do domínio e tarefas). Este requisito é necessário para que possa permitir ao engenheiro de software construir ferramentas para facilitar o entendimento do domínio pelos desenvolvedores (que usam o ADSOD) através do entendimento de como conceitos do domínio foram utilizados em outros projetos; e,

- ***permitir a especialização e instanciação de processos a partir de um processo padrão:*** a Estação TABA deve permitir, ao engenheiro de software, definir um processo de software para o ADS a ser instanciado a partir de um processo padrão que possua atividades para investigação do domínio.

ADSOD configurados pelo meta-ambiente TABA possuem, também, os seguintes requisitos:

- ***apoiar a instanciação da Teoria do Domínio***: o ADSOD deve apoiar a instanciação de ontologias definidas na Teoria do Domínio de acordo com a aplicação a ser desenvolvida;
- ***facilitar o entendimento do domínio e da tarefa***: o ADSOD deve oferecer ferramentas específicas que auxiliem o desenvolvedor de software no entendimento da tarefa realizada no domínio para o qual será desenvolvida a aplicação, através da interação do desenvolvedor com o domínio do problema. O ADSOD deve ajudar o desenvolvedor a decompor a tarefa direcionando-o para questões relevantes;
- ***oferecer apoio a diferentes tipos de usuário***: esse requisito refere-se ao fato de que, além dos desenvolvedores de software, os especialistas do domínio devem interagir com o ADSOD para introdução do conhecimento do domínio, ou seja, instanciação da ontologia do domínio;
- ***apoiar o acesso ao conhecimento:*** esse requisito refere-se à necessidade do ADSOD oferecer mecanismos de acesso ao conhecimento, definido na teoria do domínio, nas diferentes atividades realizadas pelos desenvolvedores de software;
- ***ser extensível:*** isto, é permitir que novas ferramentas sejam incorporadas a medida em que se tornem necessárias;
- ***incorporar arquiteturas de referência:*** o ADSOD deve permitir a definição de arquiteturas de software para os diferentes produtos desenvolvidos.

Com o atendimento a esses requisitos a Estação TABA passa a poder instanciar ADS a partir do processo de software, do domínio de aplicação ou de ambos. Até este momento a instanciação de ambientes na Estação era possível apenas a partir do processo de software (FALBO, 1998).

A Figura 5.2 mostra a organização da Estação TABA considerando a instanciação de ADSOD, as facilidades que devem ser providas e os repositórios que devem ser criados, sendo destacado em cinza claro os aspectos que foram acrescentados na Estação TABA e em cinza escuro (funcionalidade Instancia ADS) a funcionalidade que foi alterada para permitir a construção de ADSOD.

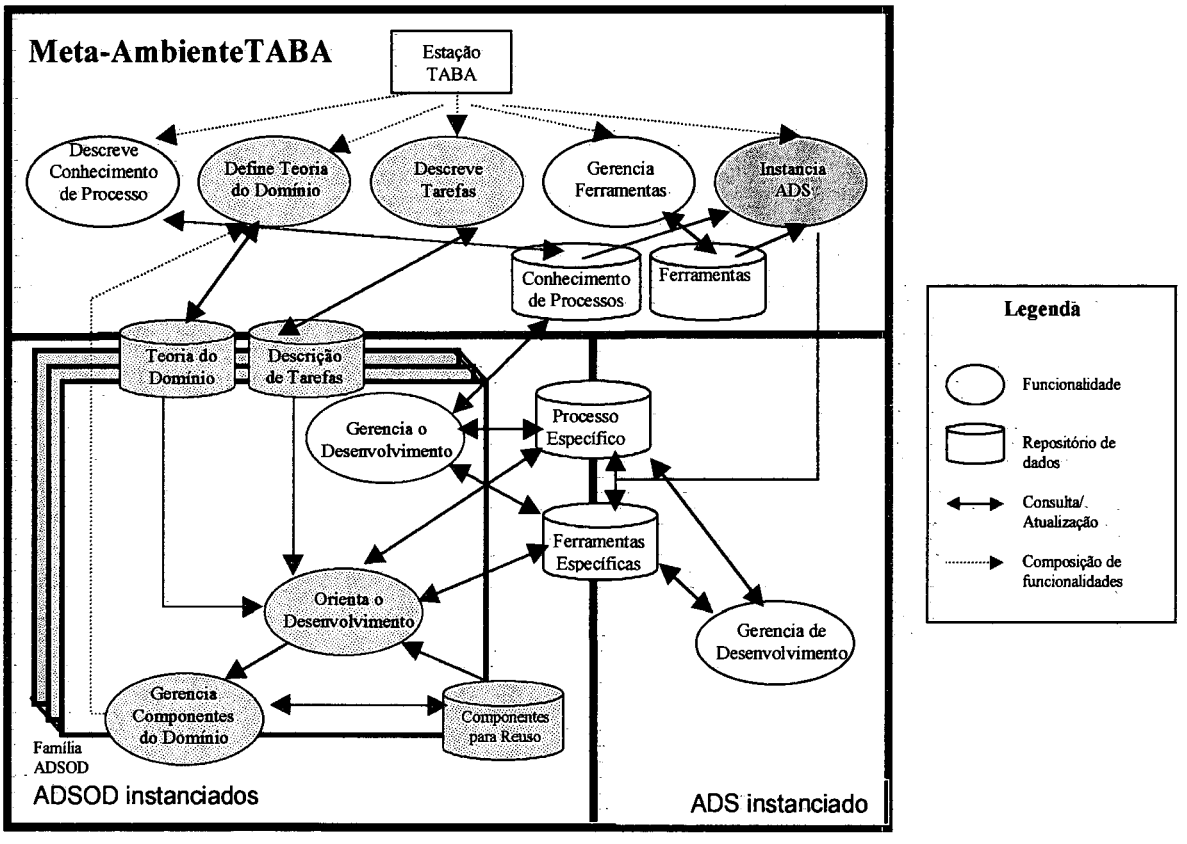

Figura 5.2 - Estação TABA e seus ADSs instanciados

O conjunto de ADSOD instanciados para um mesmo domínio e que possuem um processo de desenvolvimento definido a partir de um mesmo Processo Padrão forma uma ***Família de ADSOD***. Na mesma família um conjunto de ferramentas específicas para um domínio podem ser utilizadas por todos os ADSOD (um exemplo de ferramenta desse tipo é uma ferramenta de investigação do conhecimento do domínio). Além disso, dentro de uma mesma Família de ADSOD, os componentes de software produzidos em diferentes ADSOD poderão ser organizados e associados aos conceitos da Teoria do Domínio para poderem ser reutilizados por novas aplicações a serem desenvolvidas. Esse aspecto é importante não apenas pelos benefícios de se reutilizar software, mas também, por possibilitar o entendimento do domínio através de exemplos de projetos desenvolvidos.

As principais facilidades providas pela Estação TABA são portanto: apoiar a descrição do conhecimento de processo, apoiar a definição da Teoria do Domínio, apoiar a descrição de tarefas, gerenciar ferramentas, e, instanciar ambiente de desenvolvimento orientados a domínio ou não. *Apoiar a definição de conhecimento de processo* significa apoiar introdução de descrições gerais de atividades, processos, métodos, técnicas e demais aspectos necessários para auxiliar na definição de um processo de software. Da mesma forma, *apoiar a descrição de tarefas* implica em apoiar a descrição de tarefas a serem utilizadas no ambiente instanciado. O *apoio a definição da Teoria do Domínio* implica na disponibilização de uma ferramenta para permitir a introdução da Teoria do Domínio específico para que seja possível a instanciação de ADSOD. Essa ferramenta deve gerar módulos de conhecimento para o domínio específico produzindo um Servidor de Conhecimento[2] para o Domínio. *Gerenciar ferramentas* refere-se às facilidades da Estação TABA para integrar ferramentas, permitindo o seu uso na Estação e em seus ambientes instanciados (conforme definido por TRAVASSOS (1994)). *Instanciar ADS* refere-se à geração do ambiente instanciado, propriamente dito, de acordo com as características do processo previamente definidas.

A funcionalidade básica dos ADS instanciados, em geral, é gerenciar o desenvolvimento do software específico, ou seja, apoiar o acompanhamento, controle e alteração do processo de software durante o desenvolvimento no que se refere a inclusão de novas atividades e sub-atividades conforme definido por ARAUJO (1998). No caso de ADSOD, além da gerência do desenvolvimento, devem estar presentes também as funcionalidade "*orientar o desenvolvimento*", considerando as características do domínio e utilizando ferramentas específicas de investigação do domínio, e, "*gerenciar componentes do domínio*" para uma família de ADSOD específica. Esta função, não-automatizada, consiste da avaliação feita pelo engenheiro do domínio sobre o conhecimento e os componentes gerados em cada projeto com fins de evolução da Teoria do Domínio, necessidade de descrição de novas tarefas e identificação dos componentes reutilizáveis.

[2] Servidores de Conhecimento conforme apresentado no capítulo 2, são utilizados na Estação TABA para permitir a integração de conhecimento e sua utilização nas diferentes ferramentas construídas para serem utilizadas na Estação.

5.3 Implementação

Desde 1990, quando foi definida a Estação TABA (ROCHA, 1990), muitos trabalhos foram realizados no contexto do projeto abordando diversos aspectos desde a definição e implementação de aspectos relacionados à infra-estrutura da Estação (TRAVASSOS, 1994, ARAUJO, 1998, FALBO, 1998) até a definição e construção de ambientes específicos tais como os ambientes ORIXÁS (WERNECK, 1995) e MEMPHIS (WERNER *et al*, 1997) apresentados no capítulo 2. Foram, também, realizados vários estudos teóricos e de campo (WERNECK, 1990, CRISPIM, 1991, MOURA, 1992, ASSIS, 1992, MASSOLAR, 1993) e foram implementadas ferramentas específicas aos ambientes (AGUIAR, 1992, BARROS, 1995, CIMA, 1996, NIELBOCK, 1997, COELHO, 1997, VASCONCELOS, 1997, ARAÚJO, 1998).

Entre estes trabalhos, destaca-se o realizado por TRAVASSOS (1994) que definiu e implementou o modelo de integração de ferramentas da Estação TABA, a partir do qual os diversos ambientes e ferramentas foram construídos. Nesse modelo, TRAVASSOS (1994) organizou as classes conceituais da Estação em oito categorias (Figura 5.3): *(i)* a Estação TABA propriamente dita, que representa a essência do modelo possuindo apenas a classe Estação TABA responsável pela coordenação das tarefas do meta-ambiente; *(ii)* a categoria Conhecimento, que contém o conjunto de classes responsáveis pela integração do conhecimento sobre processo de desenvolvimento de software na Estação; *(iii)* a categoria Arquitetura, que relaciona as estruturas básicas para definição e construção de ferramentas e ambientes; *(iv)* a categoria Ambientes, que possui as classes que representam os ambientes instanciados; *(v)* a categoria Apresentação, que reúne os elementos responsáveis pela comunicação com do usuário com a Estação; *(vi)* a categoria de Dados, que possui as classes que organizam os dados utilizados na Estação; *(vii)* a categoria Ferramentas, que possui classes que representam as ferramentas desenvolvidas na Estação e que se encontram integradas nos ambientes instanciados; e, *(viii)* a categoria Controle, que possui as classes responsáveis pelo controle de execução das atividades do processo de desenvolvimento e pelo controle da gerência da qualidade no desenvolvimento de produtos nos ambientes instanciados. A última versão do modelo completo pode ser encontrada em ARAUJO (1998).

Esse modelo, proposto por Travassos, foi implementado em Eiffel, na plataforma Unix da *Sun Workstation*, por representar uma tecnologia robusta e poderosa. Os

diversos trabalhos realizados, posteriormente, na Estação TABA implicaram na inclusão de novas classes nas oito categorias procurando manter o aspecto de integração entre os diferentes componentes. No entanto, apesar de ser uma plataforma adequada para trabalhos de pesquisa, as idéias implementadas na Estação TABA não puderam, na maioria dos casos, ser realmente utilizadas para experimentações em ambientes reais pela falta de portabilidade do código para plataformas mais acessíveis e comumente utilizadas (como a plataforma de microcomputadores). Além disso, a linguagem Eiffel tem suporte técnico de difícil acesso e não é amplamente conhecida (o que dificulta, por exemplo, esclarecimentos de problemas de implementação em listas de *newsgroup*), e apresentou vários problemas com relação à interface com o usuário.

Considerando os problemas relatados, acima, e a filosofia de desenvolvimento de ambientes padronizados para organizações e para domínios específicos, apresentados no início desse capítulo, a equipe envolvida no projeto decidiu re-implementar a Estação TABA em uma plataforma que, além de permitir o desenvolvimento de pesquisas e de ADS, facilitasse a realização de experiências de uso desses ambientes nas organizações para os quais são concebidos. Essas considerações levaram à escolha da plataforma de microcomputadores e ao uso da linguagem C++.

O volume de trabalho, dado que são dez anos de pesquisas realizadas na Estação TABA, foi uma das principais dificuldades para a re-implementação. Os diferentes trabalhos realizados são relevantes para o contexto de ambientes de desenvolvimento de software e de alto interesse prático. No entanto, não seria possível, de imediato, re-implementar o resultado de todos esses projetos. Decidimos, portanto, re-implementar, inicialmente, apenas, as funcionalidades de especificação, instanciação e execução de ambientes configurados, por serem estas as fundamentais para se ter a infra-estrutura necessária para construção de ADSOD e, consequentemente, para demonstração das idéias deste trabalho.

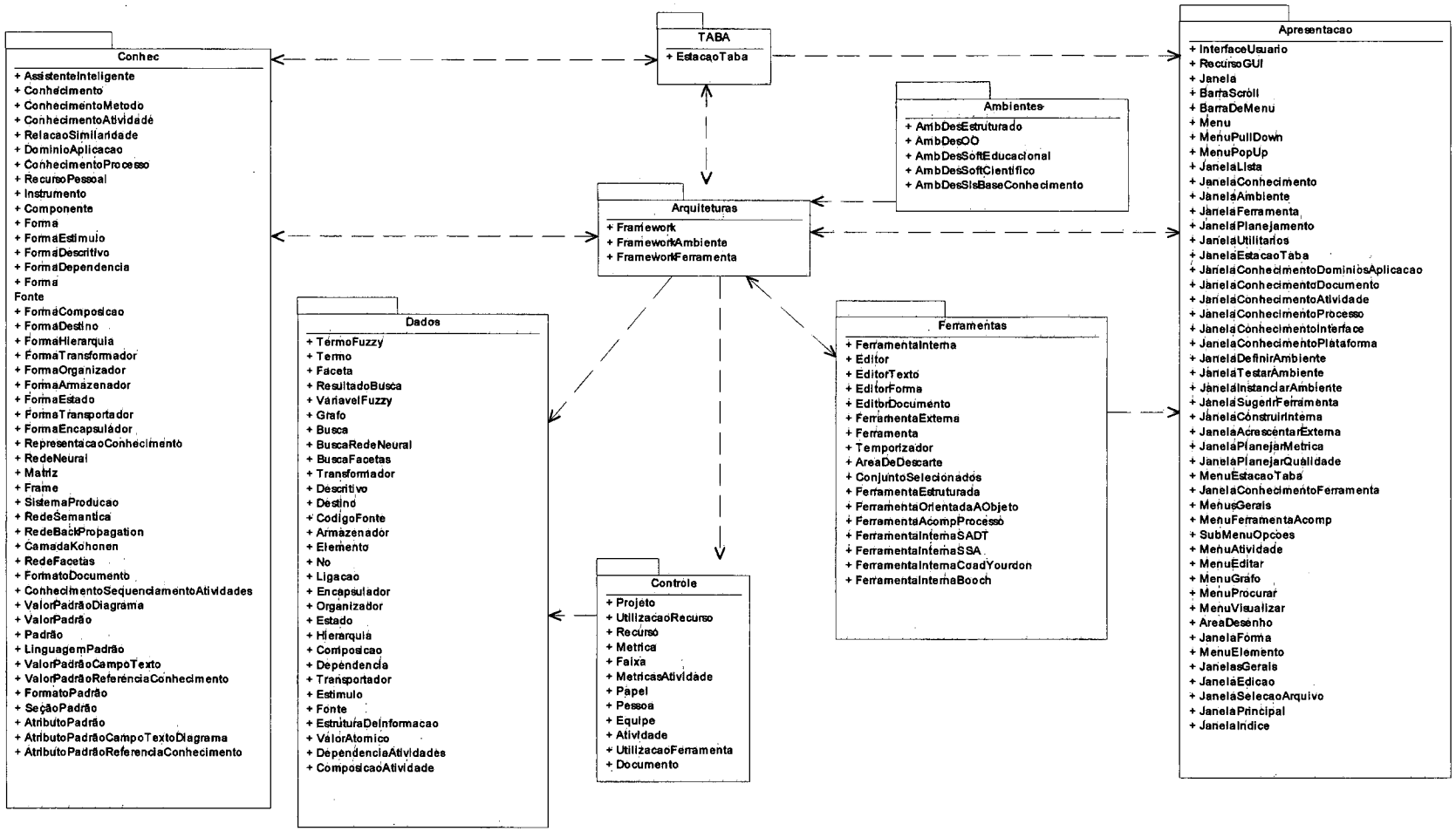

Figura 5.3 – Modelo da Estação TABA (ARAUJO, 1998)

O modelo da Estação TABA foi, então, revisto e avaliado procurando identificar esse núcleo a ser re-implementado. Todas as categorias da modelagem conceitual foram revistas sendo mantidas, apenas, as classes responsáveis por atender a essas funcionalidades. A categoria *Estação TABA* foi mantida sendo, apenas, redefinidos os atributos e métodos da classe principal. Na categoria *Conhecimento* foram mantidas as classes de conhecimento de processo para permitir a especificação de processo; e, a arquitetura de Servidores de Conhecimento proposta por FALBO (1998) já que os ADSOD devem possuir conhecimento sobre o domínio integrado no ambiente. A categoria *Ambientes* foi reorganizada para considerar a nova característica da Estação TABA de poder instanciar ADS orientados a domínio ou não. Nessa mesma linha, a categoria *Ferramentas* foi reorganizada para considerar o grupo de ferramentas genéricas para qualquer ambiente de desenvolvimento de software (como ferramenta CASE de um determinado método de análise de requisitos) e o grupo de ferramentas específicas de ADSOD. A propriedade de serem ferramentas internas[3] ou externas[4] à Estação foi acrescentada como atributo da super-classe ferramenta. A categoria *Dados* foi mantida devendo possuir as classes de dados utilizados na Estação. A categoria *Controle* for mantida tendo sido atualizada com o último trabalho de definição de processo (FALBO, 1998). A categoria *Arquitetura* foi excluída inicialmente do modelo devendo ser considerada em trabalhos futuros que tenham como objetivo o apoio a construção de ferramentas e a integração de arquiteturas de referência. Os aspectos de construção de Ambientes foram reorganizados na categoria de ambientes. Finalmente, a categoria *Apresentação* foi excluída do modelo conceitual, passando a fazer parte do modelo de projeto por se tratar de aspectos de interface. Todo o modelo foi documentado utilizando UML (*Unified Modeling Language*)(BOOCH *et al.*, 1997). Dessa forma, as categorias[5] passaram a ser denominadas pacotes para uniformização com a nomenclatura da UML.

A partir dessa modelagem incial, foram feitas as alterações para contemplar as novas características necessárias para a definição e instanciação de Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD) e a consideração de processo padrão apresentadas anteriormente. Essas alterações serão apresentas em

[3] Ferramentas internas são ferramentas desenvolvidas na própria Estação TABA (TRAVASSOS, 1994)
[4] Ferramentas desenvolvidas fora da Estação TABA e posteriormente integrada (TRAVASSOS, 1994)
[5] O termo categoria é definido pelo método Booch (1994) utilizado anteriormente para modelagem na Estação TABA

detalhes na seções seguintes. O modelo atual da Estação TABA está apresentado no Anexo 1. A Figura 5.4, mostra o diagrama de pacotes desse modelo.

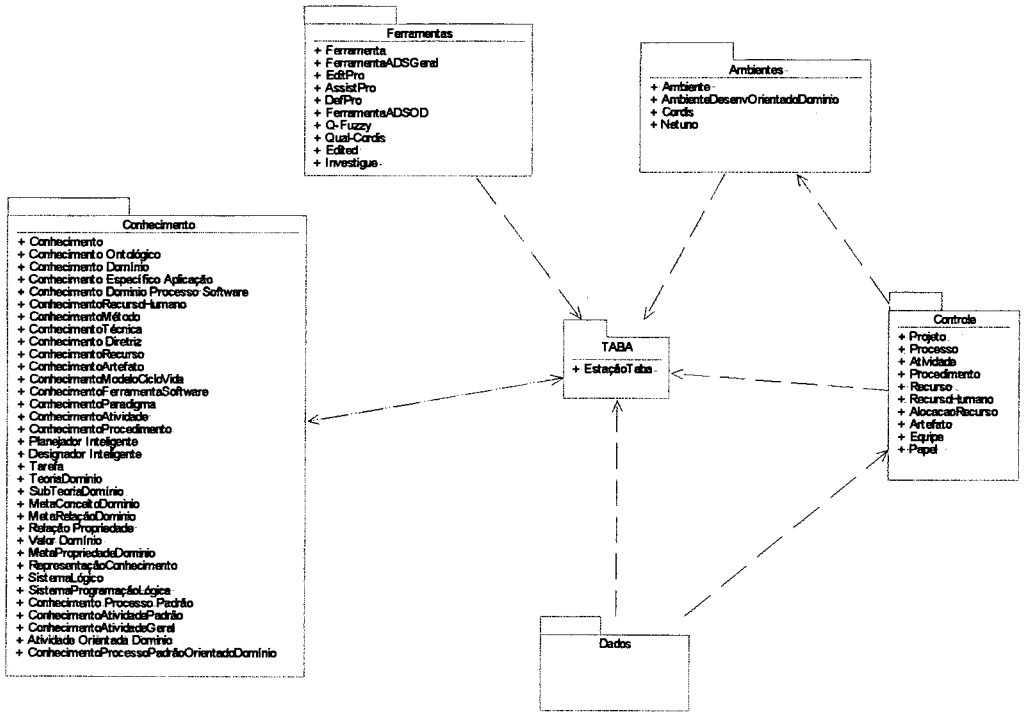

Figura 5.4 – Modelo Atual de Pacotes da Estação TABA

A re-implementação constou, então, de duas partes: a primeira visou apoiar a definição de um processo de software e a segunda, teve o objetivo de automatizar a instanciação de um ADS utilizando o processo de desenvolvimento.

A partir dessa re-implementação foram implementadas ferramentas para apoiar a definição de processo, a definição da Teoria do Domínio e de tarefas. Com essas três ferramentas implementadas é, então, possível instanciar um ADSOD.

No que se refere aos ambientes instanciados, foi implementada uma ferramenta de avaliação da qualidade (*Q-fuzzy*) que pode ser customizada para qualquer domínio específico. A seguir serão descritas, em maiore detalhes, cada uma destas ferramentas.

5.3.1 Ferramentas para Definição do Processo de Desenvolvimento

A definição de um processo de software na Estação TABA é feita considerando um conjunto de classes de conhecimento sobre processo de software presentes no pacote Conhecimento. Dessa forma, para ser possível a definição de um processo, primeiro foi re-implementada a infra-estrutura de conhecimento definida na Estação TABA e a entrada do conhecimento de processo que pode ser realizada através do menu

Conhecimento na Estação TABA (Figura 5.5). As figuras 5.6, 5.7 e 5.8 mostram as entradas de conhecimento de ferramentas, atividades e ciclo de vida, respectivamente.

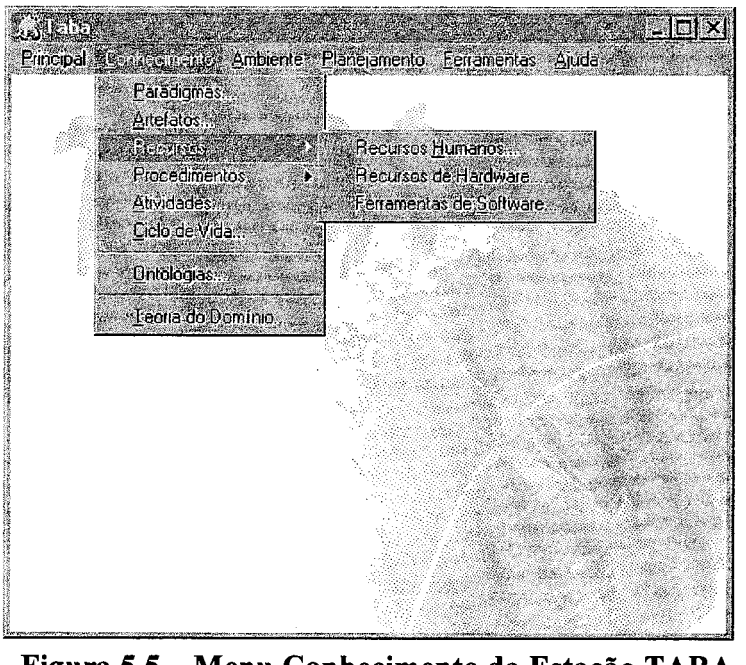

Figura 5.5 – Menu Conhecimento da Estação TABA

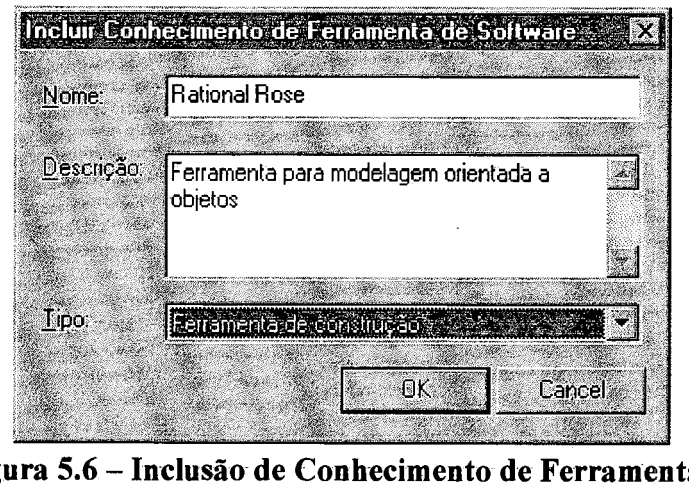

Figura 5.6 – Inclusão de Conhecimento de Ferramenta

A definição de um processo de software na Estação TABA pode ser realizada de três formas distintas, disponíveis na Estação através das seguintes ferramentas (Figura 5.9): EDIT-PRO, ASSIST-PRO e DEF-PRO.

EDIT-PRO é uma ferramenta para simples introdução de um processo na Estação. Esse processo deve ter sido previamente definido pelo engenheiro de software. Através do EDIT-PRO, o usuário inclui, inicialmente, as características do projeto que será desenvolvida. Em seguida, define o processo instanciado para o projeto escolhendo, do conhecimento sobre processo de software disponível na Estação, o modelo de ciclo de vida, as atividades, métodos, ferramentas, recursos, produtos e estabelecendo a sequência para realização de atividades. EDIT-PRO permite, ainda, customizar os nomes

das atividades e recursos para o processo específico que está sendo definido, tornando possível, assim, a utilização de nomes para as atividades já comumente usados na organização.

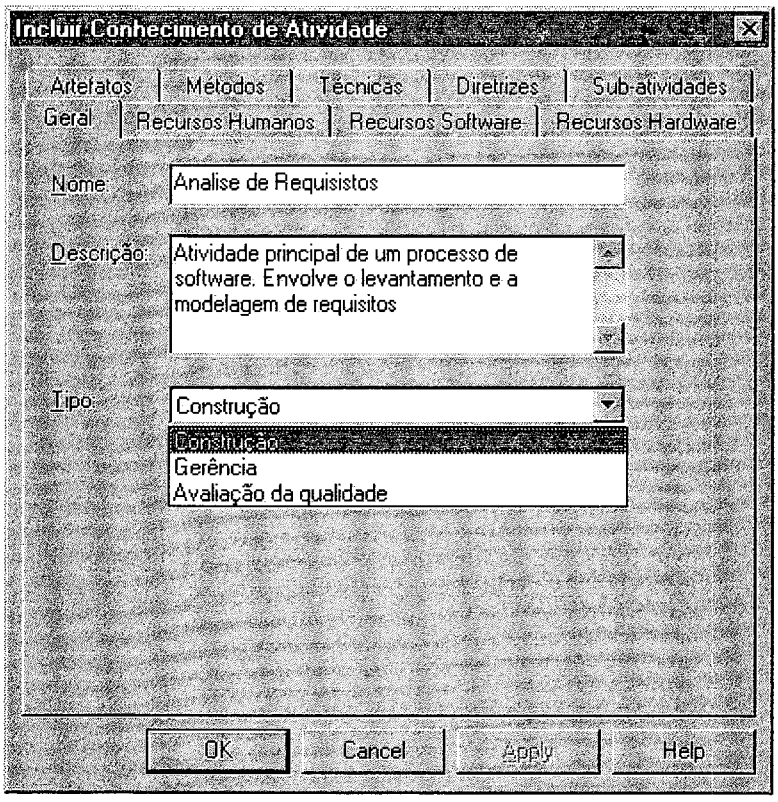

Figura 5.7 – Menu Conhecimento da Estação TABA

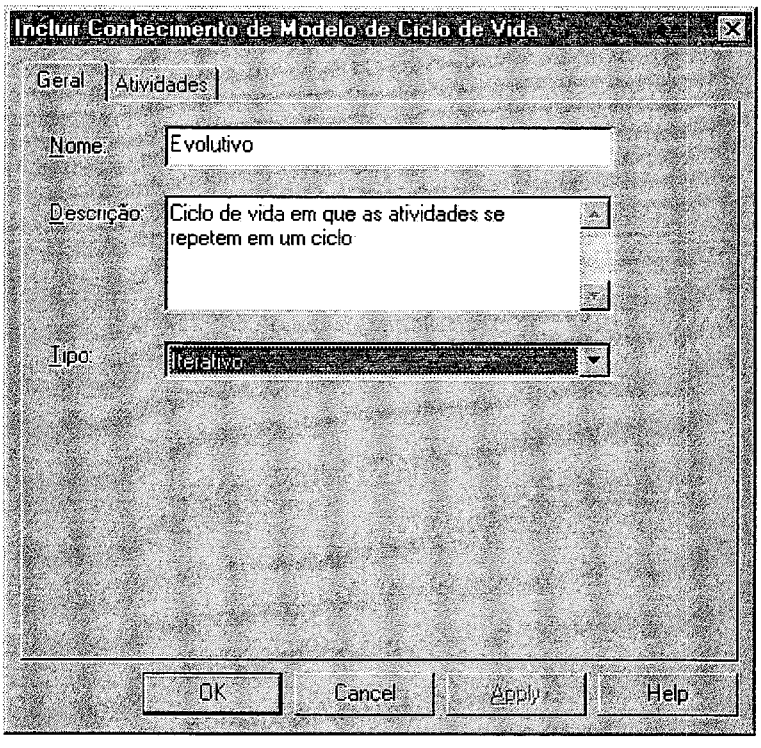

Figura 5.8 – Inclusão de Conhecimento de Ciclo de Vida

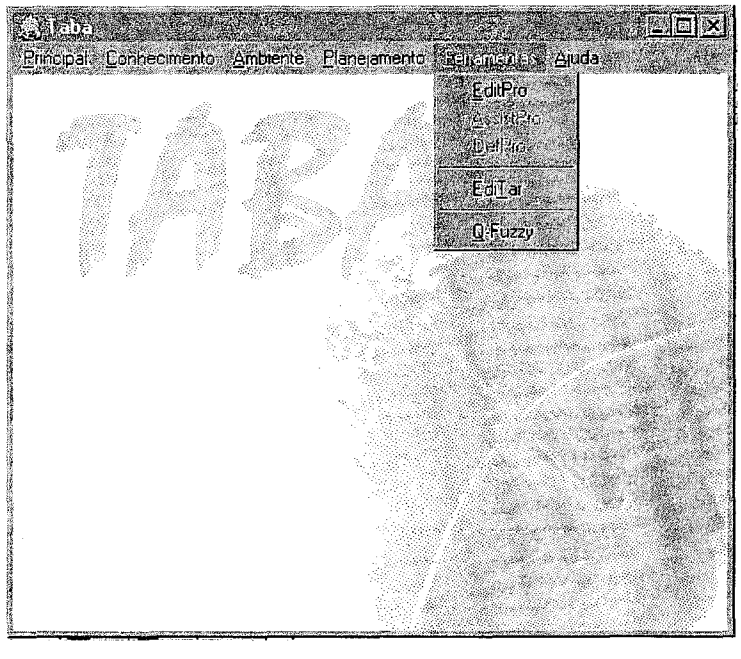

Figura 5.9 – Menu Ferramentas da Estação TABA

A segunda forma de definição de processo na Estação TABA implicou na re-definição e posterior re-implementação da ferramenta ASSIST-PRO (FALBO, 1999) apresentada no capítulo 2. O ASSIST-PRO é um assistente inteligente para definição de processos de software e foi o último trabalho realizado na versão da Estação TABA construída em Eiffel. A re-definição foi realizada para permitir que ASSIST-PRO utilize a filosofia de definição de processos a partir de um processo padrão, como apresentado neste capítulo. Essa redefinição implicou na inclusão de um conjunto de classes no pacote Conhecimento que considerou a definição de atividades, responsáveis e os produtos definidos no Processo Padrão. O conjunto de classes está apresentado na Figura 5.10. A Tabela 5.1 apresenta a descrição das classes. Com essa re-definição, ASSIST-PRO ao assistir o engenheiro de software na definição de um processo específico baseado em um processo padrão é capaz de garantir a presença de todas as atividades do processo padrão no processo instanciado gerado.

Finalmente, DEF-PRO é uma ferramenta proposta por MACHADO e ROCHA (1999) que estabeleceu um modelo geral para definição, especialização e instanciação de Processos de Software na Estação TABA. Esse modelo considera a norma ISO/IEC 12207 (1998), os modelos de maturidade CMM (PAULK *et al.*, 1995) e SPICE (EMAM *et al.*, 1998), as características da organização, tipo de software, tecnologia de

desenvolvimento e o tipo de ambiente que se deseja instanciar (orientados ao domínio ou não).

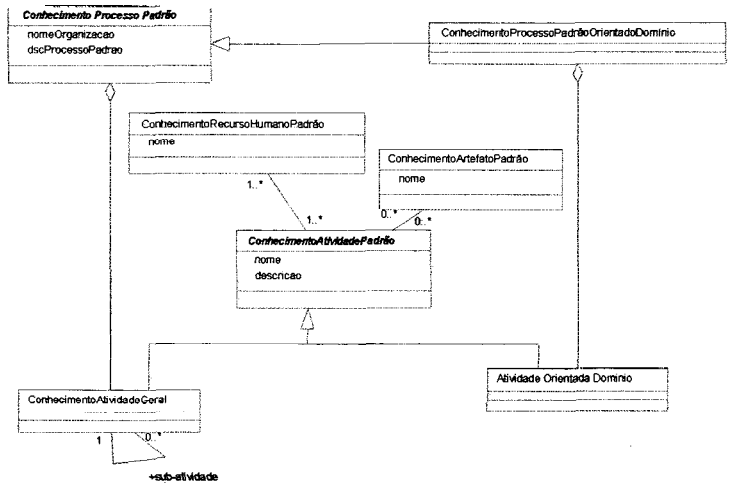

Figura 5.10 - Conjunto de classes sobre Conhecimento de Processo Padrão

Tabela 5.1 - Descrição das Classes de Conhecimento de Processo Padrão

Classe	Descrição
Conhecimento Processo Padrão	Conhecimento do processo padrão definido para uma organização. Deve ser composto do conjunto de atividades definidas
Conhecimento Processo Padrão Orientado ao Domínio	Processo padrão que possui atividades de orientação ao domínio
Conhecimento Atividade Padrão	Atividade do processo padrão realizada por responsáveis especificados e gera produtos (artefatos) específicos.
Conhecimento Atividade Geral	Atividade geral de um processo padrão.
Conhecimento Atividade Orientada a Domínio	Atividade específica de processos padrão orientado ao domínio. Representam a atividade investigação do domínio definida no capítulo 4.
Conhecimento Recurso Padrão	Responsável pela realização de uma atividade padrão.
Conhecimento Artefato Padrão	Produto de software (artefatos) de uma atividade padrão.

Neste momento, apenas, a ferramenta EDIT-PRO está disponível na Estação TABA, permitindo a construção de ADS e ADSOD. As duas últimas ferramentas estão em fase final de implementação.

5.3.2 Ferramenta para Definição da Teoria do Domínio

A Teoria do Domínio, conforme descrita no capítulo anterior, é um modelo que utiliza ontologias do domínio e relaciona os conceitos do domínio com tarefas pertinentes àquele domínio.

Para ser possível a incorporação de uma Teoria do Domínio na Estação TABA foi definida e implementada uma ferramenta interna denominada EDITED, um editor de Teoria do Domínio. A construção de EDITED compreendeu três etapas: *(i)* a inclusão de classes específicas para definição dos conceitos de qualquer domínio no pacote *Conhecimento* do modelo conceitual da Estação TABA, *(ii)* a concepção de como incorporar o conhecimento do domínio, editado através da ferramenta EDITED, na estrutura de integração de conhecimento da Estação TABA de forma a poder ser acessado nos ambientes instanciados e utilizado para construção de ferramentas, e *(iii)* a implementação do EDITED propriamente dita.

A primeira etapa implicou na definição de um meta-modelo, que considera as características da Teoria do Domínio, ou seja, suas sub-teorias, conceitos, características desses conceitos e suas relações com as tarefas. Essas classes fazem parte do pacote Conhecimento do modelo da Estação TABA. A Figura 5.11 mostra o conjunto de classes do meta-modelo de EDITED. Na Tabela 5.2 são descritos cada um dos conceitos desse modelo. Com a definição do meta-modelo é possível, então, incorporar os conceitos sobre qualquer domínio desejado.

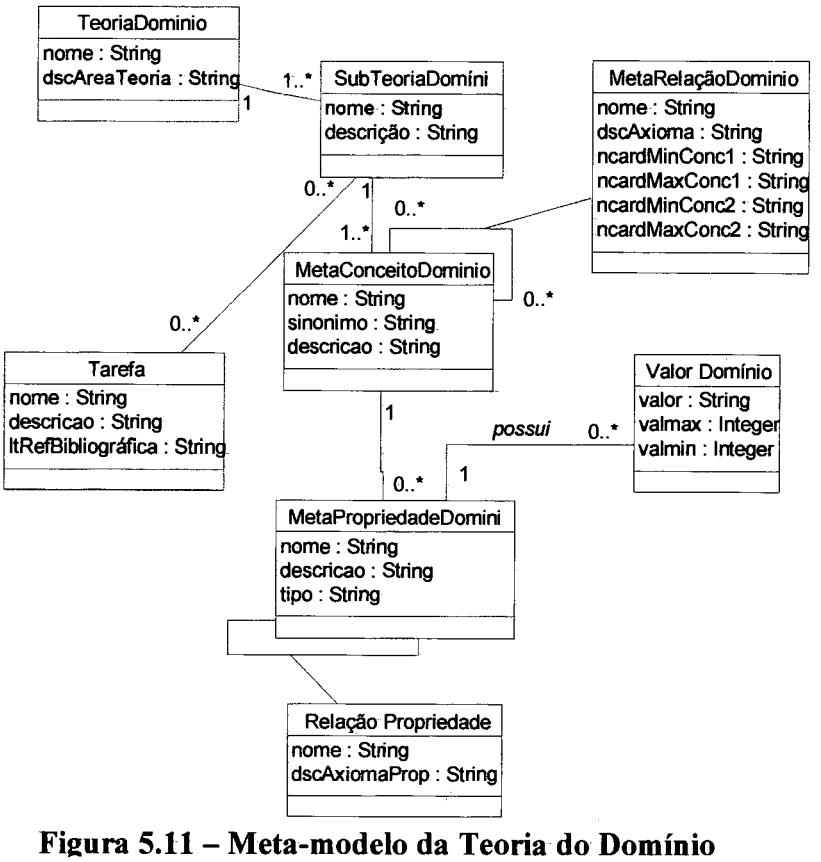

Figura 5.11 – Meta-modelo da Teoria do Domínio

A segunda etapa teve como objetivo o estudo de como tornar as ontologias editadas através da ferramenta EDITED disponíveis como Servidores de Conhecimento do Domínio. Servidores de Conhecimento foram propostos por FALBO (1998, 1999a) como forma de integração do conhecimento definido através de ontologias em ADS. A arquitetura de Servidor do Conhecimento, corresponde, basicamente à definição de módulos de conhecimento que representam ontologias e suas instanciações e que se tornam disponíveis para o compartilhamento e reuso do conhecimento (ver capítulo 2). A arquitetura geral do Servidor de Conhecimento foi implementada na versão anterior da Estação TABA e, portanto, teve que ser re-implementada. O modelo do Servidor de Conhecimento para contemplar a arquitetura (Figura 5.12) é composto do seguinte conjunto de classes: Servidor de Conhecimento, Conhecimento e suas especializações (Conhecimento Ontológico e Conhecimento do Domínio), representação do conhecimento e assistente inteligente. Na Tabela 5.3 são descritas cada uma dessas classes.

Tabela 5.2 – Descrição das Classes do meta-modelo do EDITED

Classe	Descrição
Teoria do Domínio	Contém as sub-teorias do domínio definidas para uma Teoria do Domínio.
Sub-teoria do Domínio	Conjunto de conceitos de um domínio semanticamente relacionados. Ex.: para cardiologia definimos as sub-teorias de anatomia, patologia, terapia e achados médicos.
Meta Conceito Domínio	Os conceitos de um domínio que pertencem a uma sub-teoria. Ex.: sintomas para a sub-teoria de achados.
Meta Relação Domínio	A relação entre os conceitos. Numa ontologia definimos as relações explicitamente, incluindo as relações do tipo *is_a* e *part_of*. Todas as relações são definidas através de axiomas formais que explicitamente definem uma semântica (conhecimento) para as relações. Ex.: patologia manifesta sintomas
Meta Propriedade	Uma propriedade de um conceito. Numa ontologia nos preocupamos com o requisito de compromisso ontológico mínimo para descrever as propriedades. Ex.: intensidade para o conceito sintoma.
Valor Domínio	Valor de uma propriedade
Relação Propriedade	Relação entre as propriedades, definindo axiomas

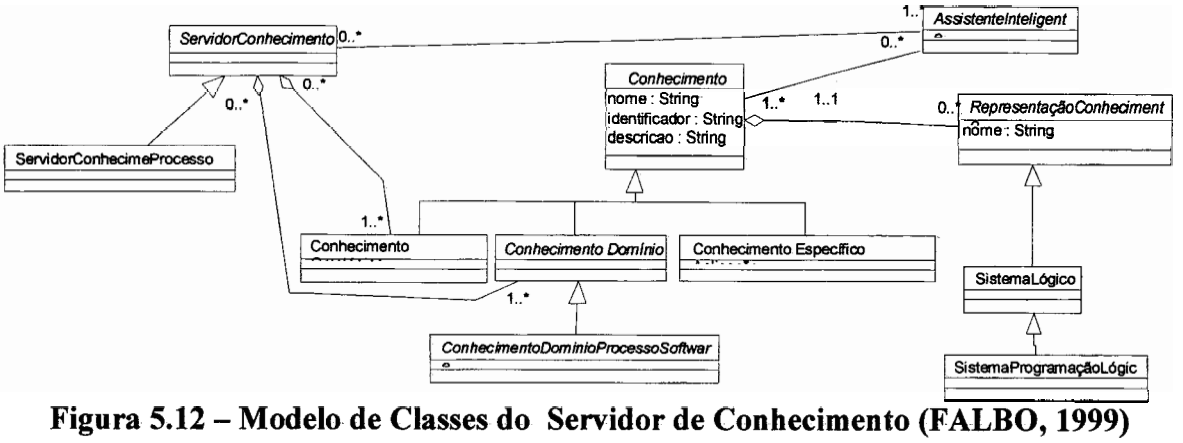

Figura 5.12 – Modelo de Classes do Servidor de Conhecimento (FALBO, 1999)

Tabela 5.3 – Descrição de Classes do Servidor de Conhecimento

Classe	Descrição
Servidor Conhecimento	Classe que agrega a arquitetura dos Servidores de Conhecimento na qual os módulos de conhecimento são modelados como objetos da classe *Conhecimento* e os templates de resolvedores de problemas são modelados como sub-classes da classe *Assistente Inteligente*. Essa classe deve ser especializada para Servidor de Conhecimento de Domínio específico (na Figura 5.12, por exemplo, foi definida a classe Servidor de Conhecimento de Processo apara o conhecimento de processo na Estação TABA)
Conhecimento	Classe responsável pelo armazenamento e acesso ao conhecimento na Estação TABA que deve ser especializada para contemplar os vários tipos de conhecimento desejado. Para o servidor de conhecimento foram criadas as sub-classes *Conhecimento Ontológico e Conhecimento Domínio*.
Conhecimento Ontológico	Classe que contém objetos referentes a conhecimento sobre axiomas de uma ontologia
Conhecimento Domínio	Super-classe de classes utilizadas para mapear os pacotes de conhecimento consideradas nas ontologias definidas. Deve ser especializada para cada domínio considerado (na Figura 5.12 por exemplo foi definida a classe de Conhecimento do Domínio de Processo na qual todas os conceitos sobre conhecimento de processos são organizados (ver modelo completo no Anexo 1)
Assistente Inteligente	Conjunto de resolvedores de problemas para serem utilizados na construção de assistentes inteligentes usando o conhecimento de ontologias descritas nas classes Conhecimento Domínio e Conhecimento Ontológico
Representação do Conhecimento	Classe para prover a Estação TABA a capacidade de representar o conhecimento utilizando diferentes tecnologias de representação. Atualmente tem-se disponível a tecnologia de representação de sistema lógico com a integração do SWI-Prolog representado na classe *Sistema Programação Lógico*. Embora não faça parte da arquitetura Servidor e sim da estação como um todo, essas classes são mencionadas na Figura 5.12 por serem utilizadas pela Servidor de Conhecimento para descrever o conhecimento numa tecnologia de representação.

A integração da ontologia descrita por EDITED é feita através da construção de um Servidor do Domínio, específico para o domínio em questão. Isso implica na criação de uma sub-classe para a classe *Servidor de Conhecimento* para o Servidor de Conhecimento específico que se deseja construir, uma subclasse da classe *Conhecimento Domínio* para mapear o conhecimento das ontologias do domínio e criação de objetos na classe *Conhecimento Ontológico,* para representar os axiomas (restrições). A sub-classe *Conhecimento Domínio,* específica, deve possuir especializações com o conjunto de conceitos das ontologias do domínio considerado. Dessa forma, o Editor de Teoria de Domínio deve gerar tanto as classes de especialização do servidor específico e superclasse de conhecimento do domínio, quanto classes para cada um dos conceitos definidos.

Para exemplificar, suponhamos que para uma Teoria do Domínio X tenham sido inseridos os conceitos *a* e *b*. EDITED cria a classe *Servidor de Conhecimento X*, a classe de *Conhecimento Domínio C* e suas sub-classes *a* e *b* (Figura 5.13). Classes do domínio, construídas dessa forma, permitem que os conceitos da ontologia possam ser intanciados para gerar aplicações específicas (FALBO, 1998). Todas as restrições entre

os conceito *a* e *b*, são mapeadas para a classe Conhecimento Ontológico. As relações do tipo *subtipo* e *partede* entre dois conceitos são geradas automaticamente por EDITED. As demais restrições devem ser entradas pelo usuário em Prolog. Nessa nova versão da Estação Taba foi re-implementada a forma de representação através de sistema de programação e lógica utilizando uma máquina Prolog externa, o SWI-Prolog (WIELEMAKER, 1998), encapsulada na classe Sistema Programação em Lógica conforme definido por (FALBO, 1998).

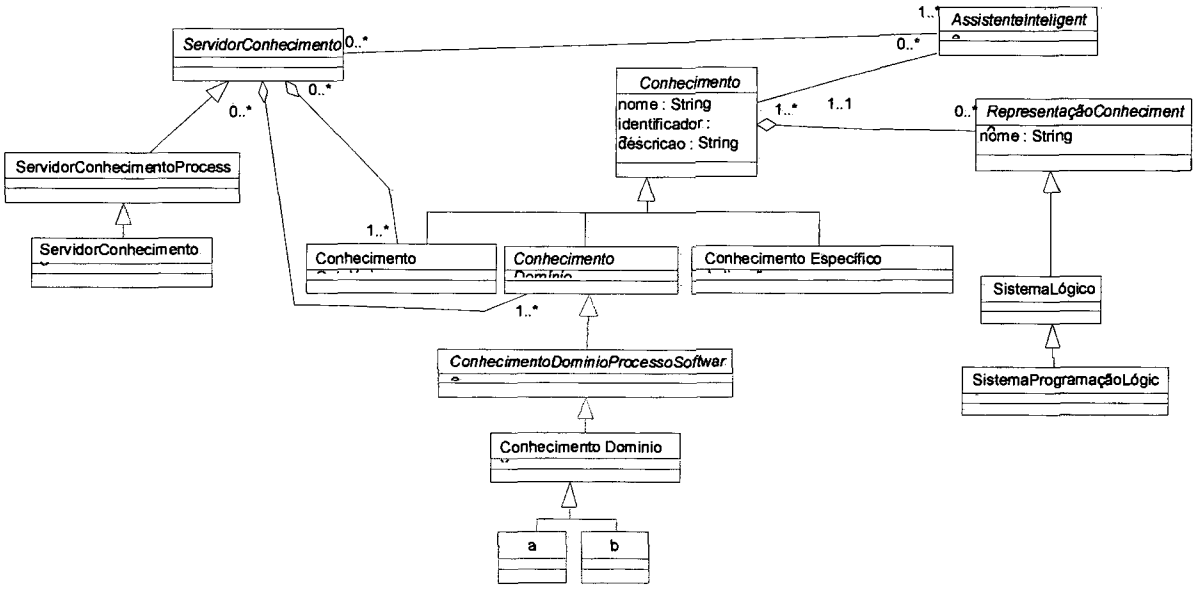

Figura 5.13 – Classes criadas depois de inserido a Teoria do Domínio X

Finalmente, com a infra-estrutura modelada e o mecanismo de integração do conhecimento estabelecido, foi implementada a ferramenta EDITED. Os requisitos de entrada de EDITED englobam todos os conceitos utilizados para definir uma ontologia, ou seja, o que uma ontologia deve possuir para ser útil em um ADSOD, e o mapeamento de conceitos com as tarefas. Dessa forma, os requisitos de entrada são: os conceitos que caracterizam o domínio, como os conceitos se interrelacionam, quais as propriedades mínimas desses conceitos, quais os eventos que ocorrem que modificam o valor de propriedades, que estados uma propriedade de um conceito pode possuir, quais as restrições entre os conceitos, a que sub-teoria cada conceito está relacionado, e, quais as tarefas relacionadas a cada sub-teoria.

As principais funções de EDITED são: *(i)* permitir a entrada de informações do domínio; *(ii)* permitir a entrada de restrições de forma informal e descritas em Prolog,

(iii) gerar o conhecimento ontológico no servidor, e, *(iv)* gerar as classes para cada conceito do domínio, como especialização da classe Conhecimento Domínio. A seguir descrevemos como cada uma dessas funcionalidades foi implementada em EDITED.

A edição da Teoria do Domínio pode ser feita através da opção correspondente no menu Conhecimento da Estação TABA (ver Figura 5.5). Selecionada a opção "TeoriadoDomínio" o engenheiro do ambiente tem disponível o EDITED (Figura 5.14).

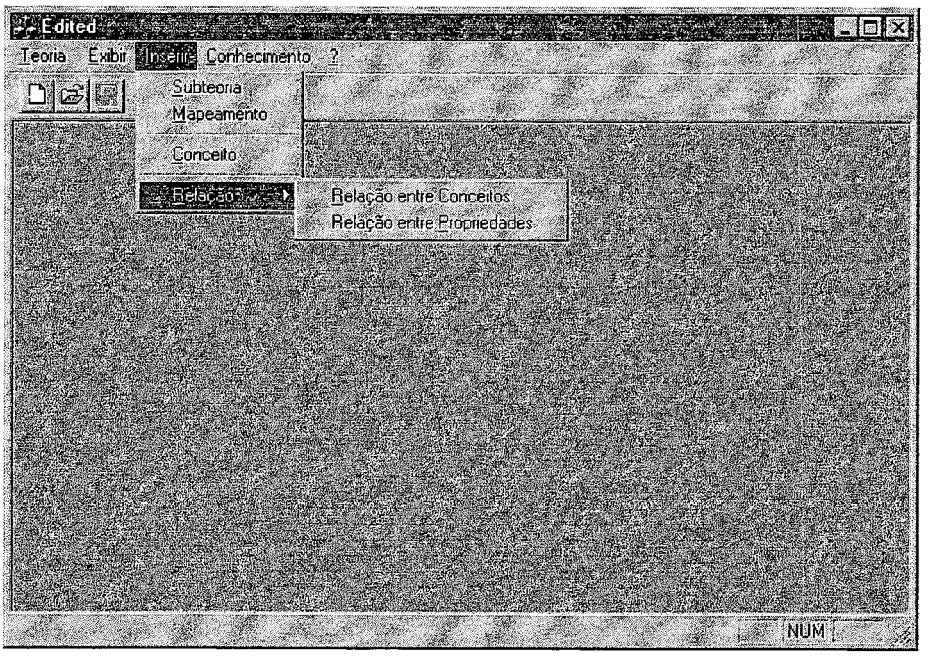

Figura 5.14 – Tela Principal do EDIED

Inicialmente, o engenheiro do ambiente deve entrar o nome e descrição da Teoria do Domínio que deseja editar (Figura 5.15). De forma semelhante, são inseridas as subteorias e suas descrições. Em seguida, é inserido, para cada sub-teoria, o conjunto de conceitos e suas propriedades (Figura 5.16). Após inseridos todos os conceitos, é informada a relação entre os conceitos e entre as propriedade, como mostra a Figura 5.17. As teorias são, então, então relacionadas às tarefas já previamente cadastradas (Figura 5.18) e o engenheiro do conhecimento insere o conhecimento ontológico (axiomas) para cada relação entre conceitos e propriedades (como mostrado na Figura 5.19). Os axiomas devem ser escritos em Prolog pois formarão os módulos de conhecimento, do Servidor de Conheciemnto, a serem utilizados por futuras ferramentas. Nesta versão, todos os axiomas devem ser inseridos pelo engenheiro do ambiente axiomas (independente de serem axiomas epistemológicos, ontológicos ou de consolidação).

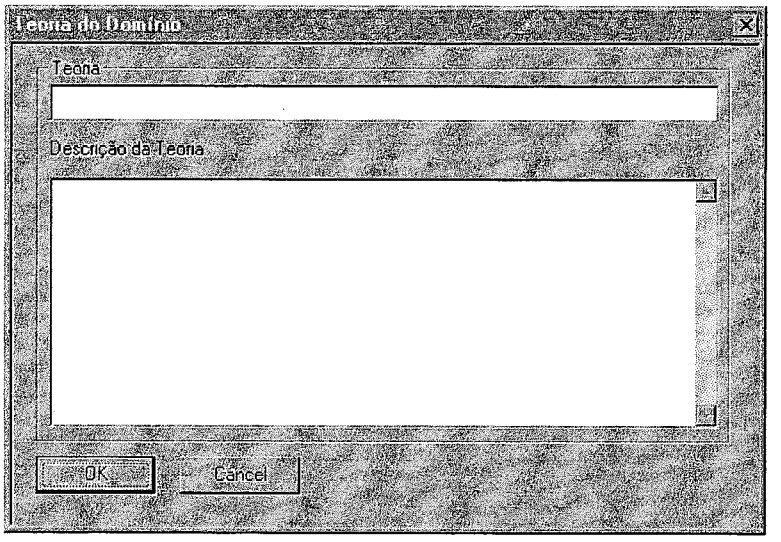

Figura 5.15 - Entrada da Teoria do Domínio

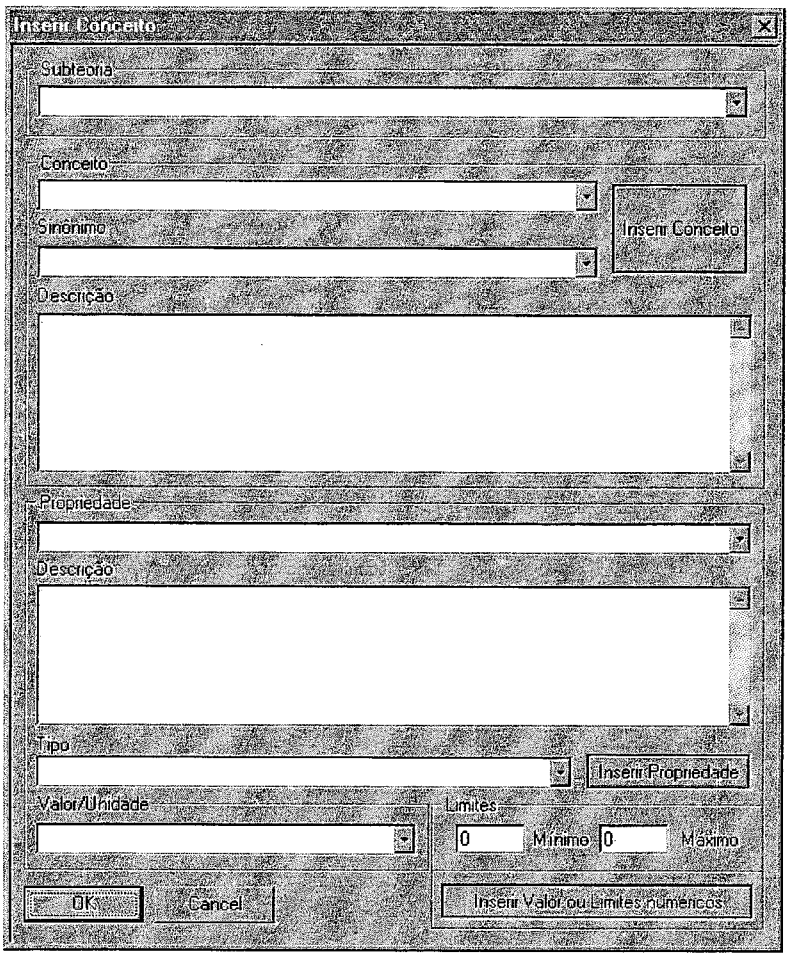

Figura 5.16 - Entrada de conceitos e propriedades

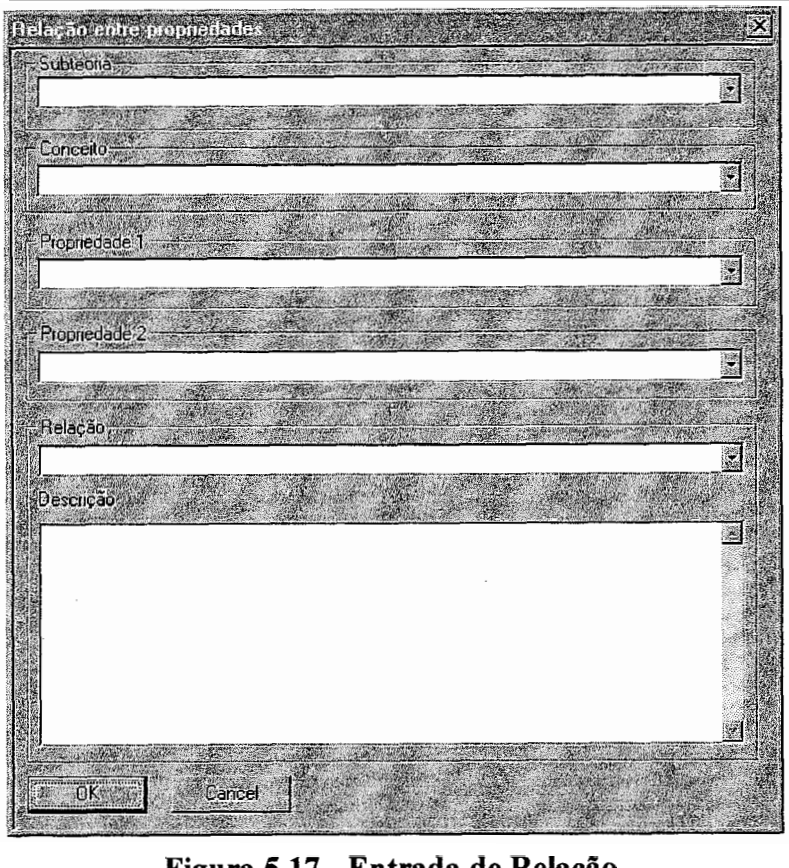

Figura 5.17 - Entrada de Relação

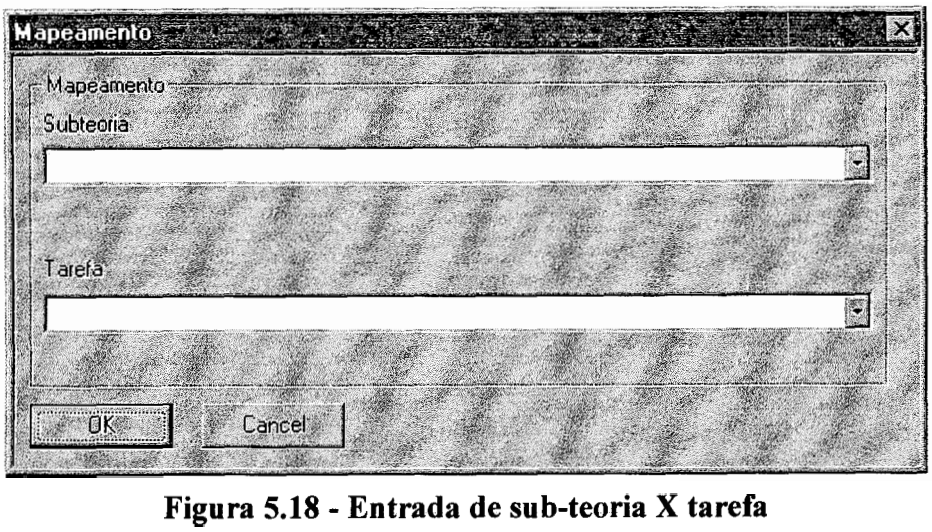

Figura 5.18 - Entrada de sub-teoria X tarefa

Depois de editado todo o conhecimento da Teoria do Domínio, o usuário solicita que sejam geradas as classes do domínio. EDITED cria, então, as classes, no modelo da arquitetura do Servidor de Conhecimento, conforme descrito anteriormente.

No capítulo 6 é mostrado um exemplo de entrada de uma Teoria do Domínio no EDITED.

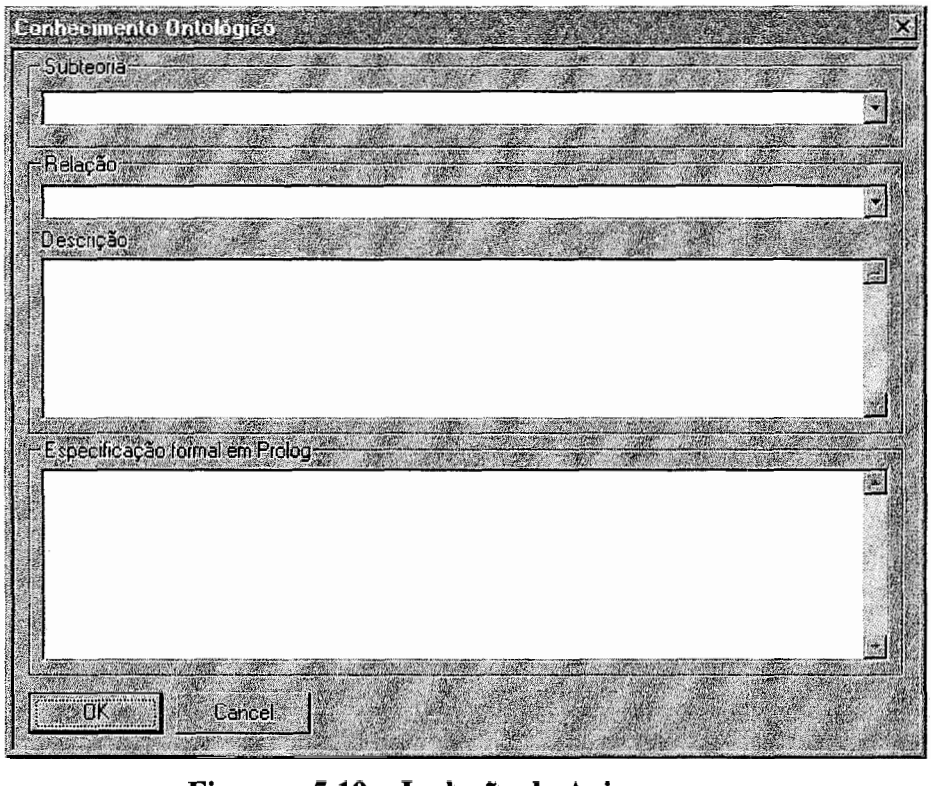

Figura – 5.19 – Inclusão de Axiomas

5.3.3 Ferramenta para Instanciação de ADS

A instanciação de ADS refere-se à geração de um ambiente que pode ser executado na Estação TABA ou independente da Estação. A geração de um ADS implica na criação e compilação do código referente à definição das classes que possuem informação sobre o processo e, no caso de ADSOD, as classes do domínio geradas por EDITED. A definição de como implementar a instanciação passou por várias decisões no que se refere à interface final dos ambientes instanciados, a persistência da informação e a integração das informações de controle do processo. Uma descrição detalhada sobre os problemas e as decisões desse processo pode ser encontrada em (ZLOT e SANTOS, 1999).

A instanciação do ADS está disponível na Estação TABA, na opção de menu *Ambiente* (Figura 5.20). Inicialmente, o usuário deve definir o ambiente que deseja instanciar selecionando a opção *DefinirAmbiente* (Figura 5.21). Deve, então, ser definido o nome do ambiente a ser gerado, uma descrição geral e caso se deseje pode-se esolher um ícone, já pré-desenhado, para ser utilizado no ambiente. Deve-se, também, ser escolhido o processo específico que se deseja instanciar. No futuro será também possível, caso o processo ainda não esteja definido, chamar uma das ferramentas de definição de processo (ASSIST-PRO ou EDIT-PRO). Finalmente, no caso de se desejar

instanciar um ADSOD, deve-se escolher a Teoria do Domínio que será utilizada no ambiente.

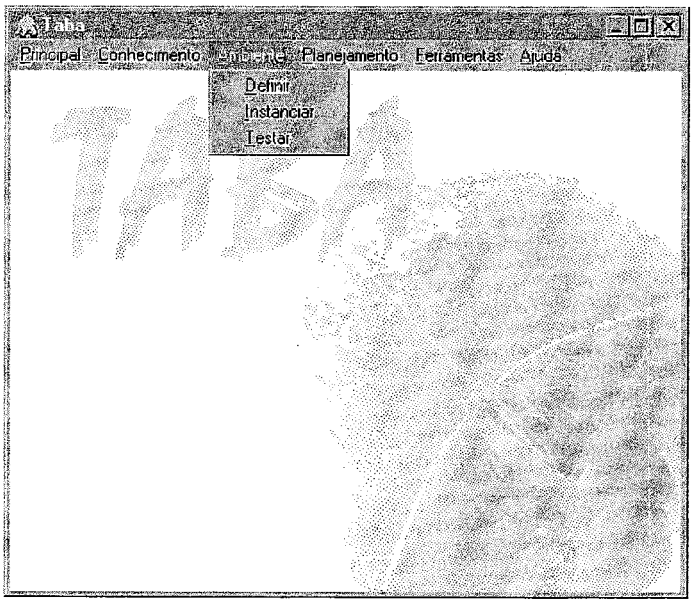

Figura 5.20 – Menu Ambiente da Estação TABA

Definido o ambiente, o usuário pode *Instanciar o Ambiente* (também definido na opção *Ambiente*) (Figura 5.22). A instanciação é iniciada após a seleção do nome do ambiente que se deseja instanciar e escolha do diretório de trabalho em que deve ser salvo. O usuário pode, ainda, escolher se deseja ter apenas o código fonte salvo no diretório especificado ou se deseja apenas o executável. A Estação TABA compila, dinamicamente, o código integrando as informações sobre o processo e, no caso de ADSOD, o código das classes geradas por EDITED.

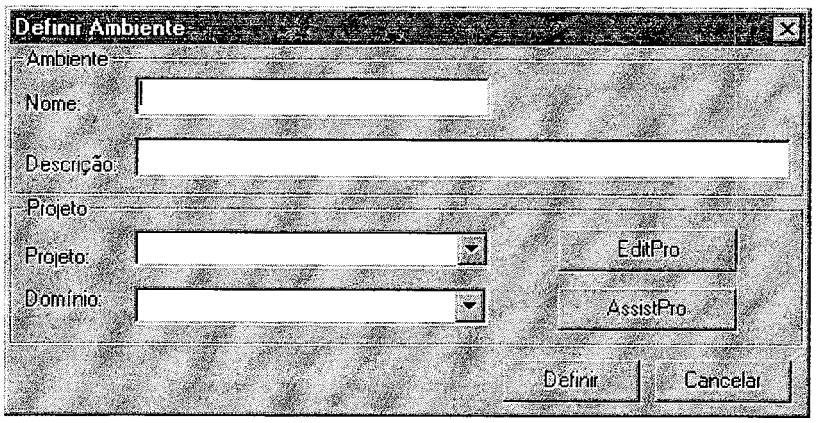

Figura 5.21 – Definição de Ambiente

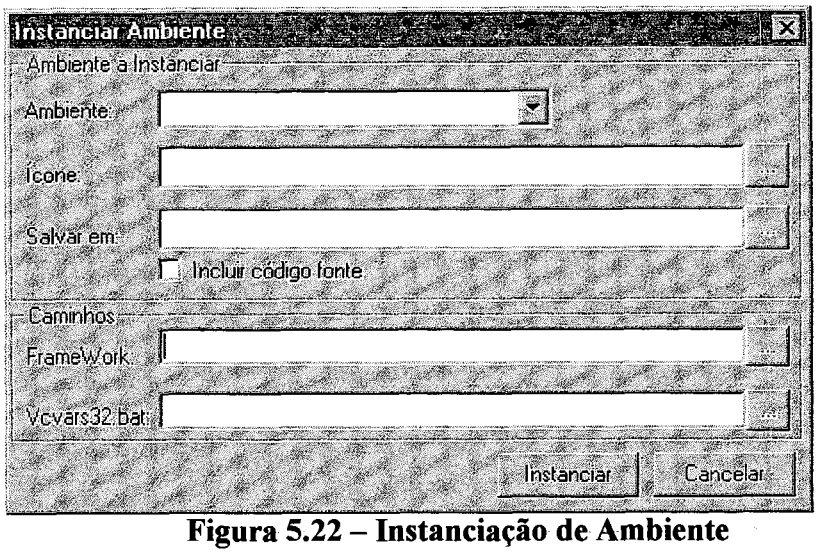

Figura 5.22 – Instanciação de Ambiente

Finalmente, o usuário pode *Testar o Ambiente* (Figura 5.23). Para isso, o usuário escolhe o ambiente que deseja instanciar e é disponibilizado o ambiente instanciado com o conjunto de atividades, ferramentas e descrições do processo definido.

Figura 5.23 – Teste de Ambiente Instanciado

No capítulo 6. Será mostrado a entrada de informações para definição, instanciação e teste de um ADSOD específico.

5.3.4 Ferramenta para Descrição de Tarefas

Para apoiar a descrição de Tarefas, foi implementada uma primeira versão do Editor de Tarefas descrito no capítulo anterior. Conforme definido cada tarefa é definida a partir de: *(i)* uma descrição de alto nível da tarefa; *(ii)* uma ontologia dos conceitos daquela tarefa específica, e, *(iii)* um conjunto de referências bibliográficas para as mesmas.

Nessa primeira versão, foram implementados os itens *(i)* e *(iii)*. Com a introdução dessas tarefas, mesmo de forma simplificada, é possível fazer o mapeamento com as sub-teorias do domínio na definição da Teoria do Domínio. Esse editor, denominado EDITAR, consiste da entrada de um texto descrevendo a tarefa e de uma lista de referências que podem ser pesquisadas para melhor esclarecimento da tarefa.

EDITAR é acessado através do menu Ferramentas da Estação TABA (ver Figura 5.9). A Figura 5.24 e Figura 5.25 mostram exemplos de descrições de tarefas inseridas através do EDITAR.

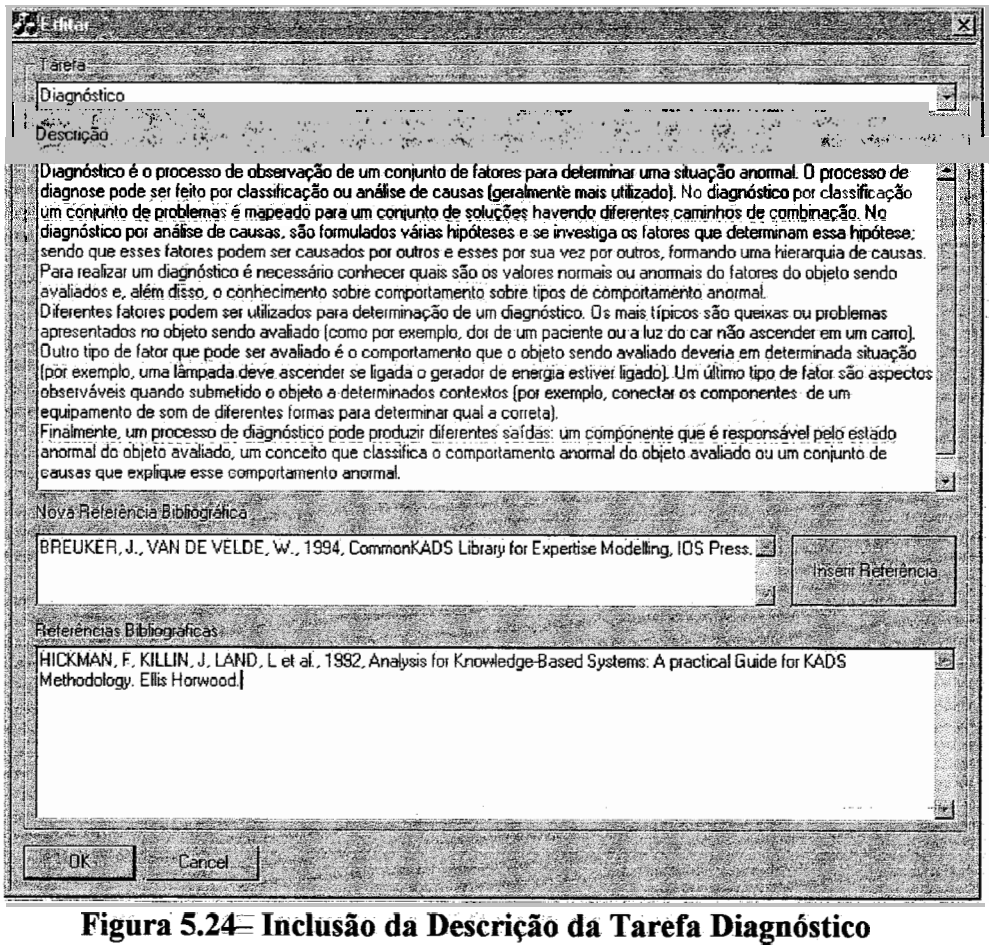

Figura 5.24 – Inclusão da Descrição da Tarefa Diagnóstico

5.3.5 Ferramenta para Avaliação da Qualidade

Para apoiar a avaliação da qualidade um ADS deve possuir ferramentas que considerem o planejamento do controle da qualidade, o levantamento de requisitos de qualidade e a avaliação dos produtos. As ferramentas para levantamento de requisitos e avaliação de produtos devem ser particularizadas, em um ADSOD, para considerarem atributos de qualidade e os tipos de software pertinentes àquele domínio. Nesse contexto, foi definida e implementada uma ferramenta para levantamento dos requisitos de qualidade de produtos de software denominada **Q-*fuzzy*** (CERQUEIRA *et al.*, 1999) que deve ser customizada para cada ADSOD instanciado na Estação TABA (OLIVEIRA *et al.*, 1999c). **Q-*fuzzy*** é uma ferramenta externa a Estação TABA existindo uma especialização do modelo, no pacote Dados, para capturar as características geradas.

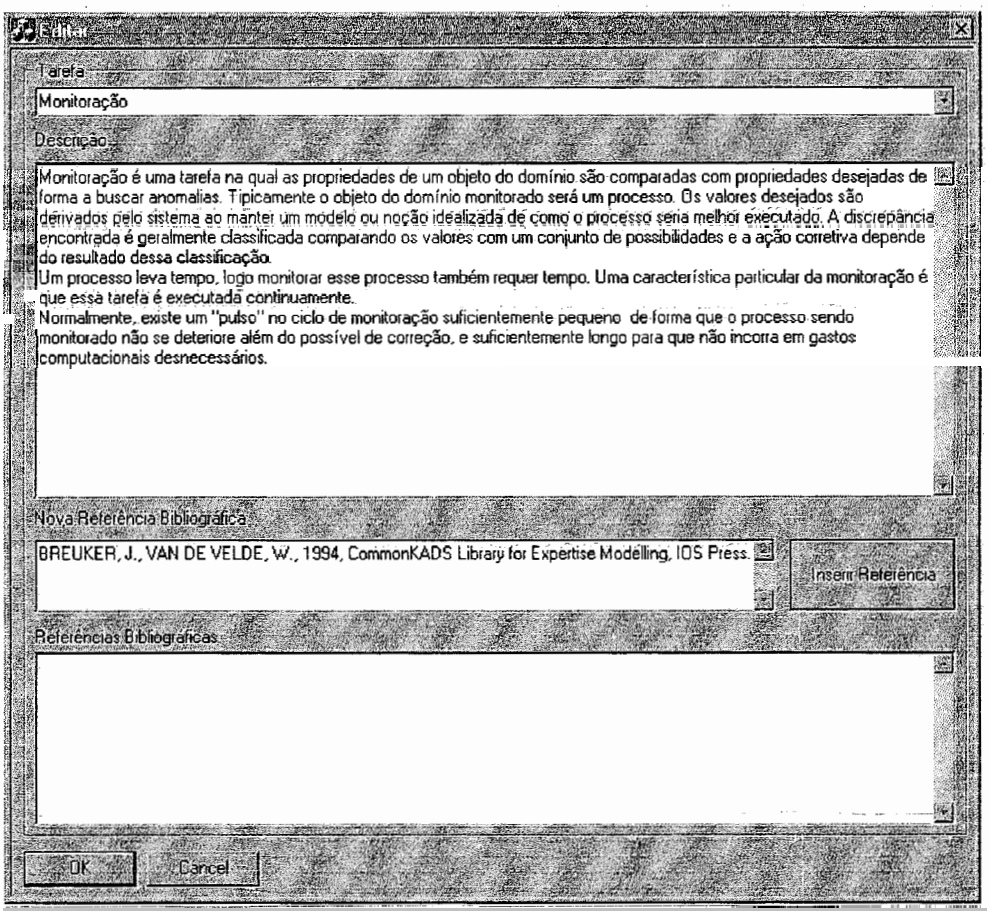

Figura 5.25 – Inclusão da Descrição da Tarefa Monitoração

Definir requisitos de qualidade significa identificar os atributos relevantes para um produto específico. Esta definição deve ser feita por usuários e desenvolvedores. Cada especialista envolvido na definição dos requisitos tem sua própria opinião e estima um grau de importância para cada um dos atributos, de acordo com sua percepção e seu nível de entendimento para a questão proposta. Além disso, a tomada de decisão é um processo de seleção de alternativas "suficientemente boas" para o objetivo desejado e envolve incertezas, sendo necessário lidar com informações imprecisas e vagas, levando em consideração diferentes visões e crenças das partes envolvidas (RIBEIRO , 1996). No final, as diferentes opiniões devem ser consolidadas representando o consenso dos avaliadores envolvidos na identificação dos requisitos de qualidade. BELCHIOR (1997) propôs identificar requisitos de qualidade e avaliar a qualidade de produtos considerando essas incertezas utilizando lógica *fuzzy*.

No que se refere ao levantamento de requisitos, BELCHIOR (1997) define quatro etapas para o modelo *fuzzy*: *(i)* identificar o objeto a ser avaliado e o conjunto de atributos de qualidade a ser considerado, *(ii)* escolha de especialistas e definição do seu perfil , *(iii)* determinação pelo especialista do grau de importância de cada característica

de qualidade identificada na primeira etapa e, *(iv)* tratamento dos dados coletados dos especialistas na avaliação de cada característica de qualidade para gerar um consenso entre os especialistas. O consenso das diferentes opiniões é feito a partir de uma função *fuzzy* específica envolvendo um cálculo do grau de concordância, a geração de uma matriz de concordância, a determinação da concordância relativa, o cálculo do coeficiente de consenso dos especialista e a determinação do valor *fuzzy* da característica de qualidade. A definição do perfil dos especialistas (etapa *(ii)*) é fundamental nesse processo pois procura-se definir um peso para o especialista através de um Questionário de Identificação do Perfil de Especialista (QIPE) específico para o especialistas de um mesmo grupo. O QIPE possui um conjunto de questões cujo objetivo é avaliar cada especialista e determinar sua importância definindo seu peso. O peso de um especialista é sempre relativo a outros especialistas. O total de escores de cada especialista é calculado segundo indicações contidas na apuração dos resultados do QIPE.

Dois aspectos são importantes nas duas primeiras etapas: o domínio e a organização no qual está sendo realizada a avaliação. O domínio, no que se refere à utilização de atributos de qualidade específicos para tipos de software particulares ou pertinentes ao domínio, e a organização no que se refere ao perfil dos especialistas a ser considerado. Dessa forma, Q-*fuzzy* deve ser customizada na Estação TABA no que se refere à esses dois aspectos. As duas últimas etapas são realizadas nos ADSOD instanciados. Q-*fuzzy* apoia, portanto, as diferentes etapas tendo sido feitas algumas adaptações na proposta de BELCHIOR (1997).

Para apoiar a primeira etapa, organizamos os atributos de qualidade em quatro grupos principais: atributos de qualidade relacionados na norma ISO9126 (ISO/IEC 9126 NBR ISO/IEC 13596, 1996); atributos gerais que consideramos necessários a qualquer software mas que não estão presentes na ISO 9126; atributos de qualidade de software considerando-se tecnologias específicas e atributos de software para domínios de aplicação específicos. Definimos que dentro de cada grupo, os atributos de qualidade devem estar organizados de acordo com o modelo ROCHA (1983) em *objetivos, fatores* e *sub-fatores*. As características e sub-características da ISO 9126 foram organizadas, respectivamente, como fatores e sub-fatores. É importante ressaltar que todos os atributos considerados são classificados em três objetivos de qualidade: *utilizabilidade*, que refere-se aos atributos de qualidade que consideram a utilização do software, sob as mais diversas formas; *confiabilidade conceitual* que se refere aos atributos que tornam o

produto confiável para seus usuários, do ponto de vista de seu conteúdo, e, *confiabilidade da representação*, que se refere aos atributos que tornam o produto confiável do ponto de vista de sua forma, de modo que este possa ser entendido e manipulado por diferentes pessoas durante sua vida útil. Os atributos de qualidade são, ainda, classificados em dois conjuntos: atributos relacionados à *qualidade externa* e atributos relacionados à *qualidade interna* do produto. Essa classificação foi definida para identificar que atributos devem ser avaliados por quais avaliadores no ADSOD instanciado: os atributos de qualidade externa, são avaliados pelos usuários finais, e os atributos de qualidade interna, são avaliados pelos desenvolvedores de software. Todos os atributos de qualidade da ISO 9126 e gerais já estão inseridos na ferramenta. Os atributos de tecnologia e domínio de ver inserido conforme necessidade do usuário. A

A partir da nossa experiência em avaliação da qualidade, entendemos que o levantamento de requisitos é, em geral, feito no nível de sub-fatores de qualidade por ser este o nível de entendimento dos usuários sobre a qualidade do produto, nesse momento. Dessa forma, Q-*fuzzy* apoia a identificação de requisitos de qualidade a partir da ponderação dos sub-fatores, em vez de critérios como proposto por BELCHIOR (1997).

Para apoiar a segunda etapa, Q-*fuzzy* permite a inclusão de questionários QIPE específicos para a organização. Estes questionários devem ser determinados levando em consideração os produtos que serão avaliados, buscando identificar o nível de especialidade dos avaliadores com relação ao produto. Desta forma, será possível identificar a importância relativa de cada avaliação realizada.

A terceira e quarta etapa referem-se à avaliação dos requisitos. Cada participante na avaliação determina o grau de importância desejado. Q-*fuzzy* permite, então, determinar os requisitos de qualidade e o grau de importância de cada requisito para o produto específico através do cálculo *fuzzy* que considera o grau de importância dado pelo especialista e o peso do respectivo especialista.

Os graus de importância (BELCHIOR, 1997, BELCHIOR *et al.*, 1997) e que podem ser utilizados pelos especialistas durante as avaliação, são definidos na Tabela 5.4. Para cada grau de importância existe uma *função de pertinência* $\mu(x)$ que pode ser expressa com o uso de um número *fuzzy* normal. A ferramenta, entretanto, não limita a avaliação dos usuários somente aos graus de importância definidos, permitindo que o usuário possa expressar sua avaliação definindo uma outra função de pertinência para o atributo

que está sendo analisado. Contudo, para que este recurso seja utilizado corretamente, é necessário que o usuário conheça o significado da função de pertinência e saiba como expressá-la.

Tabela 5.4 – Graus de Importância Utilizados pelos Avaliadores e suas Funções de Pertinência (BELCHIOR *et al.*, 1997)

Grau de Importância	Função de Pertinência expressa por um Número *Fuzzy* Normal
Nenhuma Relevância	$\tilde{N}_1 = (0,0; 0,0; 1,0)$
Pouca Relevância	$\tilde{N}_2 = (0,0; 1,0; 2,0)$
Relevante	$\tilde{N}_3 = (1,0; 2,0; 3,0)$
Muito Relevante	$\tilde{N}_4 = (2,0; 3,0; 4,0)$
Imprescindível	$\tilde{N}_5 = (3,0; 4,0; 4,0)$

Q-*fuzzy*[6] é acessada através do menu Ferramentas da Estação TABA (ver Figura 5.5). O engenheiro de software, então, define atributos de qualidade para um domínio específico, identificado se aquele atributo refere-se a qualidade externa ou interna (Figura 5.26). Para definir os questionários QIPE é necessário, incialmente, definir os grupos de participantes (Figura 5.27) identificando se são avaliadores que avaliam a qualidade externa ou interna do produto. Finalmente, o engenheiro de software define os QIPE para cada grupo (Figura 5.28). Dessa forma, a ferramenta está customizada para ser utilizada em um ambiente instanciado.

No próximo capítulo 6 será apresentada uma customização dessa ferramenta para um domínio específico. Será exemplificado, ainda, como a ferramenta é utilizada no levantamento de requisitos de qualidade.

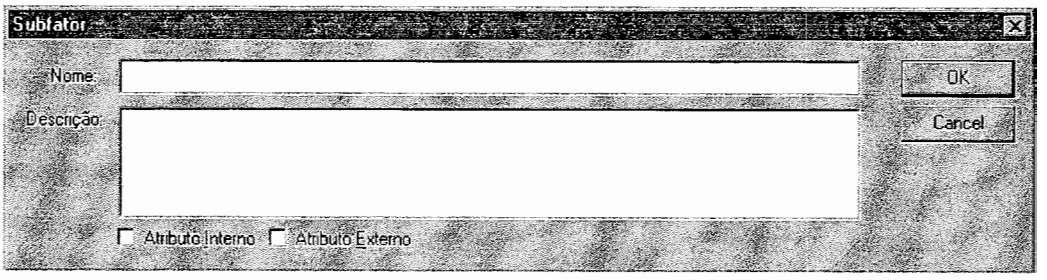

Figura 5.26 – Inclusão de Atributos de Qualidade

[6] O modelo de classes da ferramenta Q-fuzzy está apresentado no Anexo 5

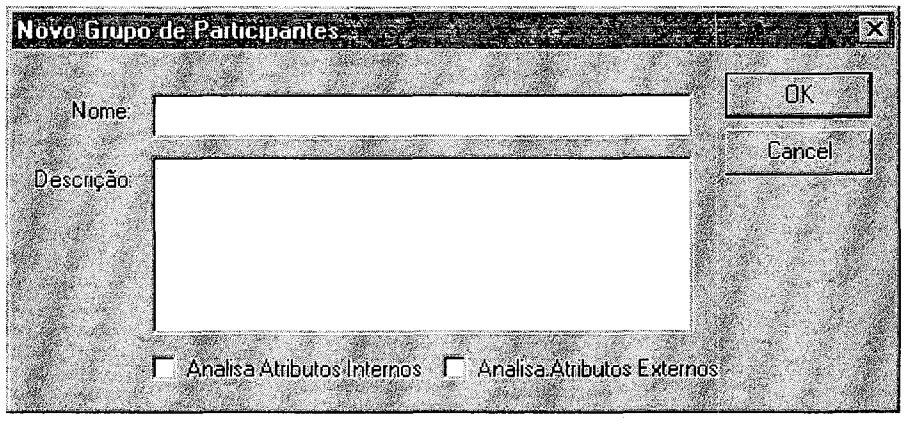

Figura 5.27 – Inclusão de Grupos

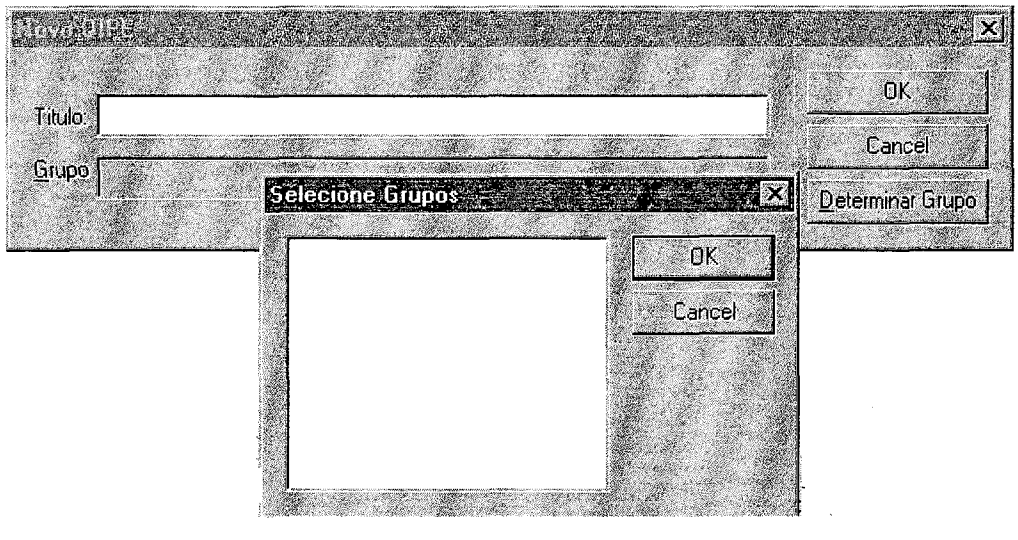

Figura 5.28 – Inclusão de QIPE

5.4 Considerações Finais

A construção de um ADSOD não é uma tarefa simples. Utilizando a infra-estrutura da Estação TABA, optou-se por não somente criar um ADSOD específico, e sim estendê-la para que possibilitasse a definição e construção de ADSOD para qualquer domínio desejado. Com esse enfoque, as principais extensões ao modelo da Estação TABA foram no que se refere ao meta-ambiente para permitir a definição das especificidades do domínio, ou seja, a Teoria do Domínio e ao processo de desenvolvimento particular para uma organização e considerando o domínio.

Nesse capítulo apresentamos as extensões realizadas no modelo para contemplar esse objetivo e as implementações realizadas. Por decisão de implementação, resolvermos re-codificar o núcleo da Estação TABA o que nos possibilitou alterar as funcionalidades especificar e instanciar ADS para considerar um domínio específico. No próximo capítulo, será apresentado um ADSOD específico construído a partir desta infra-estrutura.

Capítulo 6

CORDIS: Um Ambiente de Desenvolvimento de Software Orientado ao Domínio de Cardiologia

Este capítulo descreve a experiência de construção de um ADSOD para um domínio específico: cardiologia. CORDIS[1] é um ADSOD, definido e construído de acordo com a infra-estrutura e características apresentadas nos capítulos anteriores. A seguir, serão apresentadas algumas considerações sobre o domínio e a motivação para sua escolha, a teoria do domínio, o processo de desenvolvimento definido para o ambiente CORDIS, a implementação do ambiente e as ferramentas já desenvolvidas para o mesmo.

6.1 Considerações Iniciais

A medicina é considerada ciência e arte que tem como objetivo a modificação do estado anormal de um paciente através de alguma ação terapêutica real em vez de apenas se ter uma idéia. O problema fundamental de projetar e construir sistemas na área médica surge da necessidade de combinar todos os dados observáveis, que vão de processos fisiológicos físicos a químicos, até características éticas sutis e impressões clínicas vagas. O desenvolvimento de sistemas requer, portanto, um contínuo aprendizado desses aspectos pelos desenvolvedores de software, o que implica em muitas interações com os especialistas. Constatamos esse fato ao trabalhar, nos últimos cinco anos, no desenvolvimento de diferentes sistemas na Unidade de Cardiologia e Cirurgia Cardiovascular/Fundação Bahiana de Cardiologia da Universidade Federal da Bahia (UCCV/FBC). No desenvolvimento de um sistema especialista (RABELO *et al.*, 1997), por exemplo, mesmo com um processo de software bem definido, percebeu-se que a atividade de estudo do domínio, embora não presente neste processo, era constante em todas as fases do desenvolvimento.

[1] CORDIS – é uma palavra em latim que significa coração.

O trabalho realizado na UCCV/FBC, não só serviu de motivação para o estudo e definição de ADSOD, como, também, permitiu observar que o uso do conhecimento do domínio poderia atuar como diferencial importante durante o desenvolvimento de produtos de software.

Dessa forma, Cardiologia foi escolhida como o primeiro domínio onde seriam experimentadas as idéias apresentadas nesse trabalho. Outros fatores reforçaram essa escolha, como por exemplo:

- cardiologia é um domínio adequado para ADSOD, por ser possível definir uma Teoria do Domínio e por ser pertinente se considerar o desenvolvimento de diferentes tipos de software neste domínio;
- a existência desde 1994 de um convênio entre a área de Engenharia de Software da COPPE/UFRJ e a UCCV/FBC para projetos conjuntos envolvendo o desenvolvimento de software, com experiências no desenvolvimento de sistemas especialistas (RABELO *et al.*, 1997), aplicações na Web (GAMA *et al.*, 1996), (LIMA, 1999, LIMA *et al*, 1999), educação de pacientes (VALLE *et al.*, 1997), sistemas de informação (BLASCHECK, 1995, CARVALHO, 1997, GRISOLIA *et al.*, 1997). Estas experiências forneceram subsídios para este trabalho e o convênio nos facilitou a interação com especialistas (cardiologistas) para definição da Teoria do Domínio;
- nossa experiência anterior no desenvolvimento de software para este domínio (OLIVEIRA, 1995), (RABELO *et al.*, 1997), e,
- a experiência anterior da UCCV/FBC com a COPPE/UFRJ na definição, uso, avaliação e melhoria de processo de software (WERNECK, 1995, WERNECK *et al.*, 1995a, 1995b, WERNECK *et al.*, 1997).

A partir da infra-estrutura descrita no capítulo 5, foi iniciada, então, a construção de CORDIS, um ADSOD específico para Cardiologia. A seguir são descritos os aspectos de definição e construção do ADSOD CORDIS.

6.2 A Teoria do Domínio

A Teoria do Domínio de cardiologia foi definida segundo o processo apresentado no capítulo 4 a partir do conhecimento de um cardiologista da UCCV/FBC. Ao longo do trabalho, dois outros cardiologistas opinaram, discutiram algumas questões e avaliaram os modelos parciais realizados. Além disso, foi consultada a literatura

específica (AZEVEDO, 1984, SCHLANT e ALEXANDER, 1994, ROBINS, 1974, UMLS, 1998).

É importante salientar que não é objetivo desse trabalho explorar em todas os detalhes o domínio de cardiologia. A Teoria do Domínio, apresentada a seguir, é uma aproximação do domínio real e está sujeita a detalhamentos. Nosso objetivo foi utilizar essa teoria para mostrar sua aplicabilidade nos ADSOD.

Como citado no capítulo 3, um sistema de grande importância que utiliza uma ontologia biomédica é o UMLS (UMLS, 1998). Para realização desse trabalho, utilizamos essa ontologia como fonte de consulta e discussão. UMLS organiza todos os conceitos numa única árvore hierárquica. No entanto, os conceitos não possuem atributos (além de seu significado), não existem axiomas definidos e não é feita separação entre termos importantes (como sinais e sintomas, testes laboratoriais e outros resultados) definidos como um conceito único que os representa. Por ser genérico para a área biomédica, o UMLS perde as características da especificidade do contexto de cardiologia, como, por exemplo, para a organização da anatomia do coração. Dessa forma, muitos termos foram desconsiderados por se tratarem de aspectos sem interesse para o contexto de cardiologia (como por exemplo tipos de animais, tipos de estruturas químicas, entre vários outros) ou com o objetivo de limitar o escopo desse trabalho. Outros conceitos foram refinados (como os achados) para tornar possível se expressar uma melhor semântica. O UMLS pode ser útil ainda como servidor de conhecimento para definição de instâncias dos conceitos pois existem várias instâncias definidas. Para que isso seja possível, contudo, é necessário que se identifique corretamente que instâncias são específicas para cardiologia.

A seguir serão apresentados cada um dos passos realizados para a definição da Teoria do Domínio.

6.2.1 Identificação do Propósito

A definição de uma Teoria do Domínio para cardiologia tem como objetivo apoiar o desenvolvimento de software no ADSOD CORDIS. Dessa forma, é estabelecido um vocabulário comum que possa facilitar a comunicação entre os diferentes participantes no desenvolvimento de produtos de software utilizando o ambiente, ou seja, desenvolvedores, engenheiros de conhecimento, especialistas (cardiologistas) e usuários em geral.

Cardiologia é um ramo da medicina que lida com o estudo do coração. A medicina utiliza as ciências naturais como fonte de conhecimento, combinando-as em uma descrição de processos patofisiológicos ocorridos no paciente. Associada a essa representação profunda, existe uma definição superficial das doenças ou cenários médicos, motivada por um conhecimento parcial do comportamento dos processos patofisiológicos e da necessidade de uma representação compacta do conhecimento para ser rapidamente explorada na prática clínica. Contudo, à medida em que aumenta o conhecimento das ciências naturais, as possibilidades de ampliar essas explicações na medicina são certas, o que possibilita capturar, mais completamente, a essência (VAN HEIJST, 1995).

A Teoria do Domínio de cardiologia procura definir, hierarquizar e organizar esses conceitos no que se refere aos processos, cenários e práticas clínicas relacionadas a problemas cardíacos. Alguns conceitos, no entanto, são comuns na medicina em geral e, portanto, também aplicáveis à cardiologia.

6.2.2 Definição

Para definição da Teoria do Domínio vários ciclos de captura, conceituação e especificação foram realizados. Inicialmente, a partir de uma reunião de *brainstorming* com dois cardiologistas, procuramos registrar os conceitos aleatoriamente à medida em que iam surgindo. Com essa reunião foi identificado um conjunto de conceitos importantes na área de cardiologia que englobam informações de diferentes níveis de abstração. Desenvolver sistemas nesta área pode requerer qualquer uma dessas informações o que leva à necessidade da definição de diferentes conjuntos de conceitos que irão compor diferentes sub-teorias para o domínio.

Para identificação das sub-teorias foram definidas, inicialmente, questões de competência gerais com base no objetivo definido para a Teoria do Domínio, ou seja, o que seria importante e útil para o desenvolvimento de software e que, portanto, a Teoria deveria contemplar. Foram identificados dois grupos principais. O primeiro se refere a como está, realmente, estruturado o objeto principal da cardiologia, ou seja, o coração. O segundo grupo, se refere a um conjunto de conceitos sobre como descobrir, estudar e nomear eventos que ocorrem dentro do coração. A partir desses aspectos, identificaram-se as seguintes questões de competência gerais:

(a) Como está organizado o coração, seus componentes e estruturas internas?

(b) Como funciona o coração?

(c) O que é investigado pelo médico para estudar o que acontece no coração?

(d) Como se caracterizam os diferentes tipos de diagnóstico definidos pelo médico?

(e) Como se caracterizam os diferentes tipos de terapia empregadas pelo médico?

(f) Que tipos de patologia são estudadas na área de cardiologia?

A partir dessas questões gerais, foram definidas cinco sub-teorias a serem exploradas (Figura 6.1):

- **anatomia do coração**, para responder às questões (a) e (b), que possui os conceitos sobre a estrutura do coração e sua fisiologia;
- **patologia**, para responder à questão (f), que representam estados do coração mas cuja classificação é importante para o propósito da Teoria do Domínio no ADSOD;
- **avaliação clínica e exames complementares**, para responder à questão (c), que representam os conceitos utilizados pelos cardiologistas no processo de investigação médica;
- **diagnóstico**, para responder à questão (d), com as classificações dos tipos de diagnóstico, e,
- **terapia**, para responder à questão (e), que contém os conceitos sobre terapia empregadas em cardiologia.

Nos ciclos de definição, a seguir, foram exploradas cada uma dessas sub-teorias no que se refere à especificação, captura e conceituação dos diferentes conceitos envolvidos.

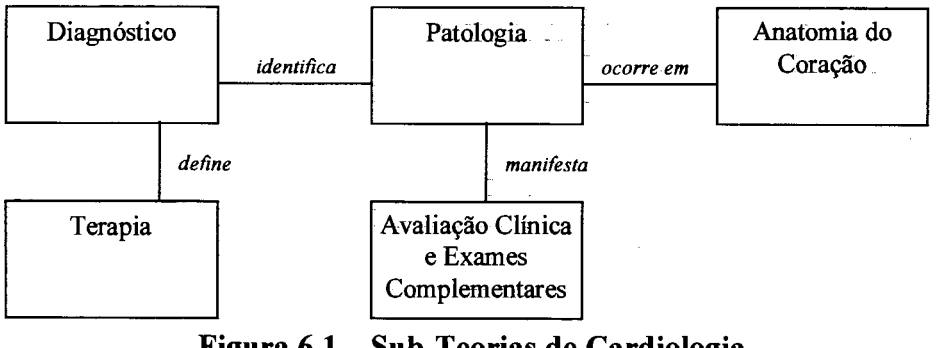

Figura 6.1 – Sub-Teorias de Cardiologia

a) Sub-Teoria Anatomia do Coração

- **Especificação**

O coração é o objeto de estudo da área de cardiologia. Todo trabalho do cardiologista está baseado na avaliação do estado do coração e na determinação de procedimentos para mantê-lo em funcionamento adequado. Esta avaliação é feita

baseada em evidências externas percebidas pelo cardiologista já que o coração é um orgão interno do corpo humano. No entanto, para que o cardiologista faça essa avaliação, ele tem que possuir um conhecimento profundo da estrutura do coração e de suas funcionalidades. A identificação de anomalias considera, portanto, o entendimento da funcionalidade normal de cada componente do coração. Assim sendo, os fatores externos considerados estão, direta ou indiretamente, relacionados com a anatomia do coração.

A ontologia sobre anatomia do coração deve, portanto, ser capaz de responder às sequintes questões:

- Como está organizado o coração, seus componentes e estruturas internas?
- Como funciona o coração?
- Que relações existem entre estes componentes?

• Conceituação

Analisando as questões de competência, anteriormente relacionadas, identificamos em linhas gerais um conjunto de componentes que formam um substrato anatômico. Este, por sua vez, pode ter um funcionamento normal ou anormal a partir do estado de seus componentes.

A Figura 6.2 mostra a organização dos conceitos definidos para esta teoria, representados em LINGO (FALBO, 1998). A opção de utilizar uma representação gráfica específica para ontologias e não uma representação existente como entidade-realcionamento ou linguagem de modelagem de objetos se deve ao fato de que não desejamos incorporar na descrição da ontologia a semântica natural dessas linguagens e, além disso, as relações expressas numa linguagem para descrição de ontologia devem possuir um conjunto de axiomas que as representem formalmente. LINGO possui relações capazes de capturar certos axiomas de forma implícita (FALBO, 1998).

No Anexo 2, é apresentada a descrição de todos os conceitos dessa sub-teoria.

Analisando a Teoria alguns axiomas foram capturados:

- A câmara cardíaca sempre é delimitada por uma parede
- O septo sempre separa duas câmaras cardíacas
- O sistema excito condutor intercomunica as câmaras cardíacas
- Todo substrato anatômico possui uma função que pode estar normal ou anormal

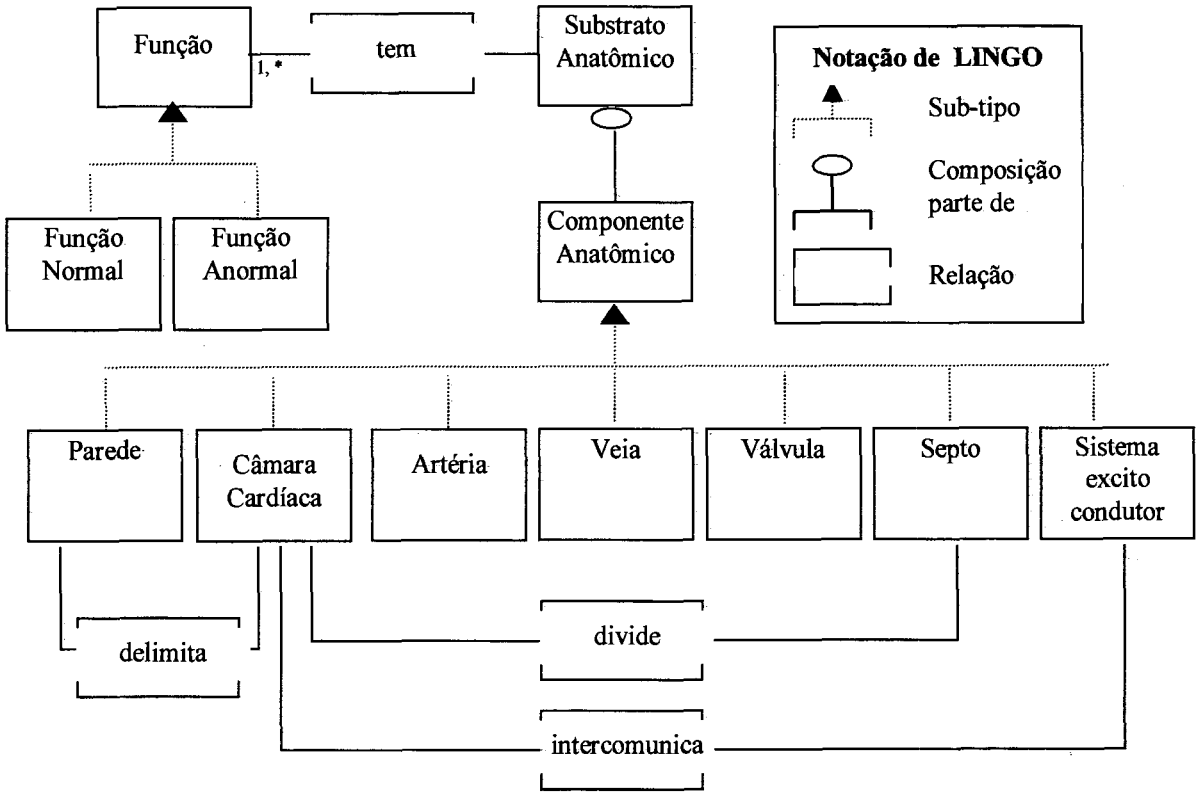

Figura 6.2 – Sub-Teoria Anatomia do Coração

b) Sub-Teoria Patologia

- **Especificação**

A patologia decorre de uma disfunção do coração devido a alguma desordem em alguma parte da estrutura cardíaca. Essa desordem, decorrente de algum fator específico, é manifestada como um conjunto de evidências apresentadas pelo paciente depois de algum tempo. A patologia se classifica dependendo do fator que a produziu possuindo características distintas. Dessa forma, uma ontologia para patologia deve responder às seguintes questões:

- Que tipo de fatores produzem uma patologia?
- Esses fatores podem vir associados a outros fatores?
- Quais os tipos de patologias que um determinado fator pode causar?

- **Conceituação**

Analisando as questões de competência, anteriormente relacionadas, foram realizadas várias entrevistas até chegarmos à sub-teoria mostrada Figura 6.3.

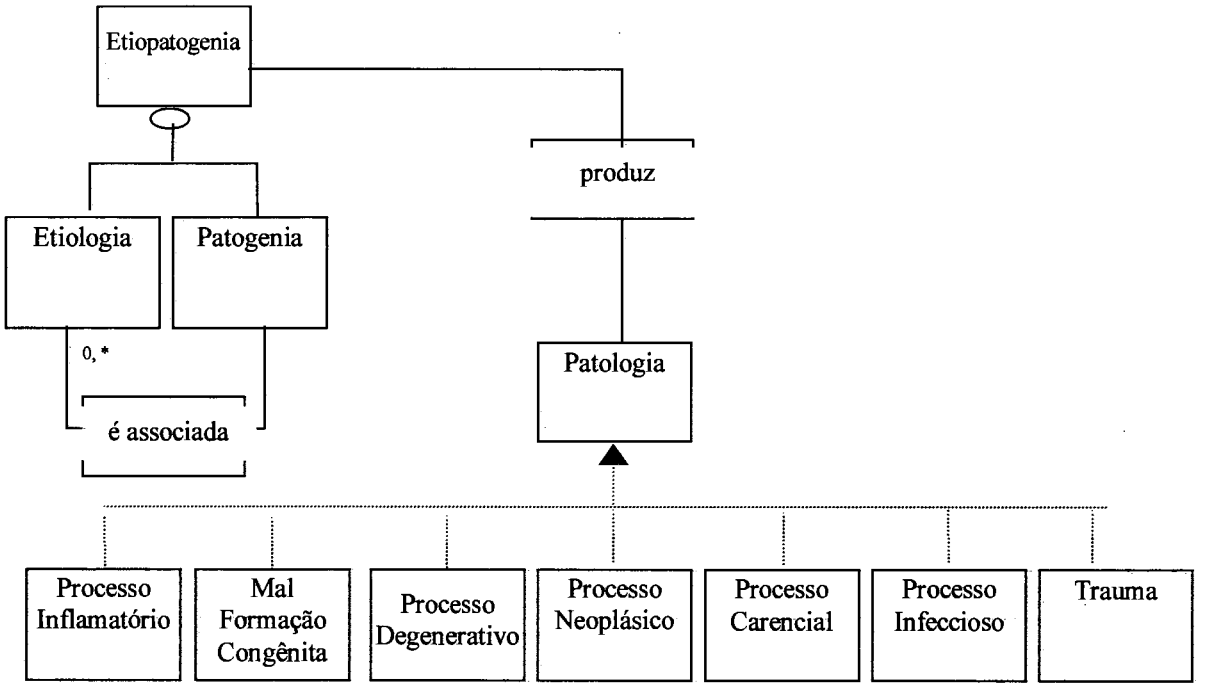

Figura 6.3 – Sub-Teoria Patologia

A documentação dos conceitos está apresentada no Anexo 2.

Como axiomas, tem-se:

- Toda patologia que é produzida por um fator etiológico não é uma mal formação congênita.
- Um fator patogênico pode não estar associado a nenhum fator etiológico.

c) Sub-Teoria Diagnóstico

- **Especificação**

O diagnóstico é a identificação de uma patologia. O diagnóstico de um paciente é definido pelo seu quadro clínico e exames complementares. Todo diagnóstico caracteriza uma síndrome e uma etiologia. A síndrome é a manifestação clínica da patologia e a etiologia são os aspectos que caracterizam a causa da doença. O conjunto de manifestações clínicas, que permitem o diagnóstico do paciente, são resultado da formação da patologia no indivíduo. A ontologia de diagnóstico deve responder apenas à seguinte questão:

- Que tipos de diagnósticos podem ser definidos?

- **Conceituação**

A taxonomia de diagnóstico está mostrada na Figura 6.4.

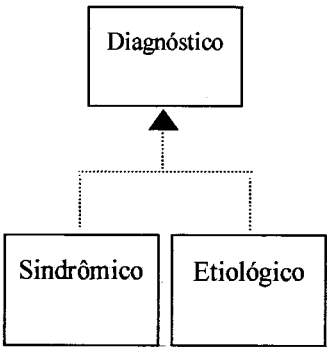

Figura 6.4 – Sub-Teoria Diagnóstico

Os axiomas referente a diagnóstico são relacionados com os conceitos de outras sub-teorias, sendo expressos quando se definir as fronteiras entre as sub-teorias.

d) Sub-Teoria Terapia

- **Especificação**

A terapia nada mais é do que o tratamento da doença diagnosticada podendo ser simplesmente clínico ou intervencionista. Cada tipo de terapia tem procedimentos próprios e informações específicas que são monitoradas ao longo do tratamento. Dessa forma, a ontologia de terapia deve responder às seguintes questões:

- A que tipos de terapia pode ser submetido o paciente?
- Como se caracteriza cada tipo de terapia?

- **Conceituação**

A taxonomia de terapia está mostrada na Figura 6.5. Nenhum axioma foi identificado nessa sub-teoria.

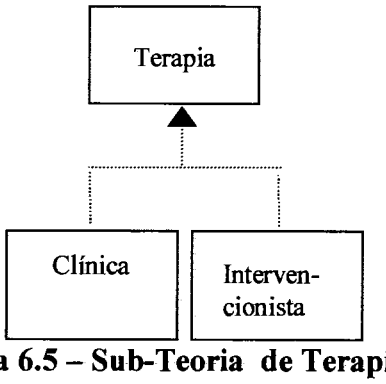

Figura 6.5 – Sub-Teoria de Terapia

e) Sub-Teoria Avaliação Clínica e Exames Complementares

- **Especificação**

Essa sub-teoria refere-se a um conjunto de características obtidas pelo cardiologista para identificar uma patologia. Através dessas características o cardiologista pode inferir ou diagnosticar o que o paciente possui para permitir a definição de uma terapia adequada. O processo de investigação das informações clínicas implica na avaliação de observações do cardiologista, relatos do paciente e exame físico. Com as informações do quadro clínico e os resultados dos exames complementares é definido o diagnóstico. A sub-teoria de avaliação clínica e exames complementares deve, portanto, responder às seguintes questões de competência:

- O que é investigado pelo médico no processo de anamnese?
- Que tipos de exames são utilizados para a avaliação do cardiologista?

- **Conceituação**

De certa forma, esse processo de anamnese é semelhante em qualquer área da medicina. O que particulariza essa sub-teoria como de cardiologia, é a definição de exames específicos requisitados pelos cardiologistas e de informações particulares investigadas em pacientes cardíacos. O modelo resultante está apresentado na Figura 6.6.

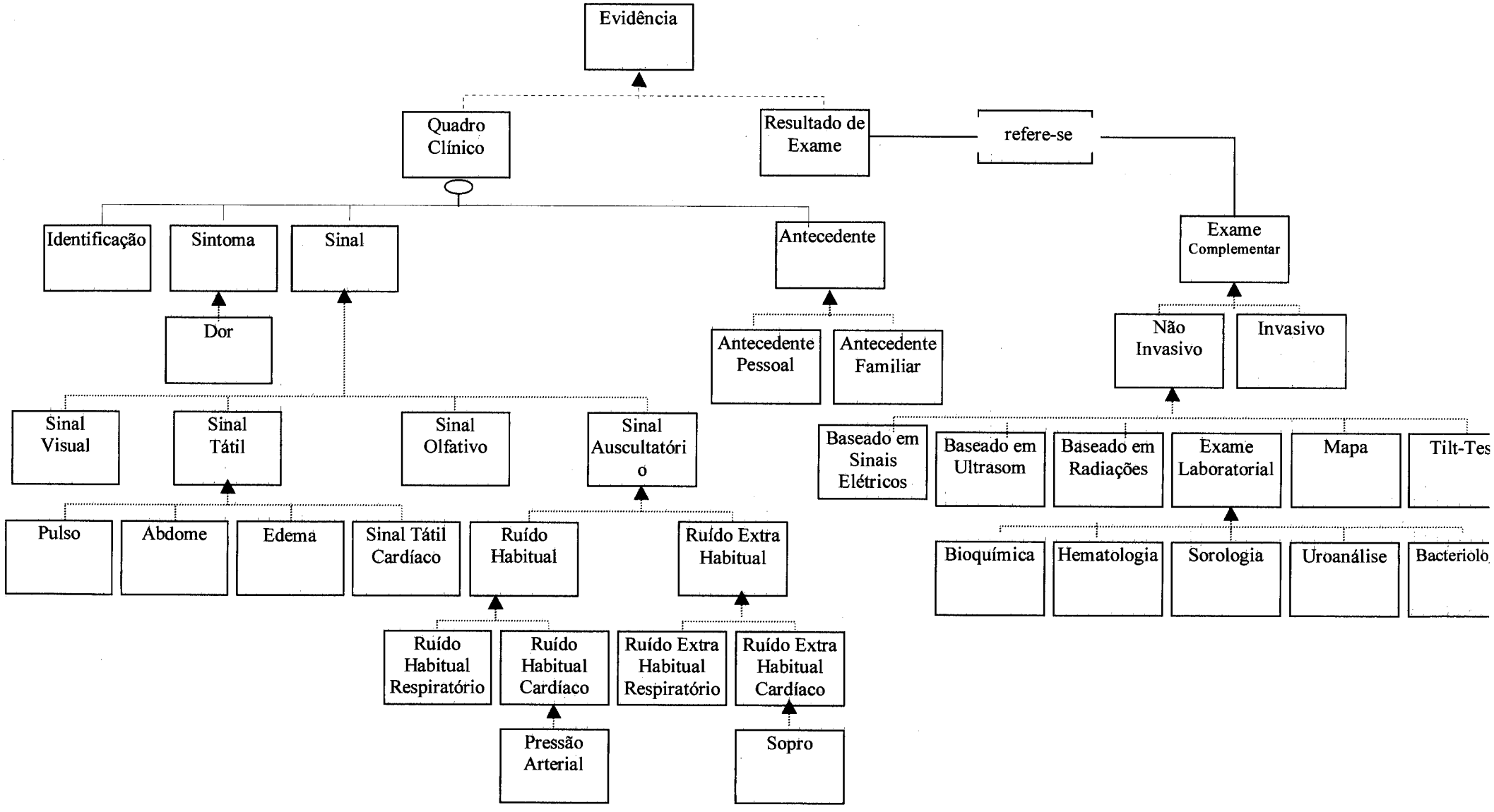

Figura 6.6 – Sub-teoria de Achados e Exames

Finalmente, as fronteiras entre as sub-teorias apresentadas na Figura 6.1 correspondem a relações entre conceitos das diferentes sub-teorias. Essas relações estão apresentadas na Figura 6.7.

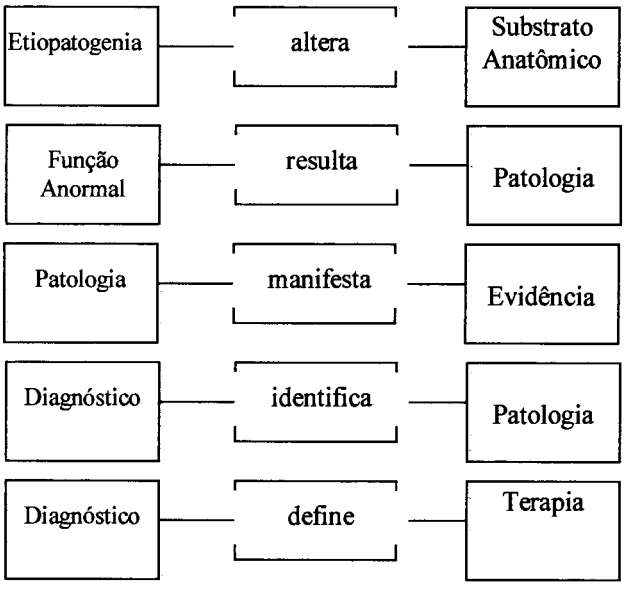

Figura 6.7 – Fronteiras entre as sub-teorias

Alguns dos axiomas para as relações entre as fronteiras são:

- sempre é possível a definição do diagnóstico sindrômico
- se existir um fator etiológico que produz uma patologia então é possível identificar um diagnóstico etiológico

O último ciclo de definição corresponde à identificação de tarefas realizadas no domínio e seu mapeamento com conceitos do domínio. As tarefas mais típicas realizadas no domínio de cardiologia são: diagnóstico, planejamento terapêutico, simulação e monitoração de pacientes. O mapeamento, definido numa Teoria do Domínio, entre conceitos e tarefas indica o que deve ser considerado quando se for desenvolver uma aplicação. Para a tarefa de diagnóstico é importante se considerar os conceitos das sub-teorias de avaliação clínica e exames complementares, diagnóstico e patologias. Para planejamento terapêutico são importantes as sub-teorias de terapia e patologia. No caso de monitoração são consideradas as sub-teorias de anatomia do coração e a de avaliação clínica e exames complementares. Finalmente, para simulação é importante a sub-teoria de anatomia do coração.

6.2.3 Formalização

Por serem descritos em LINGO, os axiomas epistemológicos de sub-tipo estão implicitamente definidos, como por exemplo:

(∀ a) artéria(a) → componente_anatômico(a).

Dessa forma, estes axiomas epistemológicos não serão descritos neste trabalho. Da mesma forma os axiomas referentes à composição todo_parte, não serão descritos.

Os diferentes axiomas definidos para as sub-teorias, apresentadas anteriormente, foram formalizados em lógica de primeira ordem, por ser essa a representação formal atualmente utilizada na Estação TABA (FALBO, 1998). Esses axiomas estão representados na Tabela 6.1.

Por ser utilizada a lógica de primeira ordem, os valores de domínio dos atributos apresentados no Anexo 2, também são explicitamente descritos conforme exemplo na Tabela 6.2.

Tabela 6.1 – Formalização de Axiomas

Descrição do Axioma	Formalização do Axioma
A câmara cardíaca sempre é delimitada por uma parede	(∀c) câmara_cardíaca (c) → (∃ p) parede(p) ∧ delimita (c, p).
O septo separa sempre duas câmaras cardíacas:	(∀c_1, c_2,) câmara_cardíaca (c_1) ∧ câmara_cardíaca (c_2) → (∃ s) septo(s) ∧ divide(s, c_1, c_2) ∧ diferente_de(c_1, c_2).
O sistema êxito condutor intercomunica as câmaras cardíacas	(∀c1, c2, ∃ s) câmara_cardíaca (c_1) ∧ câmara_cardíaca (c_2) ∧ diferente_de(c_1, c_2) ∧ sistema_condutor (s) → intercominca (s, c1, c2).
Todo substrato anatômico possui uma função que pode estar normal ou anormal	(∀s, ∃ f) substrato_anatômico (s) → tem_função (s, f) ∧ (função_normal (f) ∨ função_anormal (f))
Toda patologia que é produzida por um fator etiológico não é uma má formação congênita	(∀ p, ∃ e) (patologia(p) ∧ etiopatogenia(e) ∧ produz(e,p) → ¬ma_formacaocong(p))
Um fator patogênico pode não estar associado a nenhum fator etiológico	(∃ p, ∀ e) (patogenia(p) ∧ etiologia(e) → ¬associada(p, e))
Sempre é possível a definição do diagnóstico sindrômico	(∀ p) patologia(p) → (∃ d) diag_sindromico(d) ∧ identifica(d, p)
Se existir um fator etiológico que produz uma patologia então é possível identificar um diagnóstico etiológico	(∀e, ∃p ∃d) (etiologia(e) ∧ patologia(p) ∧ produz (e, p) → diag_etiologico(d) ∧ identifica(d, p))

Tabela 6.2 – Formalização de Axiomas de Domínio de Valores

Descrição do Axioma	Formalização do Axioma
Todo pulso é arrítmico, rítmico ou biferens	$(\forall p)$ (pulso(p) $\rightarrow$ (tipo(p, arrítmico) $\wedge \neg$ tipo(p, rítmico) $\wedge \neg$ tipo(p, biferens)) $\vee$ ($\neg$tipo(p, arrítmico) $\wedge$ tipo(p, rítmico) $\wedge \neg$ tipo(p, biferens)) $\vee$ ($\neg$tipo(p, arrítmico) $\wedge \neg$tipo(p, rítmico) $\wedge$ tipo(p, biferens))

6.2.4 Codificação

Conforme descrito no capítulo anterior, as ontologias do domínio são incorporadas à Estação TABA através da arquitetura de Servidores de Conhecimento (FALBO, 1998). Para isso, a Teoria do Domínio é editada através da ferramenta EDITED, fazendo com que todos os conceitos e axiomas descritos sejam definidos como módulos de conhecimento em Prolog. Para isso é necessário que o engenheiro do ambiente escreva os axiomas apresentados em lógica de 1ª ordem em cláusulas de Horn permitindo a condificação Prolog. Esse trabalho envolve análise dos axiomas e projeto de como traduzi-los da melhor forma para Prolog. Com os módulos de conhecimento construídos, a ontologia pode ser instanciada no ambiente gerado formando uma base de conhecimento sobre o domínio de cardiologia.

6.3 Definição do Processo de Software

O Processo Padrão da UCCV/FBC é um processo de software que procura atender a todas as fases do desenvolvimento de software em um número simplificado de atividades. Os projetos da UCCV/FBC envolvem pesquisa e a equipe de desenvolvimento, geralmente, é pequena e sem grande experiência no desenvolvimento de sistemas. Além disso, há alta rotatividade de pessoal, típica de projetos de universidades e centros de pesquisa financiados com bolsas de Agências de Fomento. Os trabalhos são realizados com êxito a partir de uma gerência bem determinada, da boa definição dos projetos e de um rigoroso controle da qualidade ao longo do desenvolvimento.

O Processo Padrão a ser utilizado para especializar os vários processos de desenvolvimento e manutenção (a serem definidos para os diferentes tipos de software desenvolvidos na UCCV/FBC, conforme definido no capítulo 5), contém as seguintes atividades:

- **Análise do Problema,** cujo objetivo é obter uma visão global do que é o projeto a ser desenvolvido e uma identificação geral das áreas tecnológicas envolvidas;

- **Estudo de Sistemas Similares,** cujo objetivo é pesquisar sistemas similares de acordo com as áreas relacionadas com o propósito de obter uma visão geral do estado da arte no domínio do problema;
- **Definição Inicial dos Requisitos,** cujo objetivo é definir os requisitos do sistema de forma macroscópica (estes requisitos serão, posteriormente, detalhados na fase de Análise de Requisitos);
- **Prototipação,** cujo objetivo é oferecer uma primeira versão do produto aos cardiologistas para fins de validação do objetivo do produto;
- **Planejamento do Projeto,** cujo objetivo é elaborar a primeira versão do Plano d0 Projeto (o Plano deve ser detalhado ao longo do projeto).
- **Analise de Viabilidade do Projeto,** cujo objetivo é analisar a viabilidade de construção do sistema a partir do Plano do Projeto descrito.
- **Análise de Requisitos,** cujo objetivo é analisar, modelar e avaliar os requisitos do sistema a ser desenvolvido no que se refere às funções a serem desempenhadas e suas características de desempenho, requisitos de entrada e saída, requisitos de interface, requisitos de qualidade e demais características do sistema;
- **Projeto,** cujo objetivo é elaborar um modelo físico do sistema a partir da modelagem conceitual para permitir sua codificação;
- **Implementação,** cujo objetivo é produzir unidades de software executáveis e que implementem os componentes da arquitetura definida na linguagem de programação escolhida para o projeto;
- **Teste do Sistema,** cujo objetivo é avaliar o sistema com casos reais e integrado a todos os procedimentos manuais que tenham sido especificados, assim como com outros dispositivos de hardware necessários, para garantir a sua utilização.
- **Avaliação do Sistema em Campo,** cujo objetivo é colocar o sistema em uso no ambiente real, sendo utilizado pelos usuários finais com acompanhamento da equipe de desenvolvimento e sem a retirada do sistema anterior (caso este exista);
- **Implantação,** cujo objetivo é o uso efetivo do sistema;
- **Análise de modificações** – identificar e analisar os problemas ocorridos durante a operação e uso do sistema para determinação do que deve ser feito imediatamente e o que deve ser feito em uma próxima versão;

- **Implementação e Teste das Modificações,** cujo objetivo é codificar a alteração proposta. No caso de ser decidido fazer as alterações numa próxima versão, esta atividade implicará na realização das atividades de análise, projeto e implementação anteriormente definidas, e,
- **Implantação do sistema modificado,** cujo objetivo é implantar o sistema modificado no ambiente de uso.

Na Tabela 6.3 são apresentadas as sub-atividades relativas a cada uma dessas atividades.

Tabela 6.3 – Atividades do Processo Padrão da UCCV/FBC

Análise do Problema
Sub-atividades: • entendimento do problema – identificação do problema e definição do tipo de sistema que deve ser construído • investigação do domínio – identificação dos conceitos definidos na Teoria do Domínio relacionados com o projeto e das possíveis tarefas envolvidas para melhor entendimento do projeto; • definição do objetivo do projeto – descrição geral do projeto e dos possíveis benefícios esperados para a organização; • identificação de áreas relacionadas – identificação de possíveis áreas de pesquisa ou comerciais que oferecem solução para o projeto definido. **Produto:** Definição do Projeto **Responsáveis:** coordenadores médico e de informática do Núcleo de Pesquisa e Desenvolvimento de Software Médico da UCCV/FBC.
Estudo de Sistemas Similares
Sub-atividades: • pesquisa bibliográfica – consulta à bibliografia relacionada ao projeto, tanto da área de informática como da área médica; • pesquisa de projetos em uso – pesquisa em outras instituições (nacionais e internacionais) que possuem sistemas similares (no caso de instituições nacionais, deve-se estabelecer contatos ou visitas para melhor conhecimento do projeto); • identificação e descrição dos sistemas similares – breve descrição dos sistemas pesquisados mais semelhantes ao projeto. **Produto**: Descrição de Sistemas Similares **Responsáveis:** analistas de sistemas

Tabela 6.3 (cont.) – Atividades do Processo Padrão da UCCV/FBC

Definição Inicial dos Requisitos

Sub-atividades:

- exame do sistema atual (caso exista) – aquisição e avaliação de documentos do sistema atual procurando identificar as deficiências para justificar a necessidade do novo sistema;
- investigação do domínio – estudo dos conceitos e tarefas envolvidos no projeto;
- identificação dos requisitos do sistema – identificação, de forma geral, das funções requeridas para o sistema;
- descrição da arquitetura do sistema – definição de uma arquitetura de alto-nível do sistema identificando elementos de hardware (dispositivos externos, banco de dados, etc), software e operações manuais;
- identificação da interface com outros sistemas – definição da interface com outros sistemas já existentes na organização, em desenvolvimento ou a desenvolver;
- validação dos requisitos do sistema - avaliação dos requisitos gerais do sistema pelo coordenador médico do projeto.

Produto: Descrição do Sistema Proposto

Responsáveis: coordenador médico, coordenador de informática, coordenador técnico do projeto, analistas de sistemas e cardiologistas.

Prototipação

Sub-atividades:

Dependendo do tipo de sistema e da abordagem de desenvolvimento, este protótipo pode ser descartável ou evolutivo. Sendo evolutivo, pode requerer uma análise de requisitos mais elaborada e que sejam realizadas as outras atividades do processo sendo o produto final de um ciclo que envolva análise de requisitos, projeto, codificação e testes.

Produto: Protótipo

Responsáveis: coordenador técnico do projeto, analistas e cardiologistas.

Planejamento do Projeto

Sub-atividades:

Esta atividade consiste na elaboração do Plano do Projeto que deve conter: *(i)* Plano do Processo de Desenvolvimento; *(ii)* Plano de Organização das equipes de desenvolvimento e de gerência que atuarão no projeto; *(iii)* Plano de Documentação, com os roteiros de todos os documentos a serem gerados; *(iv)* Plano de Acompanhamento para controle do projeto; *(v)* Plano de Controle da Qualidade, com a descrição dos marcos, pontos de controle e de todos os procedimentos e atributos de qualidade a serem adotados no projeto; *(vi)* Plano de Recursos e Produtos a serem utilizados ou gerados no desenvolvimento do projeto; *(vii)* Plano de Treinamento para desenvolvedores e usuários do produto; *(viii)* Plano de Implantação e Operação, com os procedimentos necessários para a implantação e operação do produto.

Produto: Plano do Projeto

Responsáveis: coordenador médico, coordenador de informática e coordenador técnico do projeto.

Analise da Viabilidade do Projeto

Sub-atividades:

Esta atividade consiste da avaliação da viabilidade de implementação considerando-se os seguintes aspectos: viabilidade econômica, viabilidade financeira, viabilidade tecnológica, viabilidade de mão de obra, viabilidade de cronograma, viabilidade social.

Produto: Documento de Aprovação do Plano de Projeto

Responsáveis: coordenador médico, coordenador de informática e coordenador técnico do projeto.

Tabela 6.3 (cont.) – Atividades do Processo Padrão da UCCV/FBC

Análise de Requisitos

Sub-atividades:

- investigação do domínio - estudo e identificação dos conceitos da Teoria do Domínio relacionada ao projeto;
- coleta e registro de casos – organização de alguns casos existentes para utilização no processo de elicitação;
- elicitação dos requisitos do sistema, através de entrevistas com cardiologistas, do uso de ferramentas específicas do domínio (construídas utilizando a Teoria do Domínio) e dos casos coletados referentes às tarefas que estão sendo modeladas;
- identificação dos requisitos de qualidade, a partir das características da ISO/IEC 9126 (NBR 13596, 1996) e das características de qualidade específicas para aplicações no domínio médico (Anexo 3);
- especificação de requisitos - descrição dos requisitos, modelagem conceitual do sistema e, quando necessário, dos requisitos de banco de dados que deve considerar a estrutura e organização dos conceitos na Teoria do Domínio;
- avaliação da especificação de requisitos.

Produto: Especificação de Requisitos

Responsáveis: coordenador técnico do projeto, analistas de sistemas e cardiologistas.

Projeto

Sub-atividades:

Esta atividade consiste, basicamente, da transformação do projeto lógico (modelagem conceitual definida na atividade de análise de requisitos) em um projeto físico, utilizando-se os procedimentos do método de desenvolvimento escolhido. Esta atividade compreende as seguintes sub-atividades:

- investigação do domínio - verificação da organização dos conceitos, restrições e definições da Teoria do Domínio que possam ser utilizados no projeto específico;
- especificação do projeto de interface de alto nível – definição, de forma geral, da interface externa e interna do sistema;
- especificação do projeto – descrição da estrutura dos principais componentes, e detalhamento desses componentes de forma que possa ser codificados, compilados e testados;
- avaliação do projeto.

Produto: Especificação de Projeto

Responsáveis: coordenador de informática, coordenador técnico do projeto, analistas de sistemas e cardiologistas.

Implementação

Sub-atividades:

- geração de código, que implica na geração e na documentação de programas;
- teste individual de unidades e programas;
- integração dos módulos formando o sistema;
- integração do sistema com outros sistemas (se pertinente) que devem estar integrados conforme descrito na atividade da fase de definição inicial de requisitos;
- teste de integração do sistema, que compreende o teste do sistema com todos os módulos e o teste com outros sistemas.

Produto: Sistema (ou versão do sistema) testado

Responsáveis: analistas e programadores.

Tabela 6.3 (cont.) – Atividades do Processo Padrão da UCCV/FBC

Teste do Sistema

Sub-atividades:

- coleta de casos reais, no ambiente em que o sistema será operado. Podem ser usados casos históricos da organização se tiverem previamente sido registrados de acordo com os requisitos do sistema a ser implantado. Caso contrário, deve-se estabelecer o procedimento de coleta desses casos no ambiente de uso do sistema mas sem inicialmente usar o sistema para seu objetivo final;
- teste do sistema e avaliação dos resultados dos testes com casos reais – a avaliação deve ser feita através de uma reunião com o coordenador do projeto.

Produto: Sistema (ou versão do sistema) pronto para avaliação em campo

Responsáveis: coordenador médico, coordenador de informática, coordenador técnico do projeto, analistas de sistemas e cardiologistas.

Avaliação do Sistema em Campo

Sub-atividades:

- definição da equipe médica de apoio (quando necessário), para acompanhar a utilização pelo usuário no decorrer da realização de sua atividade;
- demonstração do sistema para o usuário;
- treinamento do usuário para utilização do sistema;
- avaliação do sistema pelos usuários, identificando aspectos que devem ser modificados e sugerindo ajustes para adequação ao trabalho;
- correção de problemas percebidos durante o uso e a partir de sugestões do usuário.

Produto: Sistema (ou versão do sistema) pronto para uso

Responsáveis: coordenador médico, coordenador de informática, coordenador técnico do projeto, analistas de sistemas e cardiologistas.

Implantação

Sub-atividades:

- elaboração do manual do sistema e *help on-line* que irão auxiliar os usuários na utilização do sistema;
- avaliação final dos documentos a serem disponibilizados para o usuário;
- definição do suporte ao usuário, estabelecendo que tipo de suporte o usuário terá e quem são os responsáveis por este suporte;
- treinamento dos usuários;
- instalação do sistema, podendo requerer a desativação do sistema anterior (quando for o caso).

Produto: Sistema em uso

Responsáveis: coordenador de informática, coordenador técnico do projeto, analistas e usuários.

Análise de modificações

Sub-atividades:

- identificação das modificações – identificação de deficiências no sistema, modificações ou novos requisitos solicitados pelo usuário.
- análise do problema – análise dos novos requisitos (ou erros) identificados, determinação se a alteração deve ser feita em uma próxima versão do sistema ou corrigida de imediato e proposta da solução, se for o caso;
- avaliar solução - o coordenador técnico do projeto analisa a proposta de solução para aprovação ou adequação.

Produto: Documento com alterações

Responsáveis: coordenador de informática, coordenador técnico do projeto, analistas e usuários.

Tabela 6.3 (cont.) – Atividades do Processo Padrão da UCCV/FBC

Implementação e Teste de Modificações
Sub-atividades: • detalhamento da solução - definição do projeto detalhado da solução relacionando com os componentes que serão alterados, deixando claro os pontos a serem alterados; • realização da modificação – entendimento da solução, das partes do sistema que serão alteradas e realização da modificação pelos programadores; • teste do componente alterado - o programador testa o novo componente ou o componente alterado; • teste de integração – teste do sistema com o novo componente ou componente alterado. **Produto:** Sistema modificado **Responsáveis:** coordenador de informática, coordenador técnico do projeto e analistas.
Implantação do sistema modificado
Sub-atividades: • **implantação** paralela ou substituição do sistema modificado; • treinamento do usuário para as novas funcionalidades; • retirada do sistema anterior. **Produto:** Sistema modificado em uso. **Responsáveis:** coordenador de informática, coordenador da equipe e analistas.

Várias atividades e sub-atividades definidas neste Processo Padrão são bastante particulares de sistemas desenvolvidos na área médica e, no nosso caso, particulares da UCCV/FBC. É o caso, por exemplo, da atividade de Estudo de Sistemas Similares, normalmente realizada pelo fato dos sistemas desenvolvidos na UCCV/FBC serem, em geral, de caráter inovador e de pesquisa, sendo necessário um estudo do estado da arte para permitir a identificação do que pode realmente ser feito, pois apesar do aspecto de pesquisa, existe um forte compromisso de que os sistemas desenvolvidos sejam, realmente, utilizados na prática médica. Outra atividade é a necessidade do uso de casos reais tanto para a análise de requisitos quanto para os testes finais de aceitação do sistema, pelo fato dos cardiologistas estarem habituados a trabalhar com casos (ou seja, situações de pacientes) e de raciocinarem a partir de problemas apresentados nos casos. Pelo mesmo motivo, existe sempre a necessidade da construção de um protótipo inicial que confirme os objetivos definidos para o sistema a ser desenvolvido e a necessidade de uso do sistema antes deste ser implantado.

O Processo Padrão foi especializado para um tipo de software: sistemas baseados em conhecimento. A escolha desse tipo de software foi motivada por diferentes razões. A principal delas refere-se à experiência anterior de desenvolvimento de sistemas desse tipo na UCCV/FBC (RABELO *et al.*, 1997). Essa experiência mostrou que, apesar do processo de software ser bem definido, a necessidade de investigação do domínio esteve

presente em todo o desenvolvimento. No entanto, o processo de software definido não considerava explicitamente essa atividade sendo apenas direcionado para a aquisição do conhecimento específico daquela aplicação. Outra razão é o interesse da organização (UCCV/FBC) em desenvolver outro sistema especialista.

A especialização do Processo Padrão para sistemas baseados em conhecimento (SBC) foi feita considerando-se a experiência anterior de desenvolvimento e definição de processo realizada pela COPPE e UCCV/FBC (WERNECK, 1995, WERNECK *et al.*, 1995a, WERNECK, 1995b, WERNECK *et al.*, 1997, OLIVEIRA, 1996a). Este processo foi revisto e foram incorporadas as atividades específicas definidas para ADSOD. Na Tabela 6.4 são apresentadas as atividades e sub-atividades do processo especializado para SBC.

Tabela 6.4 – Atividades do Processo Especializado para SBC

Análise do Problema
Sub-atividades: Mesmas atividades definidas no processo padrão. Observamos que a sub-atividade identificação de áreas relacionadas refere-se a sistemas na área de Inteligência Artificial e redes neurais. **Ferramentas de apoio:** ferramenta para investigação do domínio e editor de documentos **Produto:** Definição do Projeto **Responsáveis:** coordenador médico e coordenador de informática do Núcleo de Pesquisa e Desenvolvimento de Software Médico.
Estudo de Sistemas Similares
Sub-atividades: Mesmas atividades definidas no processo padrão **Ferramentas de apoio:** editor de documentos, *web*, *medline*[2] **Produto**: Descrição de Sistemas Similares **Responsáveis:** analistas de sistemas
Definição Inicial dos Requisitos
Sub-atividades: Mesmas atividades definidas no processo padrão. Acrescentando-se a seguinte sub-atividade: • Aquisição do conhecimento geral – refere-se a uma elicitação geral do conhecimento que será utilizado no sistema a ser desenvolvido, identificando possíveis áreas de conhecimento que estarão envolvidas. **Ferramentas de apoio:** editor de documentos **Produto**: Descrição do Sistema Proposto **Responsáveis:** coordenador médico, coordenador de informática, coordenador técnico do projeto, analistas de sistemas e cardiologistas.

[2] Banco de dados com referências bibliográficas e resumos sobre publicações em medicina

Tabela 6.4 (cont.) – Atividades do Processo Especializado para SBC

Prototipação
Sub-atividades: Mesmas atividade definida no processo padrão
Ferramentas de apoio: Aion-DS[3]
Produto: Protótipo
Responsáveis: coordenador técnico do projeto, analistas e cardiologistas.
Planejamento do Projeto
Sub-atividades:
Mesmas atividades definidas no processo padrão
As avaliações da qualidade são feitas através de reuniões utilizando as técnicas Walkthrough e Inspeção (PRESSMAN, 1997) conforme o coordenador técnico julgue mais adequado.
Ferramentas de apoio: editor de documentos, *web*, *medline*
Produto: Plano do Projeto
Responsáveis: coordenador médico, coordenador de informática e coordenador técnico do projeto.
Analise de Viabilidade do Projeto
Sub-atividades:
Mesmas atividades definidas no processo padrão
Produto: Documento de Aprovação do Plano do Projeto
Responsáveis: coordenador médico, coordenador de informática e coordenador técnico do projeto.
Análise de Requisitos
Sub-atividades: • investigação do domínio - estudo e identificação dos conceitos da Teoria do Domínio relacionada ao projeto assim como dos axiomas entre estes conceitos; • coleta e registro de casos – organização de alguns casos existentes para utilização no processo de elicitação; • elicitação dos requisitos de conhecimento do sistema; • identificação dos requisitos de qualidade; • especificação de requisitos - descrição dos requisitos, modelagem conceitual do sistema de acordo com o modelo de especialidade definido em (HICKMAM *et al.*, 1992) e dos diagramas definidos para a modelagem lógica de acordo com o método KADS-Estendido[4] (WERNECK, 1995, WERNECK *et al.*, 1995b); • avaliação da especificação de requisitos.
Ferramentas de apoio: editor de documentos, ferramenta case para apoiar a modelagem com o método KADS e KADS-estendido, KED[5] e Qual-Cordis[6]
Produto: Especificação de Requisitos
Responsáveis: coordenador técnico do projeto, analistas de sistemas e cardiologistas.

[3] Aion-DS é uma *shell* para desenvolvimento de sistemas especialista

[4] KADS-estendido é uma extensão do método KADS (HICKMAN *et al.*, 1992) proposto por (WERNECK, 1995, WERNECK *et al,* 1995b) que define modelos complementares para análise e um modelo específico para a atividade de projeto.

[5] KED é um editor de conhecimento para Cardiologia que será descrito na seção 6.4

[6] Qual-Cordis é uma ferramenta para levantamento de requisitos de qualidade de software que será descrita na seção 6.4.

Tabela 6.4 (cont.) – Atividades do Processo Especializado para SBC

Projeto

Sub-atividades:

- definição do paradigma de representação do conhecimento – escolha do paradigma (frames, regras, etc) mais adequado ao projeto;
- investigação do domínio - verificação da organização dos conceitos, restrições e definições da Teoria do Domínio que possam ser utilizadas no projeto específico
- especificação do projeto de interface de alto nível – definição, de forma geral, da interface externa e interna do sistema. De acordo com o método KADS-Estendido isso implica na construção do Modelo de Implementação do Usuário que consiste da definição da interação do sistema com o usuário e na definição de como e quando serão disponibilizadas informações de explicação do raciocínio utilizado na inferência;
- especificação do projeto – descrição da estrutura dos principais componentes, e detalhamento desses componentes de forma que possa ser codificados, compilados e testados. De acordo o o método KADS-estendido esta atividade é realizada através das seguintes sub-atividades:
  - Construção do Diagrama Estrutural da Base de conhecimento – consiste da estrutura geral da base de conhecimento relacionando as representações que serão utilizadas;
  - Especificação da base de conhecimento – consiste da organização do conteúdo da base de conhecimento;
  - Especificação da memória de trabalho – implica na definição do conteúdo de cada caso executado no sistema que será armazenado em arquivos ou banco de dados;
  - Especificação de módulos – deve ser realizada quando for necessária alguma função especial, por exemplo, tratamento de incerteza e outros cálculos específicos.
- avaliação do projeto.

Ferramentas de apoio: editor de documentos, ferramenta CASE para apoiar a modelagem com o método KADS-estendido

Produto: Especificação de Projeto

Responsáveis: coordenador de informática, coordenador técnico do projeto, analistas de sistemas e cardiologistas.

Implementação

Sub-atividades: Mesmas atividades definidas no processo padrão

Ferramentas de apoio: Aion-DS e editor de documentos

Produto: Sistema (ou versão do sistema) testado

Responsáveis: analistas e programadores.

Teste do Sistema

Sub-atividades:

- coleta de casos reais, no ambiente em que o sistema será operado. Podem ser usados casos históricos da organização se tiverem previamente sido registrados de acordo com os requisitos do sistema a ser implantado. Caso contrário, estabelecer o procedimento de coleta desses casos no ambiente de uso do sistema mas sem inicialmente usar o sistema para seu objetivo final;
- teste do sistema e avaliação dos resultados dos testes com casos reais – a avaliação deve ser feita através de uma reunião com o coordenador do projeto.

Produto: Sistema (ou versão do sistema) pronto para avaliação em campo

Ferramentas de apoio: ferramenta para análise estatística de resultados

Responsáveis: coordenador médico, coordenador de informática, coordenador técnico do projeto, analistas de sistemas e cardiologistas.

Tabela 6.4 (cont.) – Atividades do Processo Especializado para SBC

Avaliação do Sistema em Campo
Sub-atividades: • definição da equipe médica de apoio (quando necessário), para acompanhar a utilização pelo usuário no decorrer da realização de sua atividade; • demonstração do sistema para o usuário; • treinamento do usuário para utilização do sistema; • avaliação do sistema pelos usuários, identificando aspectos que devem ser modificados e sugerindo ajustes para adequação ao trabalho; • correção de problemas percebidos durante o uso e a partir de sugestões do usuário. **Ferramentas de apoio:** Aion-DS **Produto:** Sistema (ou versão do sistema) pronto para uso **Responsáveis:** coordenador médico, coordenador de informática, coordenador técnico do projeto, analistas de sistemas e cardiologistas.
Implantação
Sub-atividades: Mesmas atividades definidas no processo padrão **Ferramentas de apoio:** Aion-DS **Produto:** Aion-DS **Responsáveis:** coordenador de informática, coordenador técnico do projeto, analistas e usuários.
Análise de modificações
Sub-atividades: Mesmas atividades definidas no processo padrão **Ferramentas de apoio:** editor de documentos , Aion-DS **Produto:** Documento com alterações **Responsáveis:** coordenador de informática, coordenador técnico do projeto, analistas e usuários.
Implementação e Teste de Modificações
Sub-atividades: Mesmas atividades definidas no processo padrão **Ferramentas de apoio:** Aion-DS, ferramenta para análise estatística de resultados **Produto:** Sistema modificado **Responsáveis:** coordenador de informática, coordenador técnico do projeto e analistas.
Implantação do sistema modificado
Sub-atividades: Mesmas atividades definidas no processo padrão **Ferramentas de apoio:** Aion-DS **Produto:** Sistema modificado em uso. **Responsáveis:** coordenador de informática, coordenador da equipe e analistas.

6.4 CORDIS-SBC: um exemplo de ADSOD instanciado

CORDIS-SBC é um ambiente orientado ao domínio de cardiologia instanciado pela Estação TABA, utilizando a Teoria do Domínio e o processo especializado para SBC apresentados nas seções anteriores.

A construção do ambiente CORDIS-SBC constou de quatro etapas: *(i)* edição da Teoria do Domínio de Cardiologia através do EDITED; *(ii)* customização da ferramenta Q-*fuzzy*, disponível na Estação TABA, para o domínio em questão, *(iii)* construção de uma ferramenta específica do domínio que utiliza a Teoria do Domínio definida, e, *(iv)* edição do processo específico para SBC através da ferramenta EDIT-PRO.

É importante destacar que as duas primeiras etapas só são necessárias por ser esse o primeiro ADSOD definido e instanciado para Cardiologia. Com a Teoria do Domínio já inserida na Estação TABA e a ferramenta para avaliação customizada, novos ambientes da família de ADSOD CORDIS podem ser instanciados para projetos de diferentes tipos de software. A seguir descrevemos cada uma dessas etapas.

- **Edição de Teoria do Domínio**

Esta etapa consistiu basicamente da edição da Teoria do Domínio no EDITED. Dessa forma, foram incluídas as informações sobre a Teoria do Domínio (Figura 6.8), cada uma das sub-teorias (Figura 6.9), seus conceitos (Figura 6.10), relações (Figura 6.11), axiomas (Figura 6.12), mapeamento com tarefas (Figura 6.13) e gerado os módulos de conhecimento. Com a edição da Teoria do Domínio o conhecimento sobre as ontologias do domínio está integrado na Estação e pode, então, ser utilizado no momento da instanciação.

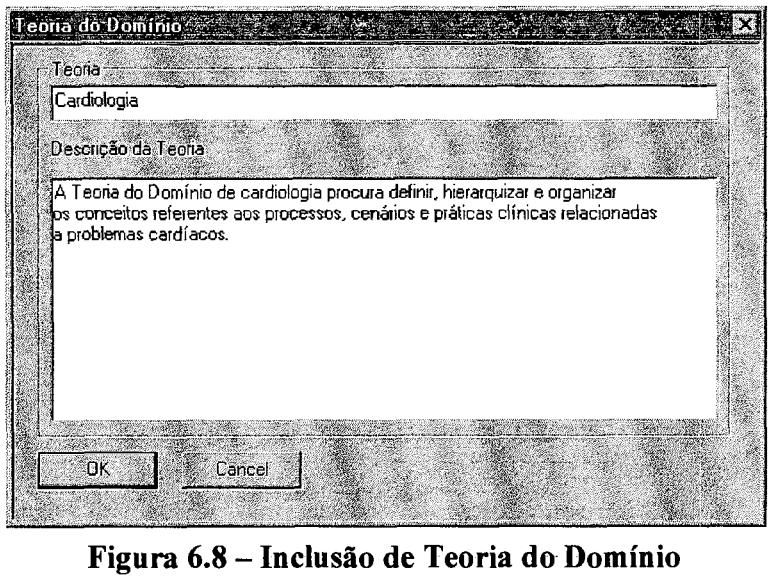

Figura 6.8 – Inclusão de Teoria do Domínio

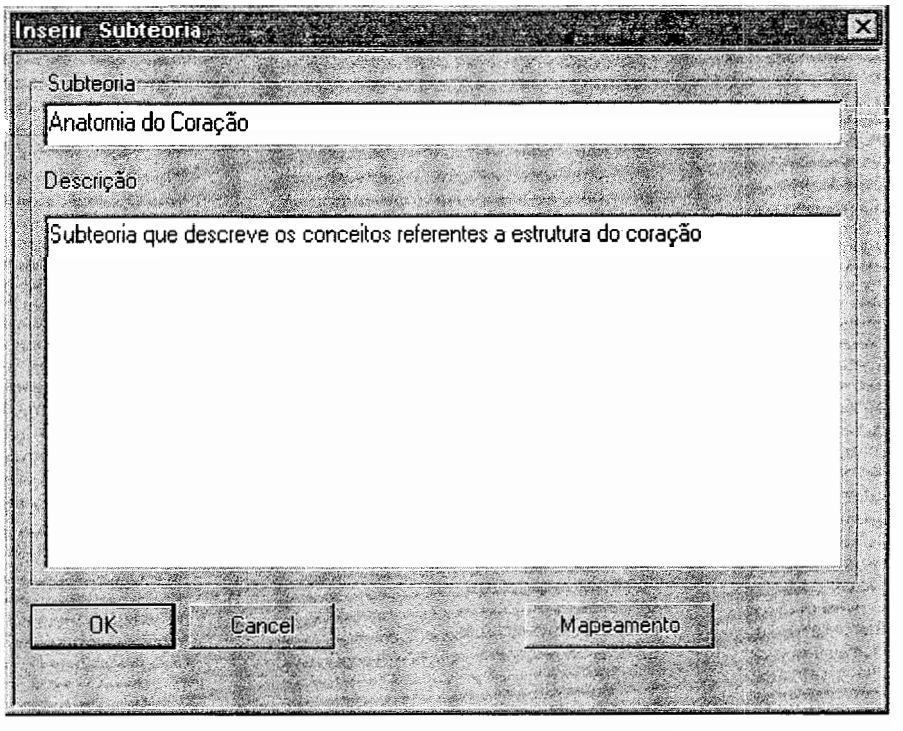

Figura 6.9 – Inclusão de Sub-teoria

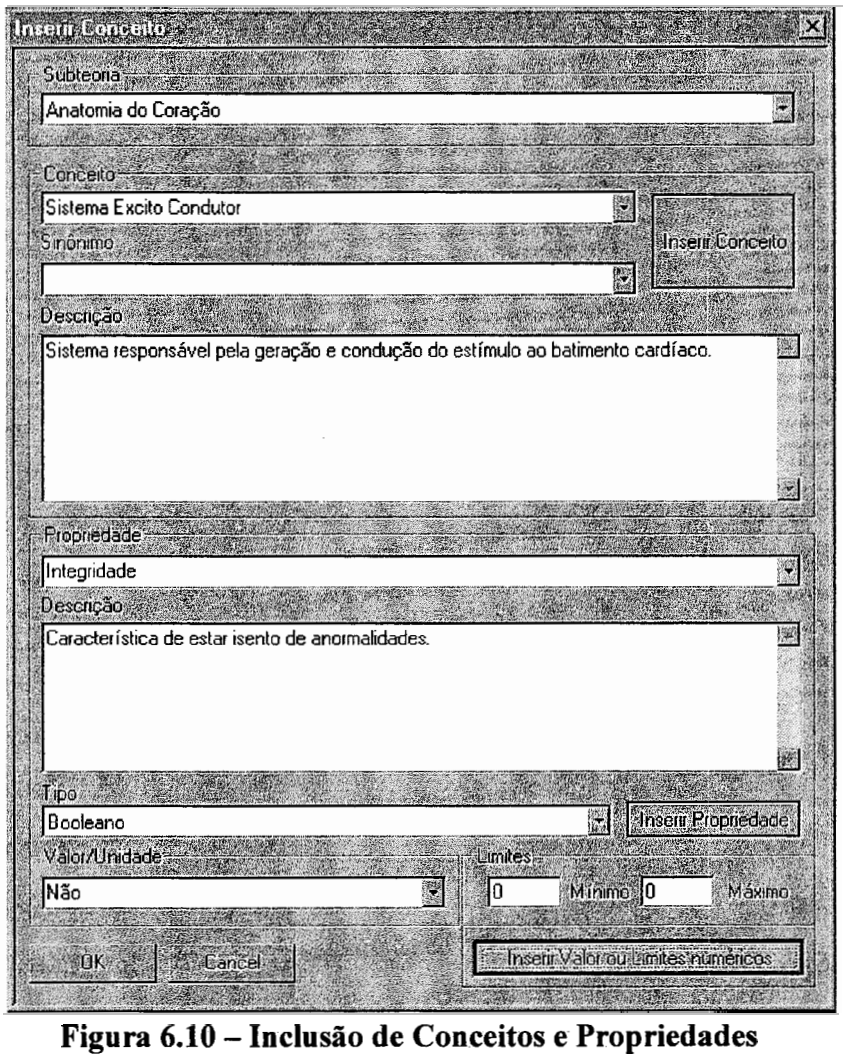

Figura 6.10 – Inclusão de Conceitos e Propriedades

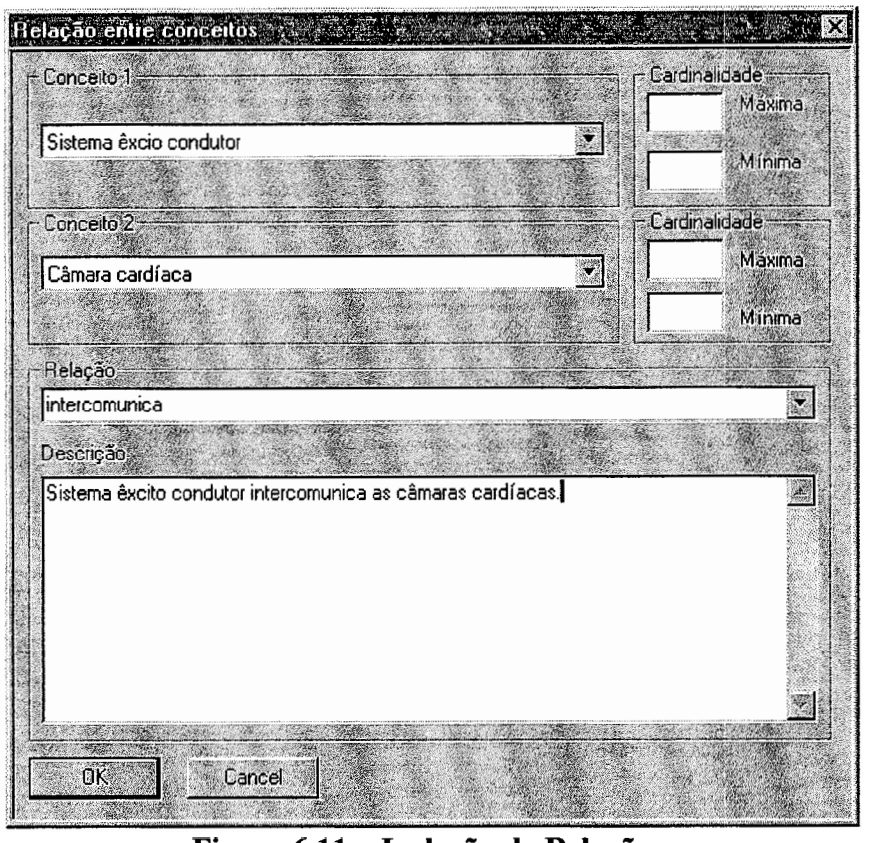

Figura 6.11 – Inclusão de Relações

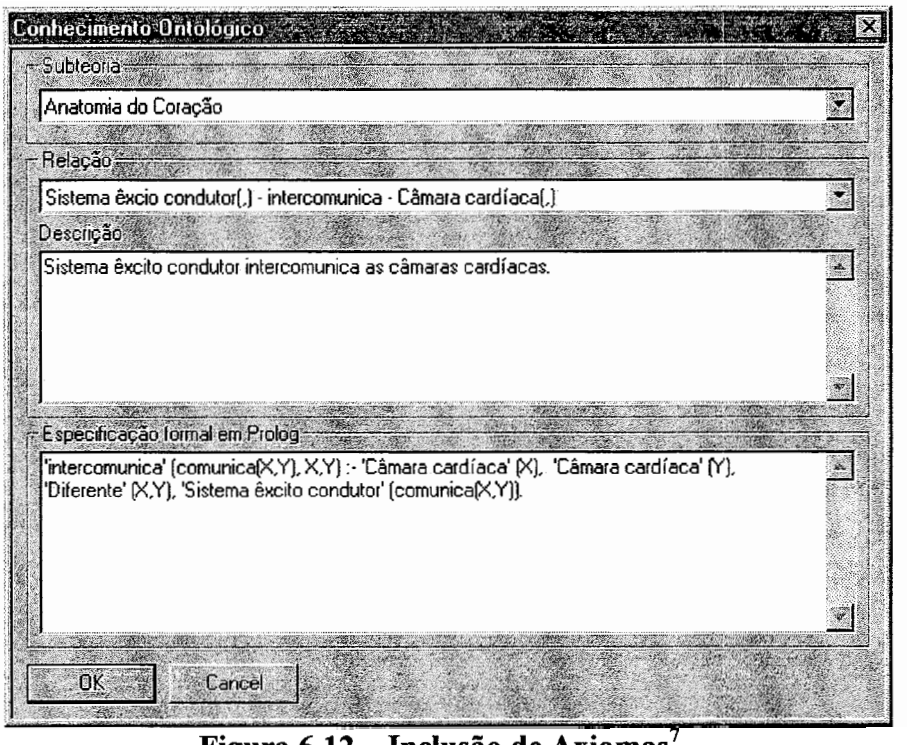

Figura 6.12 – Inclusão de Axiomas[7]

[7] A função comunica(X,Y) é uma função resultante de eskolemização realizada para o quantificador existencial (neste caso, ∃ s, Sistema êxcito condutor (s)).
Os predicados foram escritos entre apóstrofo (') para permitir a utilização dos próprios nomes dos conceitos e relações na formalização (com espaços, acentos, etc.).

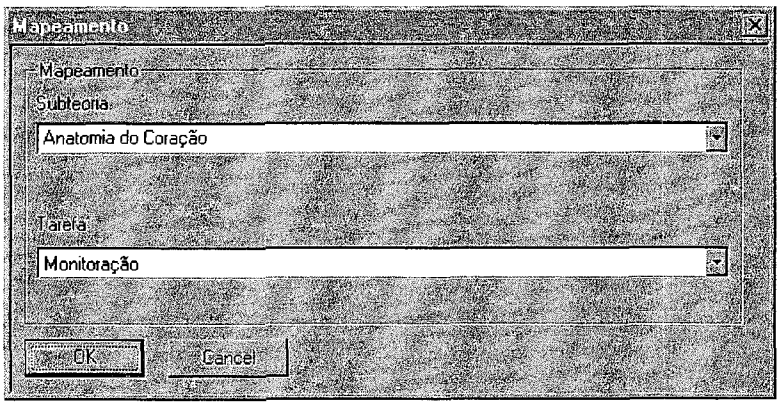

Figura 6.13 – Inclusão de Mapeamento com Tarefas

- **Customização da ferramenta Q-*fuzzy***

Conforme definido no capítulo anterior a ferramenta Q-*fuzzy* deve ser customizada para considerar atributos de qualidade referentes a diferentes tipos de software do domínio (nesse caso cardiologia, ou, de forma geral, o domínio médico) e de tecnologias de desenvolvimento. Foi definida, então, Qual-Cordis (OLIVEIRA *et. al.*, 1999c).

Para realizar essa customização organizamos e compatibilizamos os resultados de vários trabalhos anteriores, realizados na COPPE e na UCCV/FBC, que identificaram atributos de qualidade relativos a: sistemas de informação hospitalar (CARVALHO, 1997); sistema de acesso público para educação de pacientes (VALLE, 1997), prontuário médico eletrônico (CARVALHO, 1997, GRISOLIA, 1999), telemedicina (LIMA, 1999), sistemas especialistas (OLIVEIRA, 1995) e hipermídia (CAMPOS, 1994a). Foram, também, considerados atributos de qualidade de software educacional (CAMPOS, 1994b) por ser comum o desenvolvimento de tutores na área médica.

O trabalho de organização desses atributos envolveu a análise de todas as características de qualidade, a generalização de alguns conceitos que fossem aplicáveis a todos os tipos de sistemas, a definição de novos nomes para algumas características que se referiam às características já definidas e a re-classificação na organização hierárquica dos atributos proposta por ROCHA (1993). As características de qualidade consideradas em Qual-CORDIS estão no Anexo 3. Cada um dos atributos foi inserido na ferramenta como mostra a Figura 6.14.

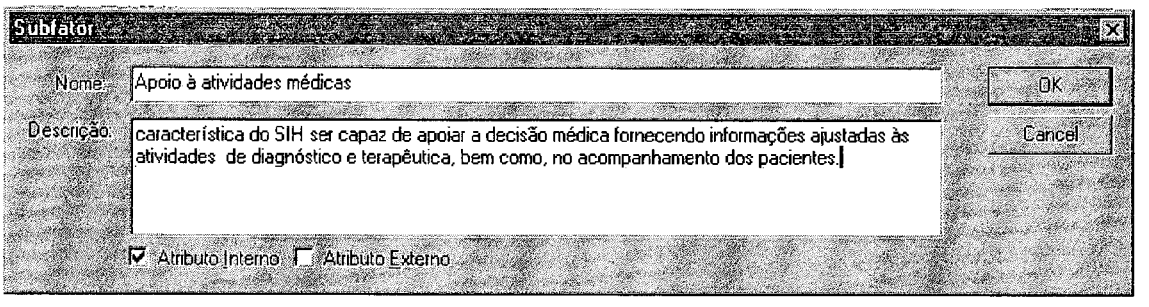

Figura 6.14 – Inclusão de atributos

O segundo passo para a customização foi a definição dos grupos de avaliadores e a elaboração de questionários para identificação do perfil do especialista (QIPE). No contexto de Qual-CORDIS foram considerados quatro grupo de especialistas: cardiologistas, enfermeiras, pessoal administrativo e pessoal de informática. A descrição de cada grupo foi inserida na ferramenta (Figura 6.15), assim como os QIPE correspondentes a cada grupo. A Tabela 6.5 mostra o QIPE específico para o grupo de cardiologistas e sua inclusão na ferramenta (Figura 6.16). Os demais QIPE estão apresentados no Anexo 4.

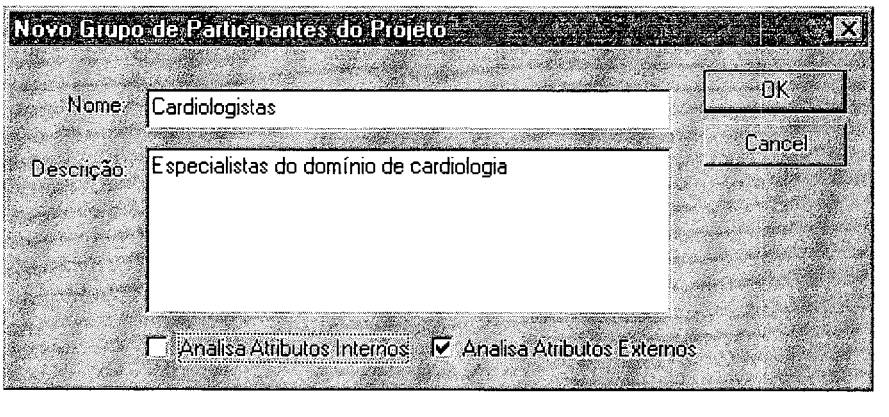

Figura 6.15 - Inclusão de novo grupo

Assim customizada, a ferramenta pode ser incluída em qualquer ambiente instanciado para o domínio de cardiologia, sendo utilizada na atividade de levantamento de requisitos. Novos atributos, grupos e QIPE ainda podem ser inseridos quando a ferramenta for utilizada no ambiente instanciado. Ao ser utilizado em um projeto específico, a ferramenta disponibiliza o conjunto de atributos de qualidade para diferentes tecnologias e para os diferentes tipos de software no domínio (Figura 6.17). Após a seleção do conjunto de atributos pertinente ao produto a ser avaliado, é disponibilizado o conjunto que deve ser submetido aos especialistas para ponderação. A este conjunto, podem, ainda, ser incluídos novos atributos e alterados ou excluídos atributos já definidos (Figura 6.18).

Tabela 6.5 – QIPE para médicos/cardiologistas

Questão	Itens
1) Marque sua experiência ou as atividades (cargos) que já exerceu, ou exerce, na área de médica.	❑ Chefe de unidade em que há desenvolvimento de software ❑ Chefe de setor em que há desenvolvimento de software ❑ Professor (universitário) na área de cardiologia ❑ Participante da equipe de desenvolvimento de software para cardiologia ❑ Usuário de software médico (ex.: sistema de informação, prontuário médico eletrônico, etc.) ❑ Outros: _____ (ex.: medline, internet, etc.)
2) á participou do desenvolvimento de quantos sistemas de computação?	❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7
3) Já usou quantos sistemas médicos (ex: sistemas de informação, prontuário eletrônico, etc.)?	❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7
4) Que tipos de atividades já realizou no desenvolvimento de sistemas?	❑ Definição do sistemas ❑ Entrevistas de elicitação ❑ Elaboração de modelos (estrutura de dados, etc.) ❑ Elaboração de casos de teste ❑ Avaliação do sistema (análise de resultados e testes)
5) Como você classificaria seu entendimento sobre o produto de software em questão?	❑ Excelente ❑ Bom ❑ Médio ❑ Baixo ❑ Nenhum
6) Marque a opção que melhor classifica seu grau de escolaridade	❑ Cardiologista com doutorado ou livre docência ❑ Cardiologista com mestrado ❑ Cardiologista ❑ Médico não cardiologista ❑ Residente
7) Quantos anos de experiência como médico?	❑ Nenhum ❑ Entre 1 e 5 ❑ Entre 5 e 10 ❑ Mais de 10

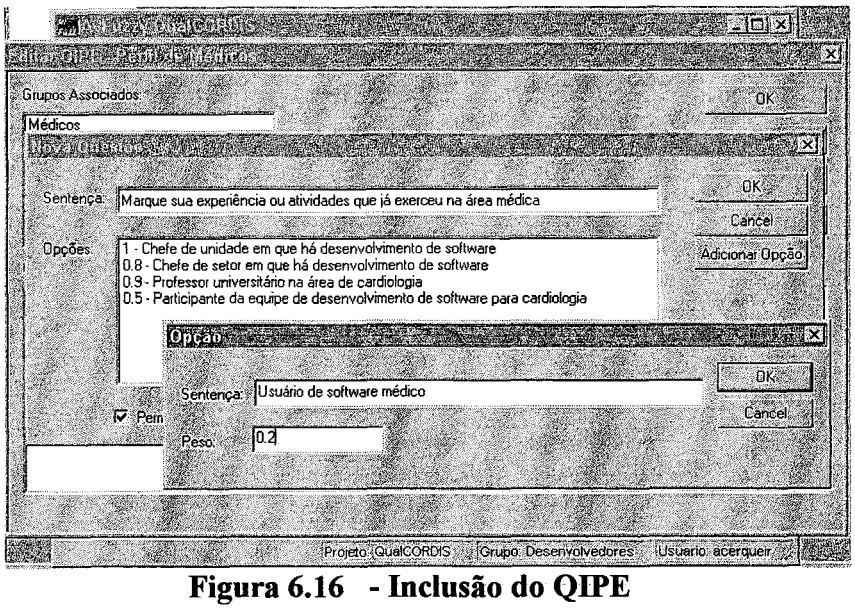

Figura 6.16 - Inclusão do QIPE

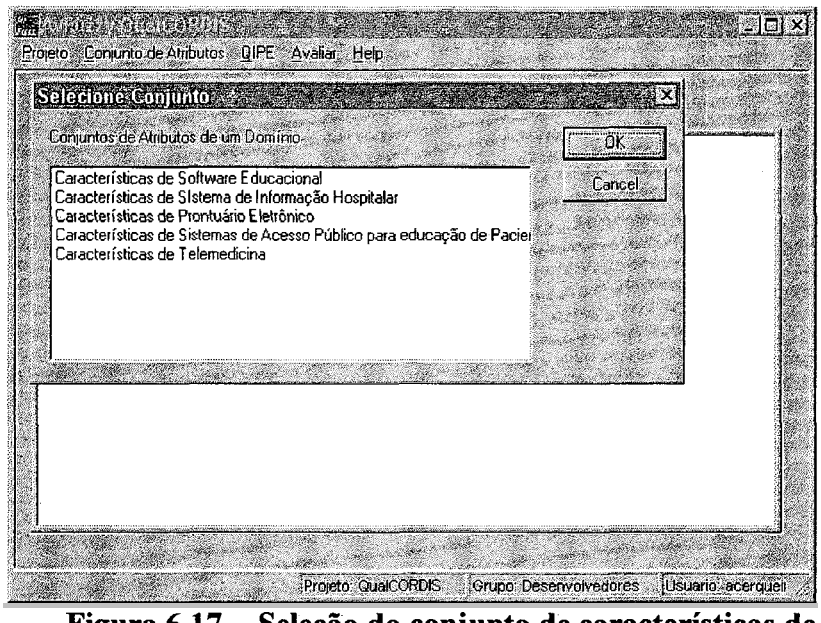

Figura 6.17 - Seleção do conjunto de características de qualidade para tipos de software de um domínio

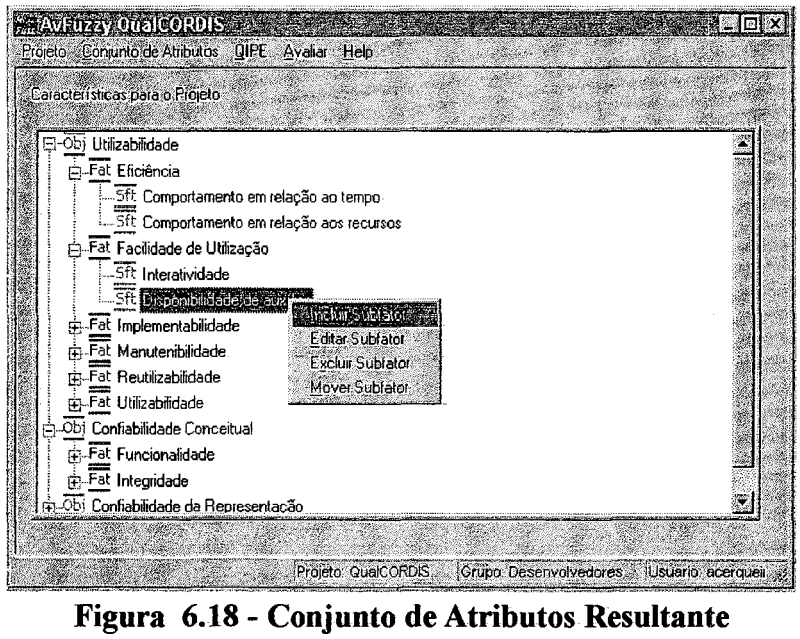

Figura 6.18 - Conjunto de Atributos Resultante

Para o levantamento de requisitos, o primeiro passo é responder os QIPE (Figura 6.19). Com base nas respostas, a ferramenta determina o peso de cada especialista para a avaliação. Após esta etapa, cada especialista deve avaliar cada atributo de qualidade (Figura 6.20) quanto a seu grau de importância no produto. Após todos os especialistas definirem o seu grau de importância para cada atributo, a ferramenta, a partir de uma solicitação do usuário administrador, determina os requisitos de qualidade do produto sobre consideração (Figura 6.21) e seus graus de importância considerando a função fuzzy específica. O engenheiro de software determina, então, a partir de que peso os

atributos de qualidade serão considerados requisitos de qualidade para o produto a ser avaliado.

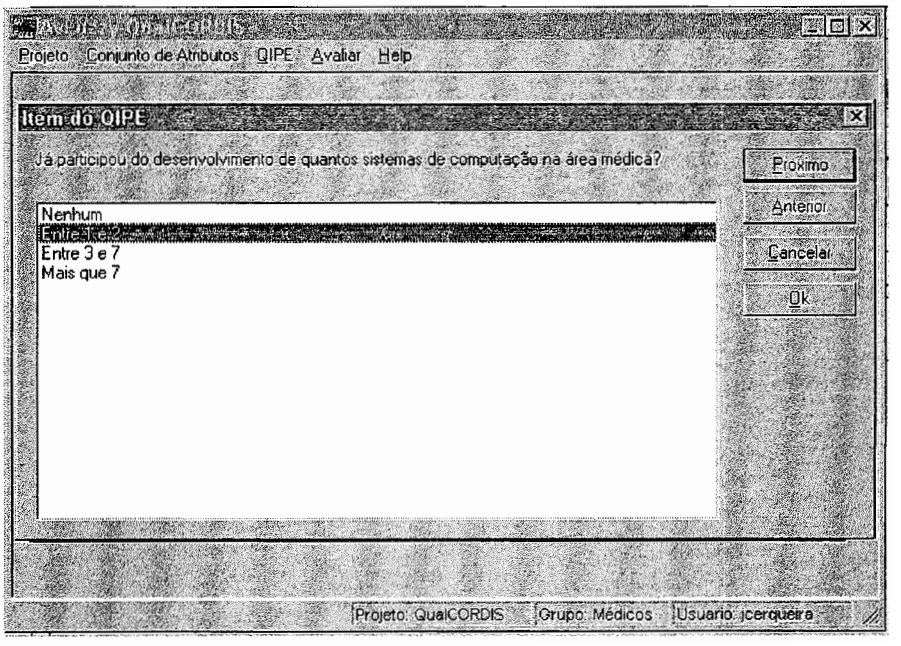

Figura 6.19 - Respondendo ao QIPE

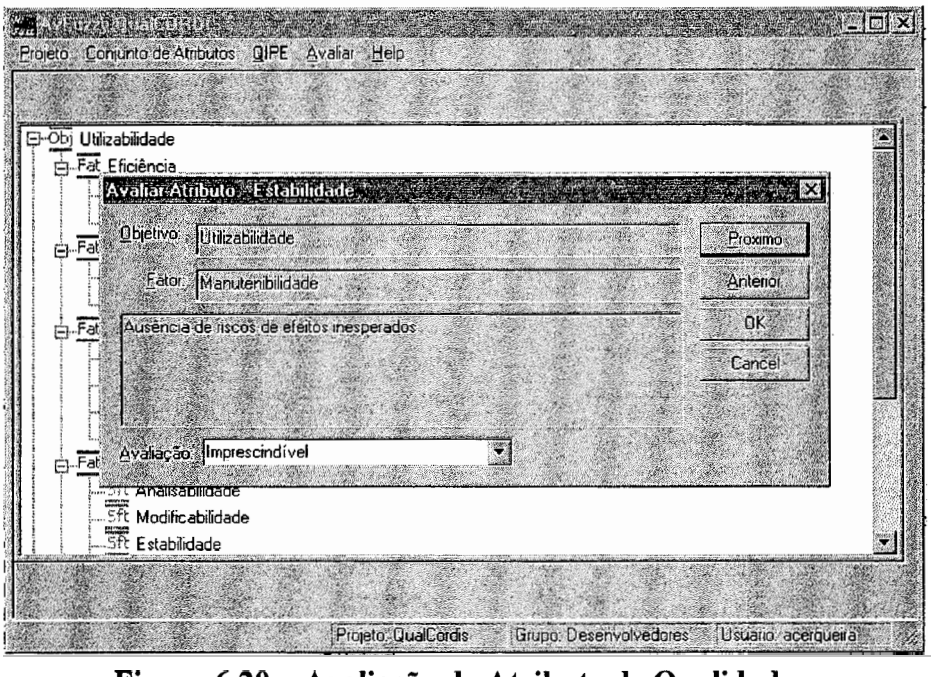

Figura 6.20 – Avaliação de Atributo de Qualidade

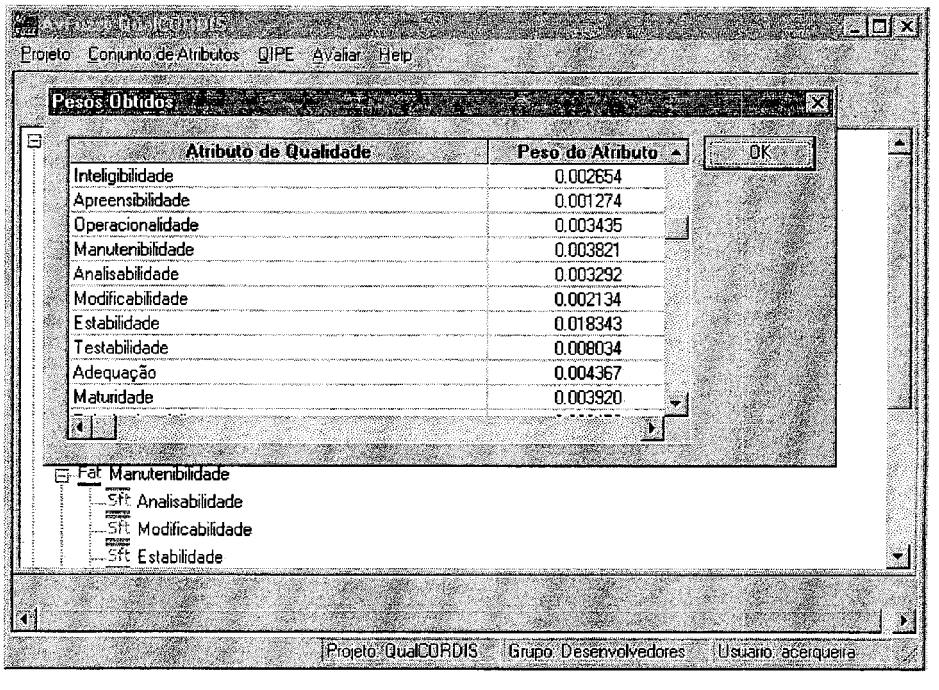

Figura 6.21 – Lista dos Requisitos de Qualidade

➢ KED: um exemplo de ferramenta específica do domínio

Ferramentas específicas apoiam a atividade de investigação do domínio (definida no capítulo 4) através do uso do conhecimento específico disponível na Estação TABA. Tendo como objetivos mostrar a viabilidade de construção de ferramentas usando a Teoria do Domínio e que essas ferramentas podem ser inseridas em um processo de software específico usado para instanciação de um ADS, construímos KED, um Editor de Conhecimento para diagnóstico em cardiologia

A decisão pela construção de um editor de conhecimento deve-se ao fato de que a elicitação do conhecimento é, reconhecidamente, o gargalo do desenvolvimento de sistemas baseados em conhecimento e que, geralmente, a principal dificuldade é a interação com os especialistas do domínio. Dessa forma, uma ferramenta específica do domínio que apoie essa atividade é de grande utilidade no desenvolvimento.

Para definição da arquitetura da ferramenta algumas premissas foram consideradas: *(i)* o reconhecimento de que ferramentas de elicitação são mais eficazes quando específicas para uma tarefa por direcionar o trabalho do especialista, *(ii)* o fato de especialistas médicos, em geral, explicarem seu raciocínio através do uso de casos, e, *(iii)* a definição de que a ferramenta deve usar a linguagem dos especialistas definida na Teoria do Domínio.

Dessa forma, as ferramentas de elicitação do conhecimento devem se basear na interação com o especialista através da entrada de casos possuindo uma arquitetura composta de três níveis (Figura 6.22): (i) o nível do conhecimento, que representa a(s) sub-teoria(s) do domínio correspondente à tarefa; (ii) o nível da modelagem da tarefa particular de cada ferramenta, e, (iii) o nível operacional, em que os casos utilizados no processo são armazenados em uma base de dados.

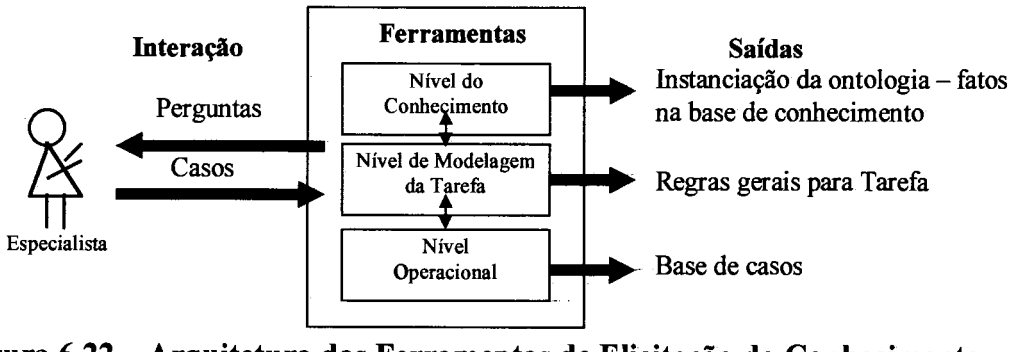

Figura 6.22 – Arquitetura das Ferramentas de Elicitação do Conhecimento

KED - Editor de Conhecimento para Diagnóstico em Cardiologia - (OLIVEIRA e MATWIN, 1998) é a primeira das ferramentas de elicitação do conhecimento desenvolvida com essa arquitetura sendo a tarefa de diagnose escolhida por ser uma das atividades médicas mais comuns. KED é uma ferramenta externa que possui integração de dados na Estação TABA e gera informações que são aproveitadas pelo seu meta-modelo.

Nesta ferramenta, a modelagem da tarefa foi definida usando padrões de análise (FOWLER, 1997) e considerando o modelo de diagnóstico sistemático baseado em causas do KADS (HICKMAN *et al.*, 1992). Este modelo KADS define que conceitos são causados por outros conceitos e estes, por sua vez, são causados por outros, formando uma hierarquia causal. Dessa forma, o processo de elicitação com o especialista consiste da entrada de casos (informações observadas no paciente e diagnóstico) e de justificativas do diagnóstico definidas pela associação dos dados do paciente. Essas justificativas são associações diagnosticas básicas usadas pelos especialistas e formarão a hierarquia causal. A modelagem conceitual da ferramenta conforme essas características esta apresentada na Figura 6.23 de acordo com a notação de padrões de análise (FOWLER, 1997).

KED realiza a generalização dessas justificativas, a partir de regras de generalização seletiva utilizadas nos algoritmos de aprendizado automático (MICHALSKI, 1986) que

se refere à substituição de *and* lógico por *or*, o *or* fechado e a eliminação de argumentos. Dessa forma, durante a elicitação com o especialista, a ferramenta é capaz de apresentar justificativas para um novo caso baseando-se em casos anteriores. Essas justificativas da ferramenta ajudam os especialistas a refinarem o conhecimento já fornecido e a se lembrarem de outros casos importantes. Esse processo de interação do conhecimento é conhecido como *lazy knowledge acquisition* (MASSEY, 1995). As regras de generalização, geradas pela ferramenta, através das justificativas do especialista, podem ser utilizadas para prototipação rápida durante a fase de elicitação do conhecimento.

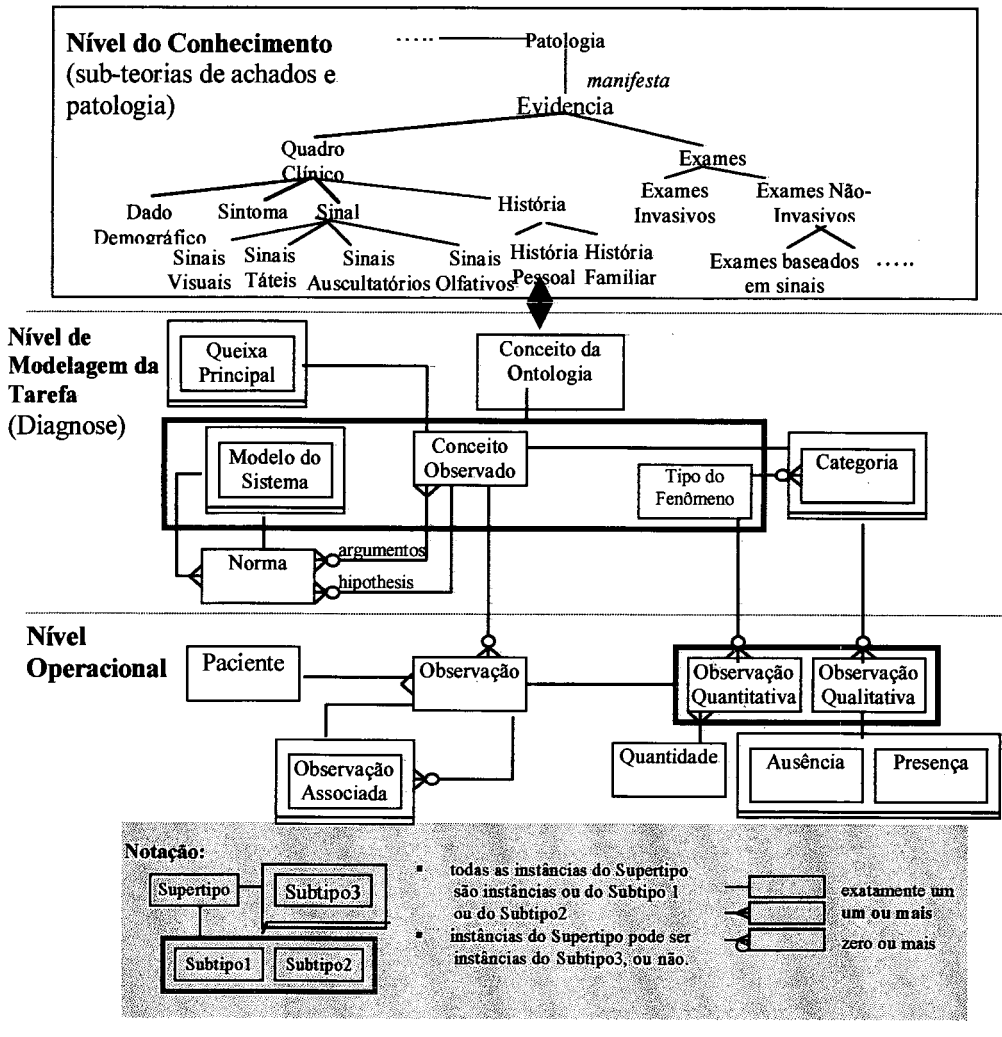

Figura 6.23 – Modelagem Conceitual de KED

O processo de elicitação do conhecimento é conduzido com o cardiologista através de três funções básicas: diálogo para entrada de casos, generalização de associações para justificativas dos casos e armazenamento das informações para uso posterior.

O ***diálogo para entrada de dados***, utilizado pelo KED, é baseado em formulários a serem preenchidos. Inicialmente, o especialista informa as características observadas em

um paciente. Em seguida, define o diagnóstico e o justifica identificando as observações que foram importantes para o seu diagnóstico realizando a associação simples das observações. A Figura 6.24 e 6.25 mostram exemplos da entrada de dados.

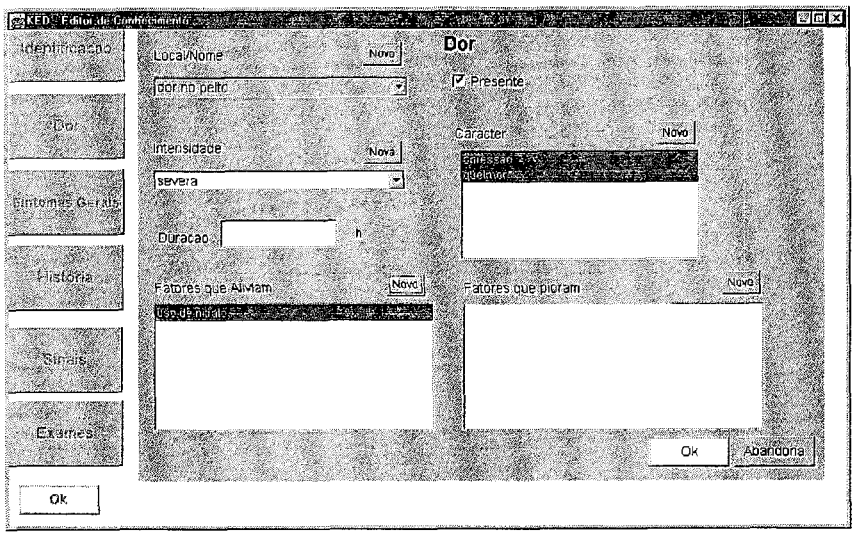

Figura 6.24 - Exemplo de Entrada de Dados
Características do Paciente

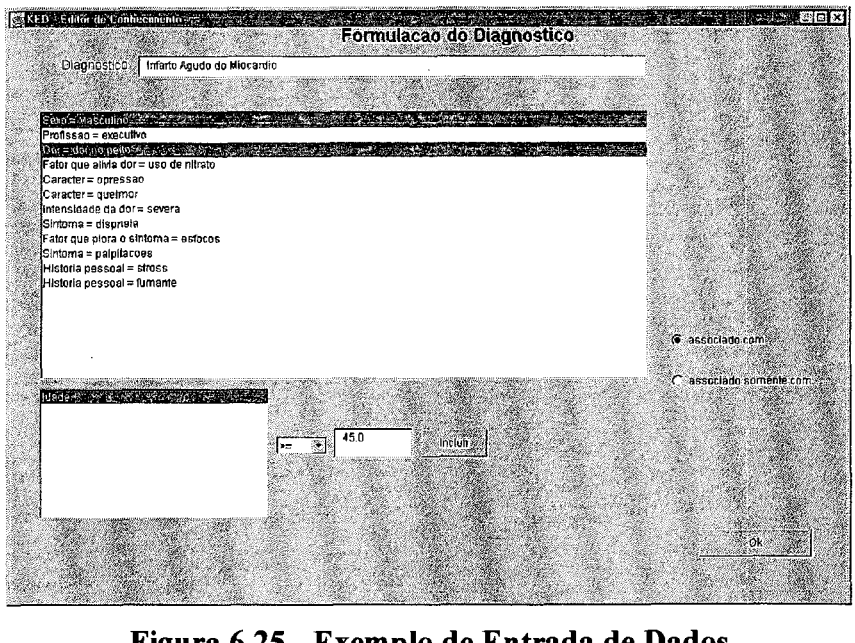

Figura 6.25 - Exemplo de Entrada de Dados
Formulação do Diagnóstico

A partir da entrada no sistema do segundo caso, KED investiga as justificativas de casos anteriores para procurar definir a justificativa para o diagnóstico do paciente através da ***generalização de associações***. A generalização dessas associações segue três regras básicas de generalização, utilizadas em algoritmos de aprendizado automático (MICHALSKI, 1986): eliminação de um dos elementos da associação, transformação de

conjunção para disjunção e definição de disjunção interna para um dos elementos da associação. KED mostra sua justificativa ao especialista que pode concordar ou discordar da generalização realizada. Neste último caso, KED aprende que essa justificativa não deve ser dada novamente em casos futuros. O especialista pode complementar as justificativas dadas por KED como novas associações importantes para o caso. Um exemplo da generalização provida por KED está mostrado na Figura 6.26.

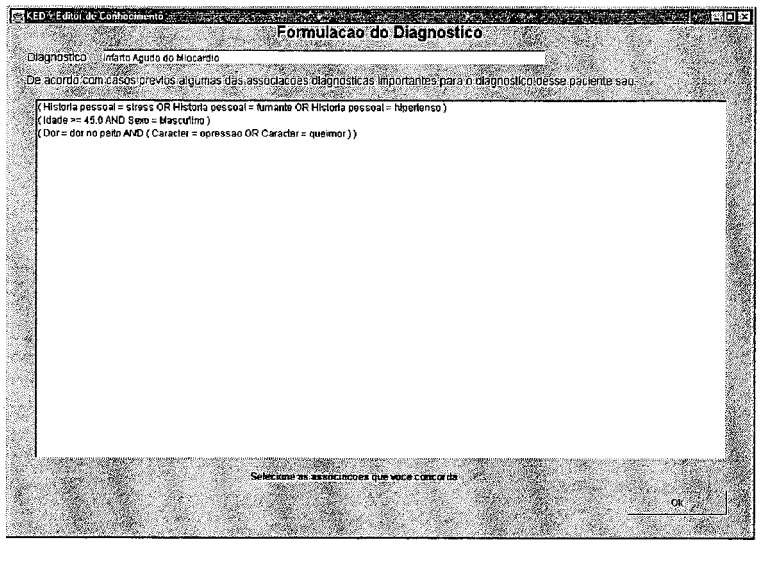

Figura 6.26 - Exemplo de generalização

Finalmente, KED ***armazena*** todos os casos em arquivos de dados assim como as associações geradas pelo processo de generalização. Essas associações podem ser usadas para prototipagem para nova elicitação com os especialistas ou, simplesmente, para discussão em entrevistas. Os casos podem, futuramente, ser utilizados para testes do sistema.

A funcionalidade da ferramenta KED foi testada com a entrada de casos coletados na UCCV/FBC e a definição de associações definidas por cardiologistas.

- **Edição do Processo e Instanciação do Ambiente**

A edição do processo foi realizada usando o EDIT-PRO definido no capítulo anterior. Dessa forma foram inseridas as atividades, sub-atividades, métodos, ferramentas e sequência das atividades. Após inserido o processo foi definido o ambiente (Figura 6.27), instanciado (Figura 6.28) e testado (Figura 6.29) sendo apresentado o ambiente CORDIS_SBC instanciado (Figura 6.30).

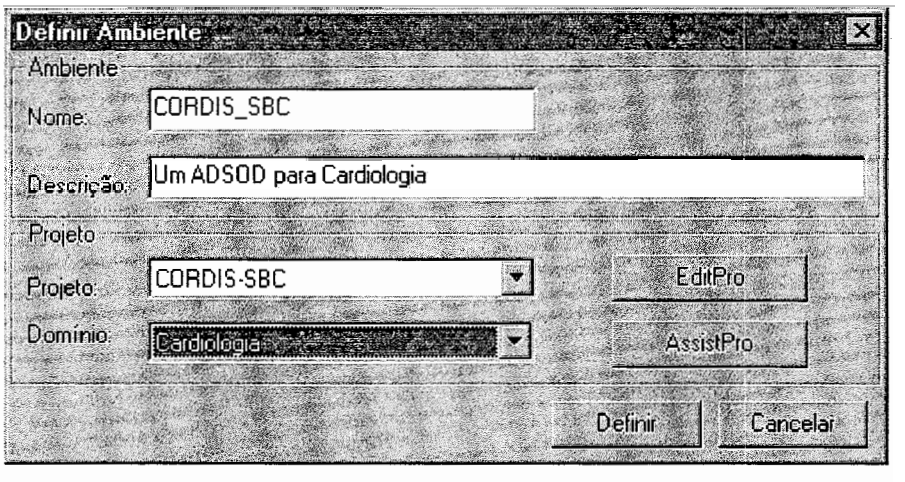

Figura 6.27 – Definição do ambiente

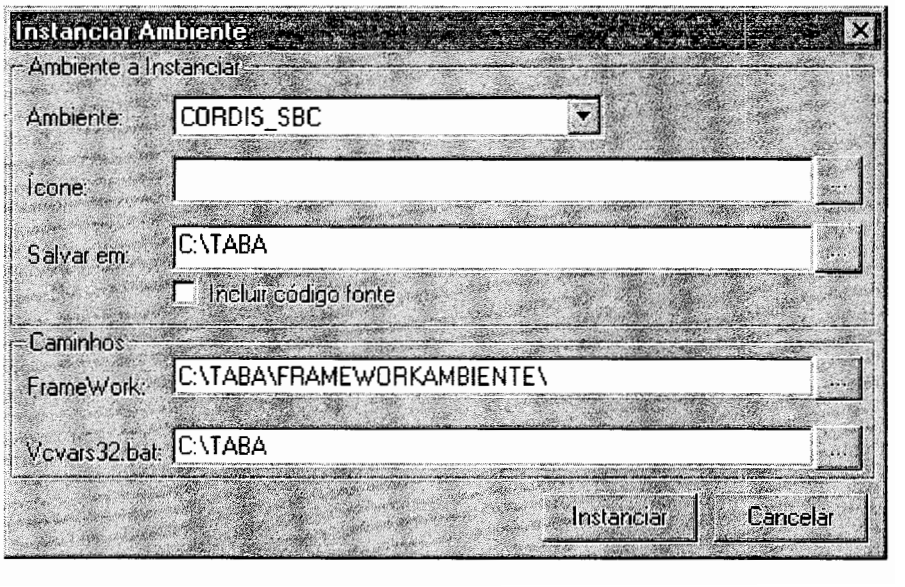

Figura 6.28 – Instanciação do ambiente

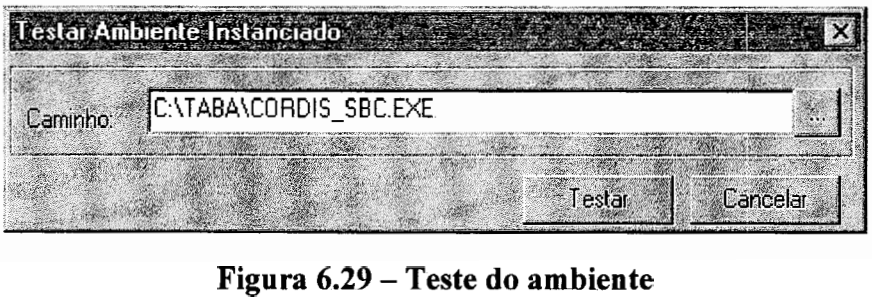

Figura 6.29 – Teste do ambiente

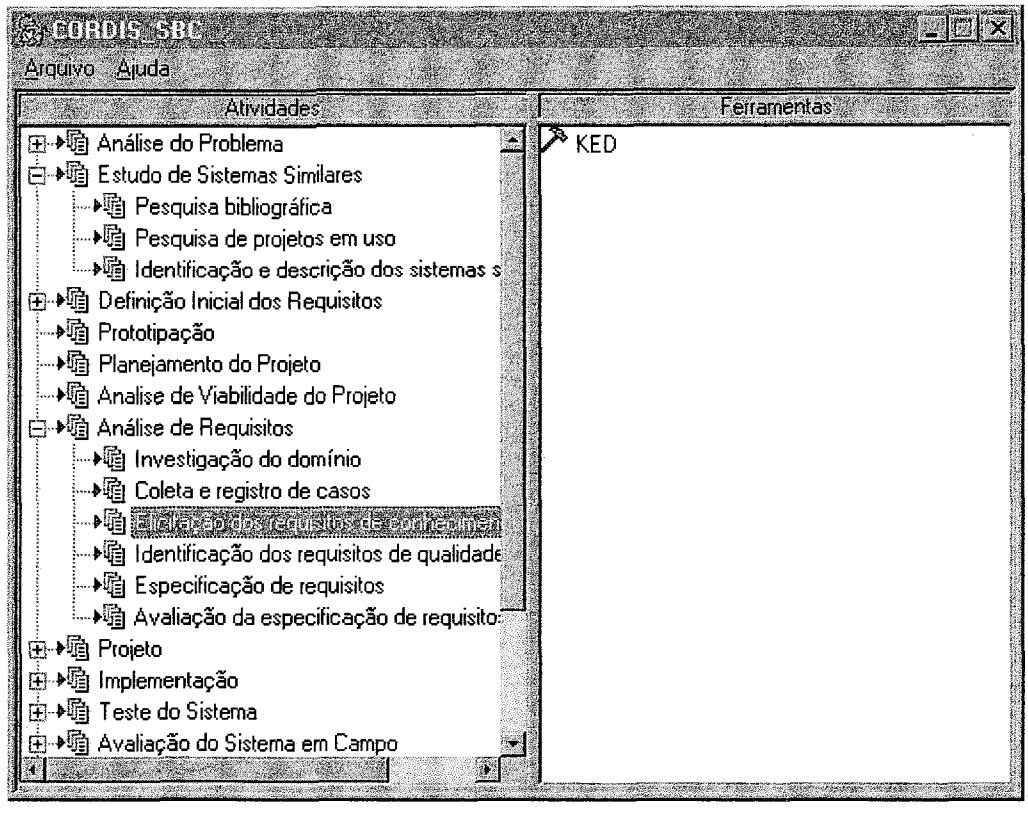

Figura 6.30 – CORDIS-SBC – Um ADSOD para Cardiologia

6.5 Considerações Finais

A construção de um ADSOD completo que atenda todos os requisitos estabelecidos no capítulo 5 é um projeto de realização a longo prazo para que seja possível a construção de todas as suas ferramentas. A definição e construção do seu núcleo principal deve, ser o primeiro passo a ser realizado. Consideramos esse núcleo principal, os aspectos que se referem à introdução do conhecimento no ambiente específico e à definição do seu uso integrado no processo de desenvolvimento. Nesse capítulo foi descrito como construímos esse núcleo principal, permitindo a instanciação do ADSOD CORDIS através da infra-estrutura TABA. Foi, também, mostrado como ferramentas gerais para ADSOD disponíveis na Estação TABA podem ser customizadas para um ADSOD específico (Qual-CORDIS) e um exemplo de ferramenta específica para o domínio (KED).

Capítulo 7

Conclusões e Perspectivas Futuras

Desenvolver software para um domínio onde não se tem familiaridade é, em geral, uma tarefa que envolve bastante dificuldade e riscos. Esta constatação, a partir da experiência no desenvolvimento de software para diferentes domínios, nos fez acreditar que a produtividade no desenvolvimento de produtos de software é, fortemente, influenciada pelo grau de conhecimento que os desenvolvedores tem do domínio, e que a disponibilização organizada e acessível do conhecimento do domínio pode auxiliar, de forma decisiva para o êxito, projetos de desenvolvimento de software. Para isso, definimos e construímos Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD).

ADSOD são ambientes de desenvolvimento de software que apoiam o desenvolvimento de software em domínios específicos através do uso do conhecimento deste domínio, durante o processo de desenvolvimento, buscando assim auxiliar os desenvolvedores no entendimento do domínio do problema. O domínio é uma área de aplicação, como por exemplo leis ou cardiologia, para a qual vários produtos de software podem ser desenvolvidos.

De acordo com a proposta deste trabalho, para construção de ADSOD, o conhecimento do domínio é organizado em um modelo, denominado Teoria do Domínio, que utiliza ontologias do domínio. O apoio no desenvolvimento é realizado através de uma atividade específica no processo de software, chamada de Investigação do Domínio. Essa atividade é apoiada por um conjunto de ferramentas do ADSOD, que utilizam as informações particulares do domínio e o conhecimento do domínio descrito na Teoria do Domínio.

Ambientes de desenvolvimento de software que consideram o domínio, como parte integrante de sua arquitetura, tem explorado a organização do domínio buscando o desenvolvimento de componentes para reutilização em futuros projetos (TAYLOR *et*

al., 1995, GOMMA *et al.*, 1996) e o apoio à geração automática de especificações (LOWRY *et al.*, 1994, LOWRY e BAALEN, 1997). Esses ambientes baseiam-se na definição de modelos de domínio, definidos através da análise de domínio, ou seja, o conhecimento reutilizável é identificado a partir da análise de sistemas existentes.

Outro trabalho com enfoque no domínio vem sendo realizado por FISCHER *et al* (1996) que se preocupa com a iteração entre as pessoas envolvidas no projeto de um produto, através do conhecimento do domínio. No entanto, Fischer se concentra em experiências de apoio à atividade de projeto (desenho) conforme realizada por arquitetos e desenhistas não tendo contribuições ao processo de desenvolvimento de software

Os ADSOD, conforme propomos, procuram concentrar o apoio ao domínio numa etapa anterior a destas propostas. Nosso objetivo é apoiar os desenvolvedores no entendimento do domínio oferecendo conhecimento relevante sobre o domínio, ao longo de todo o processo de desenvolvimento. Dessa forma, o enfoque não é a construção de componentes que possam ser reutilizados em novos desenvolvimentos, e sim a organização do conhecimento do domínio, preocupando-nos exclusivamente com este conhecimento em si e não com aspectos do desenvolvimento de futuros produtos de software. O conhecimento, assim organizado, por estar descrito sem utilizar representações específicas fruto de atividades de desenvolvimento, é passível de reutilização em qualquer atividade do desenvolvimento que necessite de conhecimento do domínio. Isso é possível pela organização do conhecimento do domínio utilizando ontologias.

Os trabalhos anteriores que utilizam ontologias, descritos na literatura tem se concentrado principalmente no apoio ao desenvolvimento de sistemas baseados em conhecimento (NETCHES *et al.*, 1991, SWARTOUT *et al.*, 1994, SCHREIBER *et al.*, 1995, VALENTE *et al.*, 1999 OUSSALAH e MESSAADIA,1999, MOTTA *et al.*,1999) pelo fato de ontologias serem um bom esquema conceitual para a construção de bases de conhecimento. No entanto, esses trabalhos não utilizam ontologias para apoiar o entendimento do domínio durante as atividades específicas do desenvolvimento destes sistemas.

São, portanto, contribuições dessa tese:

- A definição de ADSOD e sua construção no contexto da Estação TABA, o que implicou na:
    - definição dos requisitos de ADSOD;

- extensão do modelo TABA, de forma a permitir a instanciação de ADSOD;
- definição de um modelo para definição de processos instanciáveis na Estação TABA, para projetos específicos, a partir de um processo padrão definido para a organização, constituindo assim, famílias de processos e, consequentemente, de ADS;
- utilização de ontologias para a orientação a domínio no desenvolvimento de software. Ontologias são usadas para organização do conhecimento do domínio em uma Teoria do Domínio, possível de ser utilizada para entendimento do domínio ao longo do desenvolvimento e para construção de ferramentas específicas do domínio;
- re-implementação de um conjunto de funcionalidades da Estação TABA de forma a permitir a instanciação de ADS;
- definição e implementação de ferramentas específicas para ADSOD: o editor de Teoria do Domínio e Q-Fuzzy. Estas ferramentas genéricas são passíveis de customização para ADSOD específicos;
- definição da arquitetura de ferramentas específicas para elicitação de conhecimento no domínio a partir de ontologias.

- A construção do ambiente CORDIS, um ADSOD específico para Cardiologia, o que implicou na:
  - definição de uma ontologia (Teoria do Domínio) para Cardiologia;
  - construção de um servidor de conhecimento para Cardiologia;
  - definição e construção de uma ferramenta específica para o domínio a partir da ontologia: KED, uma ferramenta para apoio à elicitação do conhecimento de diagnóstico em Cardiologia;
  - organização de características de qualidade de software médico e customização da ferramenta Q-*fuzzy* para o domínio de Cardiologia, gerando a ferramenta Qual-Cordis.

A pesquisa em ADSOD não se esgota com os resultados dessa tese. Esse foi, apenas, um primeiro passo embora fundamental, em direção a um ADS que apoie as atividades presentes no desenvolvimento de software considerando o conhecimento do domínio, modelado e implementado. Várias outras pesquisas serão realizadas. Algumas já estão em andamento como teses de mestrado e doutorado na COPPE/UFRJ:

- a definição e construção do ADSOD NETUNO, para o domínio de Acústica Submarina que valida as idéias deste trabalho para um outro domínio e implementa uma ferramenta para apoio ao aprendizado do domínio utilizando a Teoria do Domínio (GALOTTA e ROCHA, 1999);
- a especificação e construção de uma ferramenta para apoiar a definição de processos de software desde o processo padrão até o processo para um projeto específico possível de ser instanciado pela Estação TABA. Este trabalho semi-automatiza a definição de processos de software, considerando a norma ISO 12207, o SPICE ou CMM e as características da organização e dos projetos específicos (MACHADO e ROCHA, 1999).
- a definição, construção e incorporação ao ADSOD de Bases de Experiências, com dados de projetos anteriores já desenvolvidos em uma organização que sirvam de base para o planejamento de novos projetos, e, de uma Base de Componentes Reutilizáveis, usando a Teoria do Domínio como meio para a recuperação de componentes, tornando, assim, os ADSOD mais orientados à uma organização específica. Estas características serão incorporadas ao ambiente CORDIS, com dados do desenvolvimento de software na UCCV/FBC;
- o uso de ontologia do domínio para modelagem de dados e definição do esquema de banco de dados relacional, o que trará facilidades para a construção de Bancos de Dados de apoio a pesquisas, construídos a partir da ontologia do domínio. Neste trabalho de tese o foco de estudo será na área de Entomologia[1];
- a definição e construção de uma ferramenta para avaliação de processos de software ao longo do desenvolvimento e após a sua conclusão, o que permitirá a coleta de dados para melhoria de processos;
- a extensão da ferramenta Q-*fuzzy* para permitir a avaliação de produtos de software. ao longo do desenvolvimento e do produto final, segundo os requisitos de qualidade previamente estabelecidos, o que requer a definição de critérios e métricas específicas para os atributos de qualidade de forma a permitir a medição.

[1] Entomologia é uma disciplina da agronomia/zoologia que estuda os insetos.

Outros trabalhos que devem ser realizados envolvem:

- a conclusão da re-implementação da Estação TABA, especialmente o que se refere a integração de ferramentas e ao acompanhamento de projetos. Estes dois aspectos são fundamentais para a Estação TABA e não foram implementados nesta primeira etapa da re-implementação;
- a disponibilização de outras linguagens de representação de conhecimento, além do Prolog, na Estação TABA que possa ser utilizada na edição da Teoria do Domínio;
- a extensão do Editor de Teoria do Domínio para permitir a geração automática de axiomas epistemológicos e o desenho gráfico do modelo;
- o uso de ontologias de tarefas, organizando a descrição de tarefas, de forma que ontologias possam ser utilizadas não apenas para auxiliar o entendimento do domínio mas, também, serem usadas e reutilizadas para o desenvolvimento de sistemas baseados em conhecimento;
- a pesquisa para definição e construção de bases de projetos desenvolvidos utilizando a Teoria do Domínio, indexadas pelo conceitos definidos na Teoria do Domínio de forma que possam ser utilizadas como fonte de exemplos para aprendizado no desenvolvimento de novos sistemas e acessadas utilizando a Teoria do Domínio, e,
- a introdução de arquiteturas de referência na Estação TABA que possam ser utilizadas em diferentes tipos de projetos no mesmo domínio;
- a definição e construção de novas ferramentas para o ambiente CORDIS bem como a incorporação de ferramentas externas;
- avaliação de experiências de uso dos ADSOD em situações reais de desenvolvimento de software.

Portanto, muito trabalho ainda deve ser realizado. Os resultados e contribuições apresentados, nesta tese, representam um primeiro passo na direção de ADSOD definindo e implementando os seus principais requisitos. Acreditamos que o conhecimento do domínio é um fator determinante para o êxito no desenvolvimento de software e que os ADSOD, propostos neste trabalho podem contribuir decisivamente para o desenvolvimento de produtos de software com maior qualidade e produtividade.

Referências Bibliográficas

ABU-HANNA, A., JANSWEIJER, W., 1994, "Modeling Domain Knowledge Using Explicit Conceptualization". *IEEE Expert* (Oct), pp 53-63.

AGUIAR, T. C., 1992, "Um Sistema Especialista de Suporte à Decisão para Planejamento de Ambiente de Desenvolvimento de Software". Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

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

ARANGO, G. E PRIETO-DÍAZ, R. (eds.), 1991, "Part 1: Introduction and Overview - Domain Analysis Concepts and Research Directions", In: *Domain Analysis and Software Systems Modelling*, chapter 1, pp 9-32.

ARANGO, G., 1994, "Domain Analysis Methods", In: Schäfen, W., Prieto-Díaz, R. Masumoto, M. (eds), *Software Reusability,* pp 17-49, Ellis Horwood Inc.

ARAÚJO, M. A. P., 1998, Automatização do Processo de Desenvolvimento de Software nos Ambientes Instanciados pela Estação TABA, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

ASSIS, S. G., 1992, "Aquisição de Conhecimento para Sistemas Baseados no Conhecimento: Uma Experiência em Engenharia de Software". Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

AZEVEDO, A C, 1984, *Cardiologia*, Sarvier.

BANDINELLI, S., DI NITTO, E., FUGGETTA, A., 1996. "Supporting Cooperation in the SPADE-1 Environment", *IEEE Transactions on Software Enngineering*, v. 22, n. 12 (Dec), pp. 841-865.

BANDINELLI, S., FUGGETTA, A., LAVAZZA, L., LOI, M., PICCO, G. 1995. "Modelling and Improving an Industrial Software Process", *IEEE Transactions on Software Enngineering*, v. 21, n. 5 (May), pp. 440-453.

BARROS, M. O., 1996, Recuperação de Componentes em Biblioteca de Software: Uma Abordagem Conexionista. Tese de M. Sc., COPPE/UFRJ Rio de Janeiro, RJ, Brasil.

BELCHIOR, A. D. 1997, *Um Modelo Fuzzy para Avaliação da Qualidade de Software*, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

BELCHIOR, A. D., XEXEO, G., ROCHA, A. R. C. DA 1997, "Um Modelo Fuzzy para Avaliação da Qualidade de Software", In: *Anais do XI Simpósio Brasileiro de Engenharia de Software*, Fortaleza, Brasil, Out.

BERNAS, P., 1995, "The Palas-X SDE". In: *Proceedings of Software Engineering Environments*, pp. 126-134, Noordwikkerhout, The Netherlands, Apr.

BISCHOFBERGER, W. R., KOFLER, T., MATZEL, K. U., SCHAFFLER, B., 1995, "Computer Supported Cooperative Software Engineering with Beyond-Sniff". In: *Proceedings of Software Engineering Environments*, pp. 126-134, Noordwikkerhout, The Netherlands, Apr.

BLASCHECK, 1995, *Planejamento Estratégico de Sistemas de Informação,* Tese de D. Sc., COPPE/UFRJ Rio de Janeiro, RJ, Brasil.

BODENREIDER, O., BURGUN, A. BOTTI, G. *et al.*, 1998, "Evaluation of The Unified Medical Language System as a Medical Knowledge Source". *JAMIA, Journal of the American Medical Informatics Association*, v. 5, n. 1 (jan/fev), pp. 76-87.

BOLCER, G. A., 1995, "User Interface Design Assistance for Large-Scale Software Development", *Automated Software Engineering*, v. 2, n. 3 (Sep).

BOOCH, G., 1994, *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings, 2ª ed, Menlo Park, Califórnia, USA.

BOOCH, G., RUMBAUGH, J., JACOBSON, I. *et al.*, 1997, *The Unified Modeling Language – User Guide*, Addison-Wesley.

BRAGA, R., TRAVASSOS, G. H., 1998, *Arquiteturas de Software e Ambientes de Desenvolvimento de Software*, Relatório Técnico COPPE/UFRJ TABA RT-19/98, Rio de Janeiro, RJ, Brasil.

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

BROWN, A., EARL, A., MCDERMID, J., 1992, *Software Engineering Environments: Automated Support for Software Engineering*, England, McGraw-Hill Book Company.

BROWN, A., W., 1993, "An Examination of the Current State of IPSE Technology". In: *Proceedings of the 15th International Conference on Software Engineering*, pp. 338-347, Baltimore, Maryland, May.

CAMPBELL, GRADY H., Jr., FAULK, *et al.,* 1990, "Introduction to Synthesis. ", *INTOR_SYNTHESIS_PROCESS-900019-N.* Herndon, VA, Software Productivity Consortium.

CAMPOS, F. A., 1994a, "Qualidade de Aplicações Hipermídia", Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

CAMPOS, G. H., 1994b, "Metodologia para avaliação da qualidade de software educacional: Diretrizes para desenvolvedores e usuários", Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

CANALS, G., CHAROY, F., GODART, C. al. 1995, "Suport fpr Collaborative, Integrated Software Development". In: *Proceedings of Software Engineering Environments*, pp. 2-10, Noordwikkerhout, The Netherlands, Apr.

CARVALHO, D. O., 1997, *Qualidade de Sistemas de Informação Hospitalar*, Tese de M, Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

CERQUEIRA, A., OLIVEIRA, K. M., ROCHA, A. R., 1999, "Apoio Automatizado para a Definição de Requisitos de Qualidade de Software utilizando Teoria *Fuzzy*", In: *Anais do Workshop de Qualidade - XIII Simposio Brasileiro de Engenharia de Software*, Florianópolis, Brasil, Out.

CHANDRASEKARAN, B., *et al.*, 1993, "Task-Structure Analysis for Knowledge Modeling". In: B. V. Cuena (ed.), *Knowledge Oriented Software Design*, Elsevier Science Publishers.

CHANDRASEKARAN, B., JOSEPHSON, J. R., BENJAMINS V. R., 1999, "What Are Ontologies, and Why Do We Need Them?". *IEEE Intelligent Systems & their applications*, v. 14, n. 1 (Jan/Feb), pp. 20-26.

CHATZOGLOU, P. D., MAKAULAY, L. A., 1998, "A Rule-Based Approach to Developing Software Development Prediction Models", *Automated Software Engineering*, v. 5, n. 2 (Apr), pp. 221-243.

CHRISTIE, A. M., 1995, *Software Process Automation- The Technology and its Adoption*, Peittsburghm Pennsylvannia, Springer-Verlag Berlin Heidelberg.

CHURCH, T., MATTHEWS, P., 1995, "An Evaluation of Object-Oriented CASE Tools: The Newbridge Experience". In: *Proceedings of 7th International Workshop on Computer Aided Software Engineering*, pp. 4-9, Toronto, Canada, Jul.

CIMA, A. M., 1996, "Desenvolvimento de Software Com Reutilização Baseada em Frameworks Orientados a Objetos". Tese de M. Sc., COPPE/UFRJ Rio de Janeiro, RJ, Brasil.

CLEMENTS, P., 1996, *Coming Attractions in Software Architecture*. Technical Report CMU/SEI-96-TR-008, Jan.

COELHO, A. C. B., 1997, *Guias de Qualidade para Construção de Frameworks Orientados a Objetos*. Tese de M. Sc., COPPE/UFRJ Rio de Janeiro, RJ, Brasil.

COHEN, S., 1994, "Feature-Oriented Domain Analysis: Domain Modelling", In: *Tutorial Notes - 3rd International Conference on Software Reuse*, Rio de Janeiro, Brazil.

CONRADI, R., HAGASETH, M., LARSEN, J-O., *et al.*, 1994, "EPOS: Object-Oriented and Cooperative Process Modelling". In: Finkelstein, A., Kramer, J., Nuseibeh, B. (eds), *Software Process Modelling Technology*, pp. 33-70, Adavanced Software Development Series Research Press Ltd.

CONRADI, R., KARLSSON, E-A, 1995. "The REBOOT Approach to Software Reuse", *Journals of Systems and Software* (special issue on software reuse), v. 30, n. 3 (Sep), pp. 201-212.

CRISPIM, M. E. H., 1991, *Avaliação de Métodos de Desenvolvimento de Software*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

DAVIS, M. J., WILIANS, R. B., 1997, "Software Architecture Characterization". In: *Proceedings of Simposium on Software Reuse* (SSR'97), pp. 30-38, Massachutes, USA.

EDEN, H., EISENBERG, M., FISCHER, G. *et al.*, 1996, "Making Learning a Part of Life", *Communications of the ACM*, pp. 40-42.

EISENBERG, M., FISCHER, G., 1994, "Programmable Design Environments: Integrating End-User Programming with Domain-oriented Assistance", In: *Proceedings of Conference in Human Factors in Computer Systems* – CHI'94, pp. 431-437, Boston, Massachusetts, USA, Apr.

EMAM, K. E., DROUIN, J. AND MELO W., 1998, SPICE – The Theory and Practice of Software Process Improvement and Capability Determination, IEEE Computer Society Press.

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

FALBO, R., MENEZES, C., ROCHA, A. R., 1999b, "Assist-Pro: Um Assistente Inteligente para Apoiar a Definição de Processos de Software", In: *Anais do XIII Simposio Brasileiro de Engenharia de Software*, pp. 147-162, Florianópolis, Brasil, Out.

FALBO, R., MENEZES, C., ROCHA, C., 1999a, "Using Knowledge Servers to Promote Knowledge Integration in Software Engineering Environments". In: *Proceedings of the 11th Software Engineering an Knowledge Engineering Conference*, pp. 170-175, Kaiserslautern, Alemanha, Jun.

FAVARO, J., 1997, "Standardizing Production of Domain Components", *Standard View*, vol 5., n. 2. (Jun), pp 66-69.

FERNÁNDEZ, M., GÓMEZ-PÉREZ A, JURISTO N., 1997, "METHONTOLOGY: From Ontological Art Towards Ontological Engineering", In: *Proceedings of AAAI97 Spring Symposium Series - Ontological Engineering*, pp 25-32, Stanford University, EUA, Mar.

FIKES, R., FARQUHAR, A., 1999, "Distributed Repositories of Highly Expressive Reusable Ontologies", *IEEE Intelligent Systems & their applications*, v. 14, n. 2 (Mar/Apr), pp. 73-79.

FISCHER, G., 1996, "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.

FISCHER, G., GINGENSOHN, A., NAKAKOJI, K. *et al.,* 1992a, "Supporting Software Designers with Integrated Domain-Oriented Design Environments", *IEEE Transactions on Software Engineering*; v. 18, n. 16 (Jun), pp 511-522.

FISCHER, G., LINDSTAEDT, S., PSTWALD, J. *et al.* 1995, "From Domain Modelling to Colaborative Domain Construction", In: *Proceedings of DIS – Symposium on Interactive Systems*, pp. 75-85, Ann Arbor, USA.

FISCHER, G., MCCALL, R., OSTWALD, J. *et al.,* 1994, "Seeding, Evolutionary and Reseeding: Supporting the Incremental Development of Design Environments", *Proceedings of Conference in Human Factors in Computer Systems – CHI'94*, , pp. 292-298, Boston, USA, Apr.

FISCHER, G., NAKAKOJI, K., OSTWALD, J. *et al.,* 1993, "Embedding critics in design environments", *The Knowledge Engineering Review*, v. 4 8:4, pp 283-307.

FISCHER, G; "Domain-Oriented Design Environments", 1992b, In: *Proceedings of The Seventh Knowledge-Based Software Engineering Conference,* pp 204-213, McLean, Virgínia; Sep.

FISCHER, G, 1994; "Domain-Oriented Design Environments", Automated Software Engineering - The International Journal of Automated Reasoning and Artificial Intelligence in Software Engineering, Vol 1, N 2 (Jun), pp 177-203.

FOX, M. S, BARBUCEANU, M., GRUNINGER, M, 1991, "An Organisation for Enterprise Modelling: Preliminary Concepts for Linking Structure and Behaviour", In: *4th Workshop on Enabling Tecnhologies: Infrastructure for Collaborative Enterprises,* pp 71-72, Berkeley Springs, USA, Apr.

FREITAS, A. L. C., PRICE, A. M. A., 1998, "Formalização de Heurísticas para o Apoio a Modelagem de Sistemas Orientados a Objetos". In: *Anais do XII Simpósio Brasileiro de Engenharia de Software*, pp. 313-328, Maringá, Brasil, Out.

FRIDMAN, N., HAFNER, C. D., 1997, "The State of the Art in Ontology Design - A Survey and Comparative Review", *AI Magazine*, (Fall), pp 53-74.

FRIDMAN, N., HAFNER, C. D., 1997, "The State of the Art in Ontology Design – A Survey and Comparative Review", *AI Magazine*, (Fall), pp. 53-73.

GAMA, C. A., SOUZA J. M., RABELO A., 1997, "HyperClinic: Cooperative Hypermedia System for Medical Education", In: *Proceedings of Educational Multimedia and Hypermedia, ED-MEDIA*, Calgary, Canada.

GAMBIR, S. D. G., 1997, "Use of Domain Analysis to Implement the Developer off-the-shelf systems (DOTSS) System Acquisition Approach"; *Software Engineering Notes*, vol 22, no 2 (Mar), pp 48-53.

GAMMA, E., HELM, R., JOHNSON, R. et al., 1995, Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA

GARG, P., JAZAYERI, M., 1995, "Introduction". In: Garg, P., Jazayeri, M. (eds), *Process-Centered Software Engineering Environments*, chapter 1, IEEE Computer Society Press.

GARLAN, D., ALLEN, R. *et al.*, 1994, "Exploiting Style in Architectural Design Environments". In: *Proocedings of he ACM SIGSOFT'94 - Symposium on Foundations of Software Engineering*, New Orleans, USA, Dec.

GÓMEZ-PÉREZ, A, 1995, "Some Ideas and Examples to Evaluate Ontologies"; In: *Proceedings of 11th Conference in Artificial Intelligence for Applications*, pp 299-305, Los Angeles, EUA, Feb.

GÓMEZ-PÉREZ, A., FERNANDEZ, M., VICENTE, A. J., 1996, "Toward a Method to Conceptualize Domain Ontologies", In: *Proceedings of Workshop on Ontological engineering/ECAI96*, Budapest, Hungary, Aug.

GÓMEZ-PÉREZ, A., JURISTO, N. ; PAZOS, J., 1995, "Evaluation and Assessment of the Knowledge Sharing Technology", In: Mars (ed.), N. J., *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, IOS Press, Amsterdan, pp289-296.

GOMMA, H, KERSCHBERG, SUGMARAN, V. *et al.*, 1996, "A Knowledge-Based Software Engineering Environment for Reusable Software Requirements and Architectures", *Automated Software Engineering*, v. 3 n. ¾ (Aug), pp. 285-387.

GOMMA, H., KERSCHBERG, L., 1995, "Domain Modelling for Software Reuse and Evolution", In: *Proceedings of the 7th International Workshop on Computer Aided Sofrware Engineering*, pp. 162-171, Toronto, Canada.

GOMMA, H., KERSCHBERG, SUGMARAN, V. *et al.*, 1996, "A Knowledge-Based Software Engineering Environment for Reusable Software Requirements and Architectures", *Automated Software Engineering - The International Journal of Automated Reasoning and Artificial Intelligence in Software Engineering*, v3 no 3-4 (Aug), pp 285-307.

GOMMA, H., SUGUMARAM, V., BOSCH, C. *et al.*, 1994, "A Prototype Domain Modeling Environment for Resable Software Architectures", In: *Proceedings of the 3rd International Conference on Software Reuse*, pp. 74-3, Rio de Janeiro, Brasil.

GRISOLIA, S. , CORREIA, P., D'AJUDA, M. *et al.* 1997, "Implementação do Prontuário Médico Eletrônico de uma Unidade Coronariana com Apoio de Computação Móvel", In: *Anais do Congresso Brasileiro de Cardiologia - Arquivos Brasileiro Cardiologia*, Vol 69 (suplemento I).

GROTH, B., HERRMANN, S., JAHNICHEN, S., KOCH, W., P., 1995, "Project Integration Reference Object Library (PIROL): An Object-Oriented Multiple-View SEE". In: *Proceedings of Software Engineering Environments*, pp. 126-134, Noordwikkerhout, The Netherlands, Apr.

GRUBER, T., 1991, "The Role of Commom Ontology in Achieving Sharable, Reusable Knowledge bases", In: *Principles of Knowledge representation and reasoning: Proeceedings of the Second Internatioal Conference*, Cambridge, MA: Morgan Kaufmann; pp 601-602.

GRUBER, T., 1993, "A Translation Approach to Portable Ontology Specifications"; *Knowledge Acquisition,* 5(2), pp 199-220.

GRUBER, T. R., 1992, *Ontolingua: A mechanism to support portable ontologies, version 3. 0.* Technical Report, Knowledge Systems Laboratory, Stanford University, USA.

GRUBER, T. R., 1995; "Toward Principles for the Design of Ontologies used for Knowledge Sharing", *International Journal Human-Computer Studies*, No 43, pp 907-928.

GRUNDY, J., MUGRIDGE, W. B., HOSKING, J. G., AMOR, R. W, 1995, "Suport for Collaborative, Integrated Software Development". In: *Proceedings of Software Engineering Environments,* pp. 84-93, Noordwikkerhout, The Netherlands, Apr.

GRUNINGER, M., FOX, M. S. N., 1995, "Methodology for the Design and Evaluation of Ontologies"; In: *Proceedings of Workshop on Basic Ontological Issues in Knowledge Sharing/IJCAI95,* Montreal, Canada; Aug.

GUARINO, N., 1995, "Formal ontology, conceptual analysis, and Knowledge representation", *International Journal of Human-Computer Studies*; vol. 43, n. 5/6, pp 625-639.

GUARINO, N., 1997a, "Undesrtanding, Building, and using Ontologies: A comentary to 'Using Explicit Ontologies in KBS Development', by Heijst, Screiber e Wielenga", *International Journal Human-Computer Studies*, vol 46, No 2/3, pp 293-310.

GUARINO, N., 1997b, "Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction and Integration", In: *Information Extraction: a Multidisciplinary Approach to an Emerging Information Tecnhology*, M. T Pazienza (ed.), Springer Verlag, pp-139-170.

GUARINO, N., 1998, "Formal Ontology abd Information System", In: Guarino, N. (ed) *Formal Ontology in Information System,* pp. 3-15, IOS Press.

GUARINO, N., GIARRETA, P., 1995, "Ontologies and Knowledge Bases - Towards a Terminological Clarification", In: Mars, N. J., (ed.), *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, IOS Press, Amsterdan, pp 25-32.

GUHA, R. V., LENAT, D. B., 1994, "Enabling Agents to Work Together", *Communications of the ACM,* Vol 37, No 7 (Jul), pp127-142.

HARDY, C., STOBART, S., THOMPSON, B., EDWARDS, H., 1995, "A Comparison of the Results of Two Surveys on Software Development and the role of CASE in

the UK". In: *Proceedings of 7th International Workshop on Computer Aided Software Engineering*, pp. 234-238, Toronto, Canada, Jul.

HARVEL, D., NAAMAD, A., 1996. "The STATEMATE Semantics of statecharts", *ACM Transactions on Software Engineering and Methodology*, vol 5., n. 4, pp 293-333. Oct.

HAYES-ROTH, B., PLEGER, K., LALANDA, P. *et al.*, 1995, "A Domain-Specific Software Architecture for Adaptative Intelligent Systems", *IEEE Transactions on Software Engineering*, v. 21, n. 4 (Apr), pp. 288-301.

HENNINGER, S., 1996, "Building an Organization-Specific Infrastructure to Support CASE Tools", *Automated Software Engineering*, v. 3, n. ¾ (Aug), pp. 239-259.

HICKMAN, F, KILLIN, J, LAND, L et al., 1992, Analysis for Knowledge-Based Systems: A practical Guide for KADS Methodology. Ellis Horwood.

ISO/IEC 9126/NBR 13596, 1996, Tecnologia de Informação – Avaliação de Produto de Software – Características de Qualidade e Diretrizes para o seu uso, ABNT – Associação Brasileira de Normas Técnicas.

JACOBSON, I., GRISS, M., JONSSON, P., 1997, *Software Reuse – Architecture Process and Organization for Business Success*, 1ª ed., ACM Press, New York.

KALFOGLOU e ROBERTSON, 1999, "A Case Study in Applying Ontologies to Augment and Reason about Correctness of Specification, In: *Proceedings of the 11th Software Engineering an Knowledge Engineering Conference*, pp. 170-175, Kaiserslautern, Alemanha, Jun.

KROGSTIE, J., 1995, "Use of Development and CASE-tools in Norway: Results from a Survey". In: *Proceedings of 7th International Workshop on Computer Aided Software Engineering*, pp. 239-248, Toronto, Canada, Jul.

LENAT, D. B., 1995a., "Steps to Sharing Knowledge", In: Mars, N. J. (ed.), *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, IOS Press, Amsterdan, pp 3-6.

LENAT, D. B., GUHA, R. V., 1990, "CYC: Toward Programs with Common Sense", *Communications of the ACM*, vol 33, n. 8 (Aug), pp 30-49.

LENAT, D. B., GUHA, R. V., 1995b, "CYC: A Large-Scale Investment in Knowledge Infrastructure"; *Communications of the ACM*, vol. 38, n. 11 (Nov), pp 33-38.

LIMA, K. et al 1999, "TeleCardio: Uma Aplicação de Temedicina em Cardiologia". In: Anales de Simposio en Informática y Salud - *28 Jornadas Argentinas de Informática e Investigación Operativa*, pp 22-33, Buenos Aires, Argentina, Set.

LIMA, K. V, 1999, *Uma Aplicação em Telemedicina,* Tese de M, Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

LISSONI, C., STELLUCCI, G., 1997. *Sodália Object-oriented Domain Analysis Guidelines*. Version 1. 3, Mar.

LIU, X. F., 1998, "A quantitative approach for assessing the priorities of software quality requirements", *The Journal of Systems and Software*, n. 42, pp 105-113.

LONCHAMP, J., 1995, "CPCE: Akernel for Building Flexible Collaborative Process-Centered Environments". In: *Proceedings of Software Engineering Environments*, pp. 95-105, Noordwikkerhout, The Netherlands, Apr.

LÓPEZ, M. F., GÓMEZ-PÉREZ, A., PAZOS SIERRA, J. *et al.,* 1999, "Building a Chemical Ontology Using Methontology and the Ontology Design Environment". *IEEE Intelligent Systems & their applications*, v. 14, n. 1 (Jan/Feb), pp. 37-44.

LOWRY, M., VAN BAALEN, J. *et al.,* 1997, "META-AMPHION: Syntesis of Efficient Domain-Specific Programa Sysntesis Systems", *Automted Software Engineering*, v. 4, n. 2(Apr), pp. 199-241.

MADACHY, R. J., 1995, "Knowledge-Based Risk Assessment and Cost Estimation", *Automated Software Engineering*, v. 2, n. 3 (Sep).

MACHADO, L.F.C., ROCHA, A.R., 1999, "Modelo para Definição, Especialização e Instanciação de Processos de Software", ", In: *Anais do Workshop de Teses em Engenharia de Software - XIII Simposio Brasileiro de Engenharia de Software*, pp. 43-47, Florianópolis, Brasil, Out.

MAIDANTCHIK, C., 1999, Modelo de Processo de Gerência de Software para Equipes Geograficamentes Dispersas, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

MASSOLAR, J. L., 1993, *SIM: Um Gerador Semi Automático de Documentos*. Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

MILI, H., MILI, F. E MILI, A., 1995, "Reusing Software: Issues and Research Directions"; *IEEE Transactions on Software Engineering*; vol. 21, n. 6 (Jun), pp 528-562.

MIZOGUCHI, R., 1994, "Knowledge Reuse and Ontology", In: Fuchi, K., Tokoi, T. (eds) *Knowledge Building and Knowledge Sharing*), Ohmsha, ltd e IOS Press pp 165-174.

MOSER, S., NIRERSTRASZ, O., 1996, "The Effect of Object-oriented Frameworks on Developer Productivity", *IEEE Computer* (Set), pp. 45-51.

MOTTA, E. FENSEL, D., GASPARI, M. *et al.* 1999, "Specifications of Knowledge Componentes for Reuse". In: *Proceedings of the 11th Software Engineering an Knowledge Engineering Conference*, pp. 36-43, Kaiserslautern, Alemanha, Jun.

MOURA, L. M. V., ROCHA, A. R. C., 1992, *Ambientes de Desenvolvimento de Software*, Relatório Técnico COPPE/UFRJ ES-271/92, Rio de Janeiro, RJ, Brasil.

MOURA, L. M. V., ROCHA, A. R. C., 1992, *Ambientes de Desenvolvimento de Software*, Publicações Técnicas COPPE/UFRJ, ES-271/92, Rio de Janeiro, Brasil.

MUSEN, M. A., 1998, "Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem Solving Methods". *JAMIA, Journal of the American Medical Informatics Association*, pp. 46-52.

NAKAKOJI, K., REEVES, B. N, AOKI, A., 1996, "eMMa: An Environment for Designing "good" Multimedia Presentations", In: *Proceedings of the First Asia Pacific Conference on Computer Human Interaction*, pp. 397-409, Signapore, Signapore, Jun.

NBR ISO/IEC 12207, 1998, *Tecnologia de Informação – Processos de Ciclo de Vida de Software*, Associação Brasileira de Normas Técnicas, Rio De Janeiro, Brasil.

NECHES, R., FIKES, R., FININ, T, *et al.*, 1991, "Enabling Technology for Knowledge Sharing", *AI Magazine*, (Fall), pp 36-56.

NECTCHES, R., 1994, "Knowledge Sharing in Itegrated User Support Environements: Applications, Framework and Infrastrucutre", In: Fuchi, K., Tokoi, T. (eds)

Knowledge Building and Knowledge Sharing; Ohmsha, ltd e IOS Press; pp 165-174.

NIELBOCK, D., 1997, *Uma Ferramenta para Planejamento e Produção de Desenvolvimento de Software*, Tese de M. Sc., COPPE/UFRJ Rio de Janeiro, RJ, Brasil.

NING, J. Q., 1994, "Developing Domain-Oriented Design - The question is How, not Why"; Automated Software Engineering - The International Journal of Automated Reasoning and Artificial Intelligence in Software Engineering; vol. 1, n. 2 (Jun), pp 215-218.

O'LEARY, D. E., 1998, "Using AI in Knowledge Management: Knowledge Bases and Ontologies". *IEEE Intelligent Systems & their applications*, vol. 13, n. 3 (May/Jun), pp. 34-39.

OLIVEIRA, K. M., 1995, *Avaliação da Qualidade de Sistemas Especialistas*, Tese M. Sc COPPE/UFRJ, Tese de M, Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

OLIVEIRA, K. M., ROCHA, A. R., RABELO Jr., A., 1995, "Verificação e Validação de Sistemas Especialistas", In: Proceedings of *XV Congresso da Sociedade Brasileira de Computação/ XXI Conferencia Latinoamericana de Informática,* pp 351-362, Canela, Brasil; Ago.

OLIVEIRA, K. M., 1996a, "Ambiente Orientado a Domínio para Cardiologia", In: *Anais do X Simpósio Brasileiro de Engenharia de Software/Workshop Pesquisas de Tese em Engenharia de Software*, pp 1-2, São Carlos, Brasil, Out.

OLIVEIRA, K. M., 1996b, "Um Ambiente de Desenvolvimento de Software Orientado ao Domínio", *Monografia de Tópicos Especiais em Engenharia de Software – Análise de Domínio*, COPPE/UFRJ, Rio de Janeiro, Brasil.

OLIVEIRA, K. M., RABELO JR., A., ROCHA, A. R. *et al,* 1996a, "An Experience of Software Quality Assurance for an Expert System in Cardiology", In: *Proceedings of 2nd Medical Engineering Week of the World*, Taipei, Taiwan, May.

OLIVEIRA, K. M., ROCHA, A. R. C. DA, WERNER, C. M. L., *et al,* 1996b, "Um Ambiente de Desenvolvimento de Software Orientado a Domínio para

Cardiologia", In: *Proceedings of LII Congresso Brasileiro de Cardiologia, Arquivos Brasileiros de Cardiologia*, Suplemento I, vol. 67, Salvador, Brasil, Set.

OLIVEIRA, K.M., WERNER, C., ROCHA, A. R, *et al.,* 1996c, "A Domain Oriented Software Environment for Cardiology", *Proceedings of VI World Congress of Cardiac Rehabilitation*; pp 25, Buenos Aires, Argentina; Junho.

OLIVEIRA, K. M., WERNECK, V. M., WERNER, C., 1996d, "Aquisição de Conhecimento: Velha fórmula, Nova Aplicação", In: *Anais do Workshop de Engenharia de Sofware e Sistemas Baseados em Conhecimento*, pp 67-71, Vitória, Brasil, Set.

OLIVEIRA, K. M., MENEZES, C., 1997, *Estudos Iniciais para Definição de uma Ontologia de Cardiologia*, II Exame de Qualificação para Doutoramento, COPPE/UFRJ, Rio de Janeiro, Brasil.

OLIVEIRA, K. M., MATWIN, S., 1998, "KED: Um Editor de Conhecimento para Cardiologia", In: *Anais do Caderno de Ferramentas - XII Simposio Brasileiro de Engenharia de Software*, pp. 21-28, Maringá, Brasil.

OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.,* 1998a, *Towards a Domain-Oriented Software Development Environment for Cardiology*, 5th Doctoral Consortium on Advanced Information Systems – CAISE 98 , Pisa, Italy, Jun.

OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.,* 1998b, *Ambientes de Desenvolvimento de Software Orientado a Domínio*, Publicações Técnicas, COPPE/UFRJ, ES 483/98, Rio de Janeito, RJ, Brasil.

OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.,* 1999a, "Using Domain-Knowledge in Software Development Environments", In: *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering*, pp. 180-187, Kaiserlauther, Alemanha, Jun.

OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.,* 1999b, "O uso da Teoria do Domínio no Processo de Desenvolvimento de Software", In: *Anais da X Conferencia Internacional de Tecnologia de Software*, pp 223-235, Curitiba, Brasil, Mai.

OLIVEIRA, K. M., CERQUEIRA, A., ROCHA, A. R. *et al.* 1999c, "Qual-Cordis: a domain-specific tool for the identification of software quality requirements using fuzzy theory", In: *Proceedings of the 2nd European Measurement Conference FESMA99*, Amsterdam, Holanda, Oct.

OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H., 1999d, "A Domain-Oriented Software Development Environment for Cardiology", In: *Proceedings of America Medical Informatics Association conference – AMIA*, Washington, D.C., Nov. (In press)

OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.*, 1999e, "CORDIS: Assistência Automatizada no Desenvolvimento de Software em Cardiologia", In: *Anales de Simposio en Informática y Salud - 28 Jornadas Argentinas de Informática e Investigación Operativa, pp 34-48*, Buenos Aires, Argentina, Set.

OUSSALAH, M., MESSAADIA, K. 1999, "An All-reuse Methodology for KBS Components Library". . In: *Proceedings of the 11th Software Engineering an Knowledge Engineering Conference*, pp. 187-191, Kaiserslautern, Alemanha, Jun.

PARSONS, J. E WAND, Y., 1997, "Using Objects for System Analysis", *Communications of the ACM*, v. 40, n. 12 (Dec), pp. 104-110

PAULK, M. C., WEBER, C. V., CURTIS, B., CHRISSIS, M. B. (eds), 1995, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Carnegie Mellon University, Software Engineering Institute, Addison-Wesley Longman Inc.

PENEDO, M. H., 1993, *Towards understanding Software Engineering Environment,* Techinical Report IMPSEE-TRW-93-003, Aug.

PFLEEGER, S. L., 1998, *Software Engineering – Theory and Practice*, New- Jersey, Prentice-Hall Inc.

PRESSMAN, R., 1997, Software Engineering – A Practitioner's Approach, Mc-Graw-Hill.

PRIETO-DÍAZ, R., 1987, "Domain Analysis for Reusability"; In: *Proceedings of The Eleventh Annual International Computer Software & Applications Conference*, pp 23-29, Tokio, Japão, Oct.

PRIETO-DÍAZ, R., 1990, "Domain Analysis: an Introduction"; *Software Engineering Notes*; Vol 15, n. 2 (Abr), pp 47-54.

PUNCELLO, P. P., TORRIGIANI, P., PIETRI, F. *Et al.*, 1988, "ASPIS: A Knowledge-Based CASE Environment", *IEEE Software*, v. 5, n. 2 (Mar), pp 53-65.

RABELO JR. A., ROCHA A. R., OLIVEIRA, K. M *et al.*, 1997, "An Expert System for Diagnosis of Acute Myocardial Infarction with ECG Analysis", *Artificial Intelligence in Medicine*, n. 10, pp. 75-92.

RADER, J., BROWN, A. W., 1995, "Computer-Aided Sub-Processes (CASPs): A Pratical Approach to the Use of CASE Technology to Support Process Improvement". In: *Proceedings of 7th International Workshop on Computer Aided Software Engineering*, pp. 20-28, Toronto, Canada, Jul.

RANDALL, R., ETT, W., 1995, "Using Software Process to Integrate Spftware Engineering Environments". In: *Proceedings of Software Technology Conference*, Salt Lake City, USA, Apr.

RECTOR, A., HORROCKS, I., 1997, "Experience Building a Large, Re-usable Medical Ontology using a Dewscription Logic with Transitivity and Concept Inclusions". In: *Proceedings of AAAI-97 Spring Symposium Series, Ontological Engineering*, pp. 100-107, Standford University, USA, Mar.

REPENNING, A. E SUMMER, T., 1995, "Agentsheets: a medium for creating Domain-Oriented Visual Languages", *IEEE Computer* (Mar), pp 18-25.

RIBEIRO, R. A, 1996, "Fuzzy multiple attribute decision making: a review and new preference elicitation techniques", *Fuzzy Sets and Systems*, n. 78, pp 155-181.

ROBBINS, J. E. e REDMILES, D. F., 1996, "Software Architecture Design From the Perspective of Human Cognitive Needs", In: *Proceedings of the California Software Symposium*, pp. 16-27, Los Angeles, California, USA, Apr.

ROBBINS, J. E., HILBERT, D. M., REDMILES, D. F., 1997, "Argo: A Design Environment for Evolving Software Architectures". In: *Proceedings of 19th Intrnational Conference on Software Engineering*, pp. 17-23, Boston, USA, May.

ROBINS, S. L., 1974, *Patologia Estrutural e Funcional*, Brasil, Interamericana.

ROCHA, A. R. 1983, *Um Modelo para Avaliação da Qualidade de Especificações*, Tese de D. Sc, PUC-RJ, Rio de Janeiro, RJ, Brasil.

ROCHA, A. R., MAIDANTCHIK, *C.*, OLIVEIRA *et al.*, 1999, *Experience in Defining, Using and Improving Software Process*, Publicações Técnicas, COPPE/UFRJ, ES 507/99, Rio de Janeito, Brasil.

ROCHA, A. R., WERNER, C. M., TRAVASSOS, G. H. *et al.*, 1996, "Processo de Desenvolvimento de Software Baseado em Reutilização", Publicação Técnica COPPE/UFRJ 1/96, Rio de Janeiro, Brasil.

ROCHA, A. R. C., AGUIAR, T. C., SOUZA, J. M., 1990, "Taba: A Heuristic Workstation for Software development", In: *Proceedings of COMPEURO 90*, Tel Aviv, Israel, May.

ROSSAK, W., KIROVA, V., JOLOLIAN, L. . LAWSON, H., ZEMEL, T. 1997, "A Generic Model for Software Architecture", *IEEE Software*, (Jul/Aug), pp. 84-92.

ZLOT, F., SANTOS, G, 1999. *Definição e Instanciação de Ambientes na Estação TABA*. Projeto Final de Curso, Universidade Federal do Rio de Janeiro, Brasil.

SCHLANT, R. C., ALEXANDER, R. W., 1994, *Hurst's The Heart*, McGraw-Hill. INC., 8ª edição.

SCHREIBER, G., WIELINGA, B., AKKERMANS, H. *et al*, 1994, "CML: The CommonKADS Conceptual Modelling Language", In: *Proceedings of 8th European Knowledge Acquisition Workshop/EKAW94 - Lecture Notes in Artificial Intelligence 867*, L. Steels, G. Screiber e W. Van de Velde (eds.), pp 1-19.

SCHREIBER, G., WIELINGA, B., JANSWEIJER, W., 1995, "The Kactus View on the 'O' Word"; In: *Workshop on Basic Ontological Issues in Knowledge Sharing/IJCAI95,* Montreal, Canadá, Aug.

SELBY, R. W., *et al.*, 1991, "Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development", In: *Proceeedings of the 13th International Conference on Software Engineering*, May.

SELFRIDGE, P. G., 1994, "Commentary on 'Domain-Oriented Design Environments' by Gerhard Fischer", Automated Software Engineering - The International Journal of Automated Reasoning and Artificial Intelligence in Software Engineering; vol. 1, n. 2 (Jun), pp. 219-222.

SHAW, M., GARLAN, D., 1996. *Software Architecture – Perspective on Emerging Discipline*, New Jersey, Prentice-Hall Inc.

SHEPPERD, M., SCHOFIELD, C., 1997, "Estimating Software Project Effort Using Analogies", *IEEE Transactions on Software Engineering*, vol. 23, n. 12 (Nov).

SHIPMAN III, F. M., MCCALL, R., 1994, "Supporting Knowledge base Evolution with Incremental Formalization", In: *Proceedings of Conference in Human in Computing System – CHI'94*, pp. 285-291, Boston, USA, Apr.

SMITH, S., BECKER, M., 1997, "An Ontology for Constructing Scheduling Systems". In: *Proceedings of AAAI-97 Spring Symposium Series, Ontological Engineering*, pp. 120-129b, Standford University, USA, Mar.

SNIFF +, 1999, "Source Code Engineering: A generation Beyon Integrated Developmemnt Environments for Software Development Tools", *Take Five Software Techincal Report*, v 1. 0, TakeFive Software, Inc.

SPS, 1996, "Domain-Oriented Software Analysis and Engineering Environments", *Software Productivity Solutions Tecnical Report*, USA.

SUTCLIFFE, A., MAIDEN, N., 1998, "The Domain Theory for Requirements Engineering", *IEEE Transactions on Software Engineering*, v. 24, n. 3 (Mar), pp. 174-196.

SWARTOUT, W. R., NECTCHES, R., PATIL, R., 1994, "Knowledge Sharing: Prospects and Challenges", In: Fuchi, K., Tokoi, T. (eds), *Knowledge Building and Knowledge Sharing*, Ohmsha, ltd e IOS Press, pp 102-109. .

SWARTOUT, W., TATE, A., 1999, "Ontologies – Guest Editors Introduction". *IEEE Intelligent Systems & their applications*, vol. 14, n. 1 (Jan/Fev), pp. 18-19.

TAYLOR, R. N., TRACZ, W., COGLIANESE, L., 1995, "Software Development Using Domain-Specific Software Architectures", *Software Engineering Notes*, vol. 20, n. 5 (Dec), pp. 27-37.

TERRY, A., HAYES-ROTH, F., ERMAN, L. *et al.*, 1994, "Overview of Teknowledge's Domain-Specific Software Architecture Programa", *Software Engineering Notes*, vol. 10, n. 4 (Oct), pp. 68-76.

TOTH, G. A., 1995, "Automated Method for Identifying and Prioritizing Project Risk Factors", *Automated Software Engineering*, v. 2, n. 3 (Sep).

TRACZ, W. 1994, "Domain-Specific Software Architecture (DSSA) – Frequently Asked Questions (FAQ)", *Software Engineering Notes*, v. 19, n. 2 (Apr), pp. 52-56.

TRACZ, W., COGLIANESE, L., YOUNG, P., 1993, "A Domain-Specific Software Architecture Engineering Process Outline", *Software Engineering Notes*, v. 18, n. 2 (Apr), pp. 40-44.

TRAVASSOS, G. H., 1994, O Modelo de Integração de Ferramentas da Estação TABA, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

UMLS, 1998, *Unified Medical Language System – Knowledge Source*, 9th ed, National Library of Medicine, Washington, D.C., USA.

USCHOLD, M., GRUNINGER, M., 1996, "Ontologies: principles, methods and applications", *The Knowledge Engineering Review,* Vol 11:2, pp 93-136.

USCHOLD, T., 1996, "Building Ontologies: Towards a Unified Methodology"; In: Proceedings of 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems; Cambridge, UK, Dec.

USCHOLD, T., KING, M., 1995, "Towards a Methodology for Building Ontologies", In. *Proceedings of Workshop on Basic Ontological Issues in Knowledge Sharing/IJCAI95,* Montreal, Canada, Aug.

VALENTE, A., 1995, *Legal Knowledge Engineering - A modelling Approach*, Amsterdan, Holanda, IOS Press.

VALENTE, A., RUSS, T., MACGREGOR, R. *et al.,* 1999, "Building and (re)Using an Ontology of Air Campaign Planning", *IEEE Intelligent Systems & their applications*, vol. 14, n. 1 (Jan/Feb), pp. 27-36.

VALÉRIO, V., FENAROLI, M., BENEDICENTI, L. *et al.,* 1999, "Software Development with Domain Analysis: Process Improvement Experiment". . In: *Proceedings of the 11th Software Engineering an Knowledge Engineering Conference*, pp. 76-82, Kaiserslautern, Alemanha, Jun.

VALLE, C., XIMENES, A. A., CAMPOS, G., *et al.* 1997, "Educação de Pacientes através de Sistemas de Acesso Público", Revista Brasileira de Informática na Educação, vol 1, n. 1 (Set).

VAN HEIJST, G. VAN, SCHREIBER, A TH., WIELENGA, B. J., 1997, "Using Explicit Ontologies in KBS Development", *International Journal of Human-Computer Studies*, vol 45, No 2/3; pp 183-292.

VAN HEIJST, G., 1995, *The Role of Ontologies in Knowlege Engineering*, PhD. Thesis, Universidade de Amsterdan, Holanda.

VASCONCELOS, F., 1997, *Reutilização de Processos de Desenvolvimento de Software baseada em Padrões*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

WAND, Y, 1996, "Ontology as a foundation for meta-modelling and method engineering", *Information and Sofware Tecnhology*, No 38, 281-287.

WEISS, DAVID, M., LAI, CHI TAU R., *Software Product-Line Engineering: A Family-Based Software Development Process*, Addison Wesley, 1999.

WERNECK, V. M, 1990, *Taxonomia de Dominios de Aplicação*, Vera Maria Benjamim Werneck, Tese de M, Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

WERNECK, V. M., 1995, *Ambiente para Desenvolvimento de Software Baseado em Conhecimento*, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

WERNECK, V. M., ROCHA, A. R. C. DA, RABELO, A. *et al.*, 1995a, "Ambiente para Desenvolvimento de Sistemas Baseados em Conhecimento"; In: *Proceedings of XV Congresso da Sociedade Brasileira de Computação/XXI Congresso Latinoamericano de Informática,* Canela, Brasil, Jul.

WERNECK, V. M, OLIVEIRA, K. M., ROCHA, A. R. *et al.*, 1997, "A Software Development Processs for Expert Systems", *Proceedings of 10th International Simposim on Methodologies for Intelligent Systems*, pp. 209-220, North Caroline, USA, Oct.

WERNECK, V. M, ROCHA, A. R. C. DA; RABELO, A., 1995b, "O método KADS-estendido", In: *Proceedings of IX Simpósio Brasileiro de Engenharia de Software*, Recife, Brasil, Out.

WERNER, C. M. L., TRAVASSOS, G. H., DA ROCHA, A. C. *et al.,* 1997, "Memphis: A Reuse Based O. O. Software Development Environment", In: *Proceedings of TOOLS*, Beijing, China, Sep.

WIELEMAKER, J., 1998, *SWI-Prolog 3.2.8 - Reference Manual.* (ftp://swi.psy.uva.nl/pub/SWI-Prolog).

WIRFS-BROCK, R. J., JOHNSON, R. E., 1990, "Surveying Current Research in Object-Orient Design", *Communications of the ACM*, v. 33, n. 9 (Sep).

Anexo 1

O Modelo da Estação TABA

Este anexo apresenta o modelo de classes atual da Estação TABA A Figura A1.1, mostra o diagrama de pacotes. Em seguida, é apresentado o modelo de classes para cada um dos pacotes (Figuras A1.2, A1.3, A1.4, A1.5, A1.6). É importante ressaltar que nesse modelo só estão apresentadas as classes atualmente utilizadas na nova versão atualmente implementada e nos trabalhos em andamento. Dessa forma, na categoria de dados, por exemplo, não foi representada nenhuma classe.

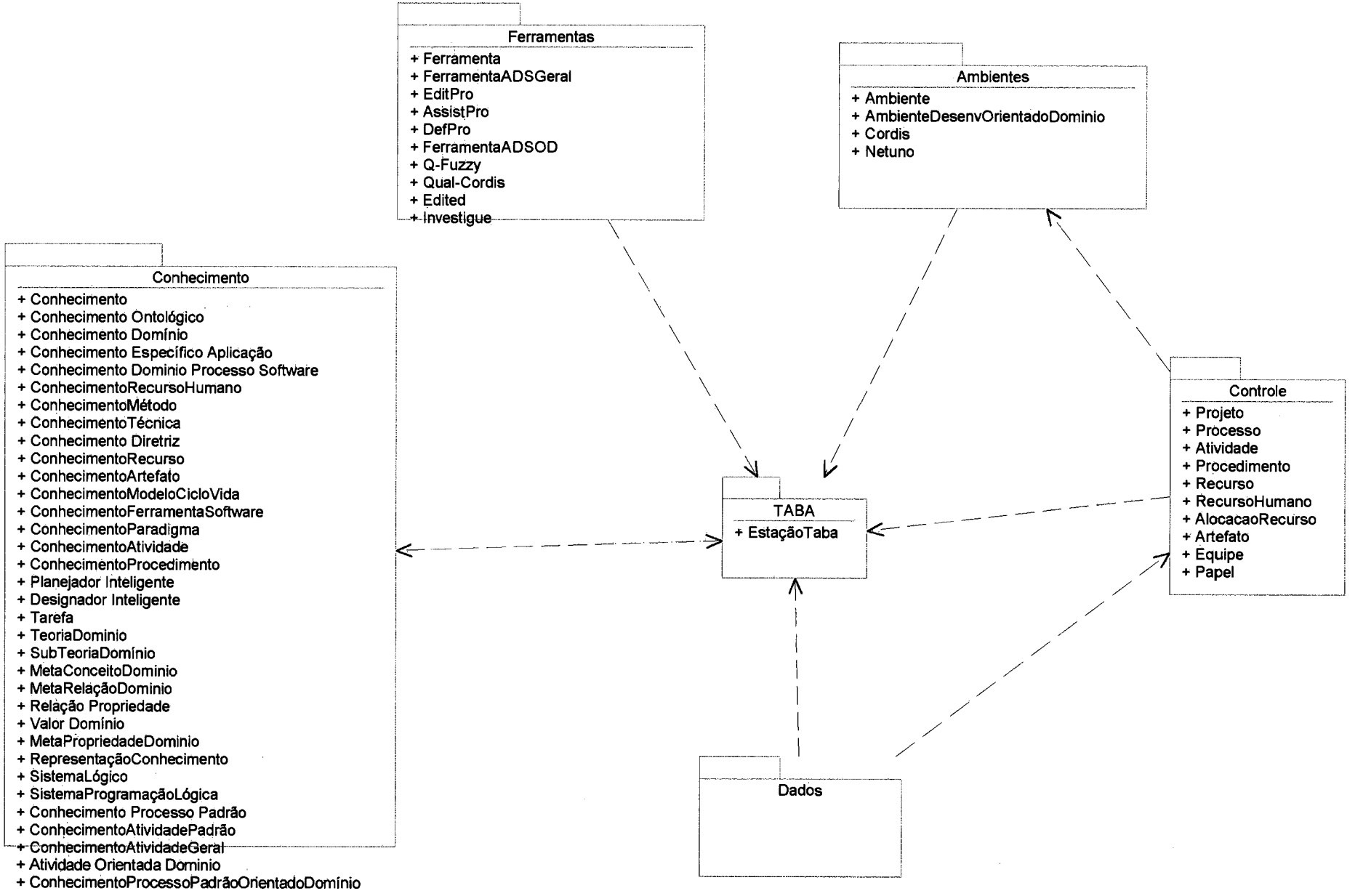

Figura A1.1 - Modelo de Pacotes da Estação TABA

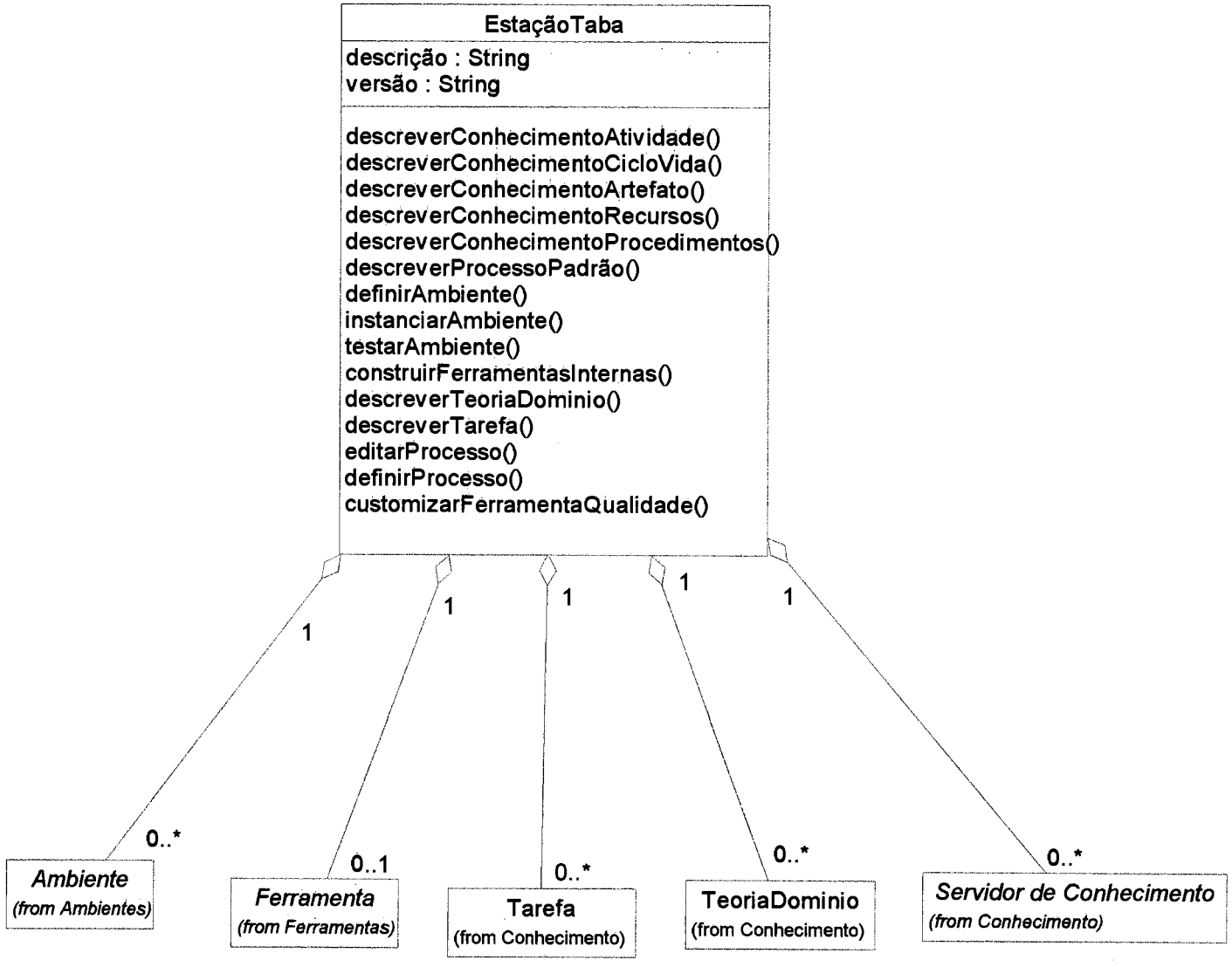

Figura A1.2 - Modelo de Classes do Pacote Estação TABA

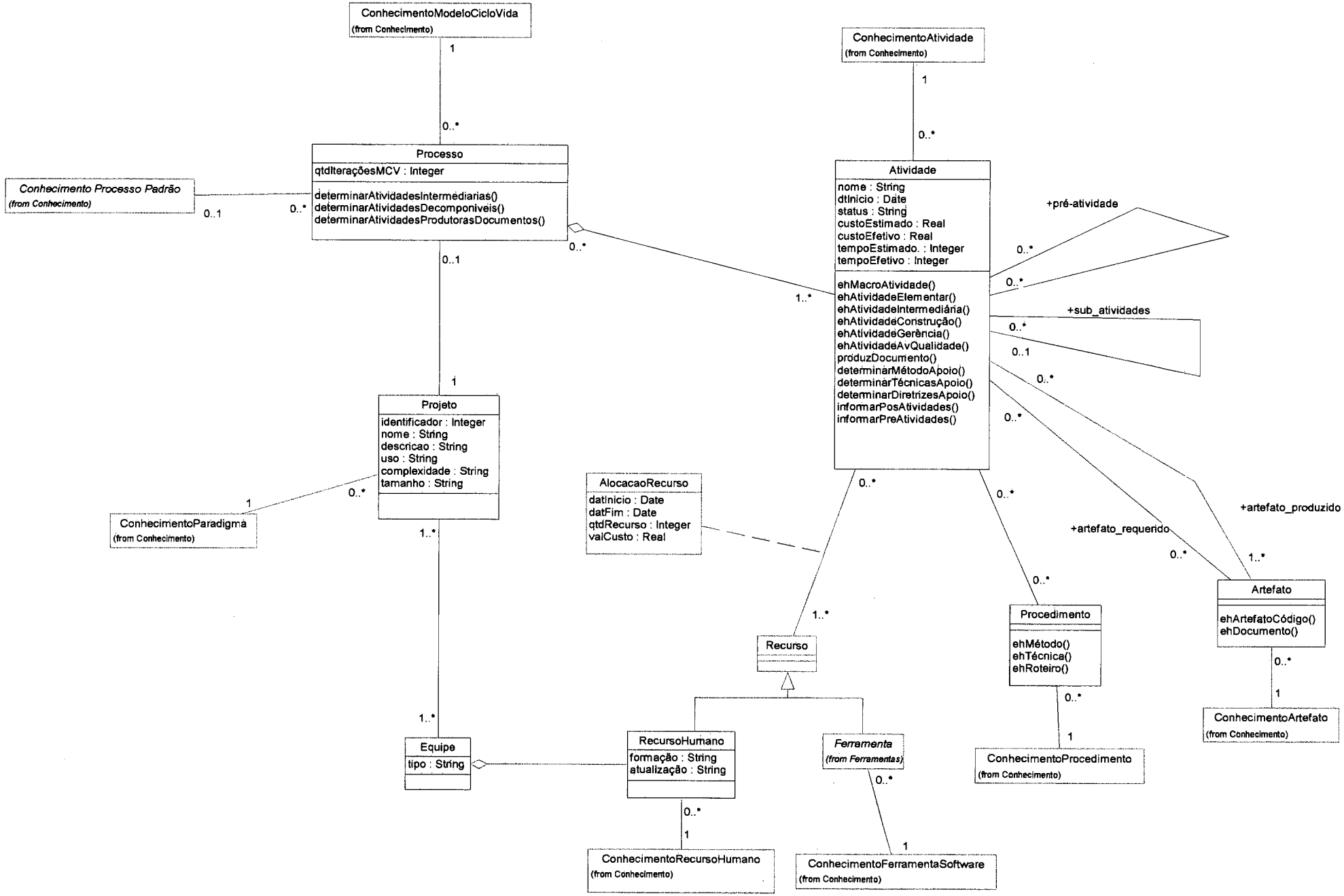

Figura A1.3 - Modelo de Classes do Pacote Controle

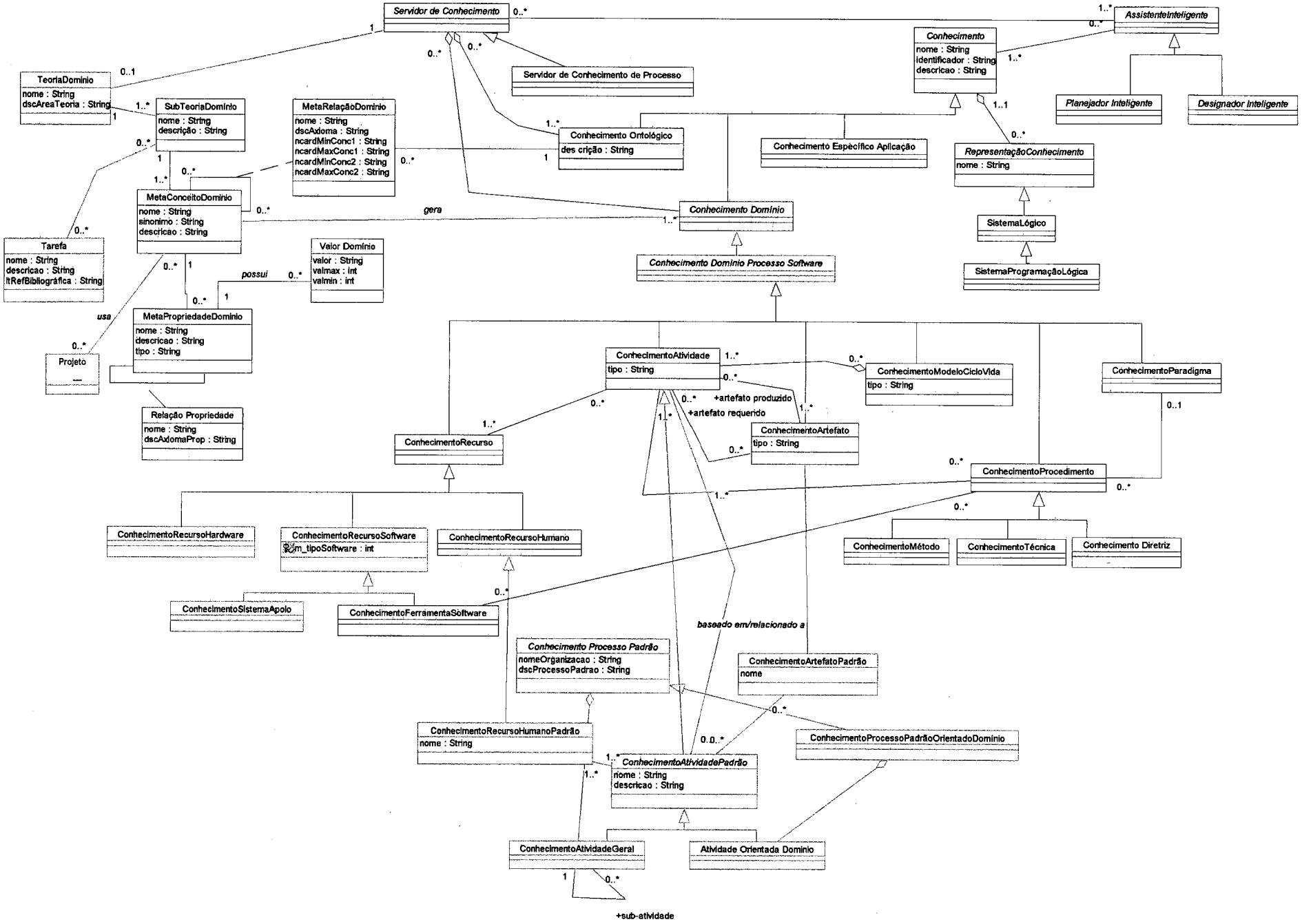

Figura A1.4 - Modelo de Classes do Pacote Conhecimento

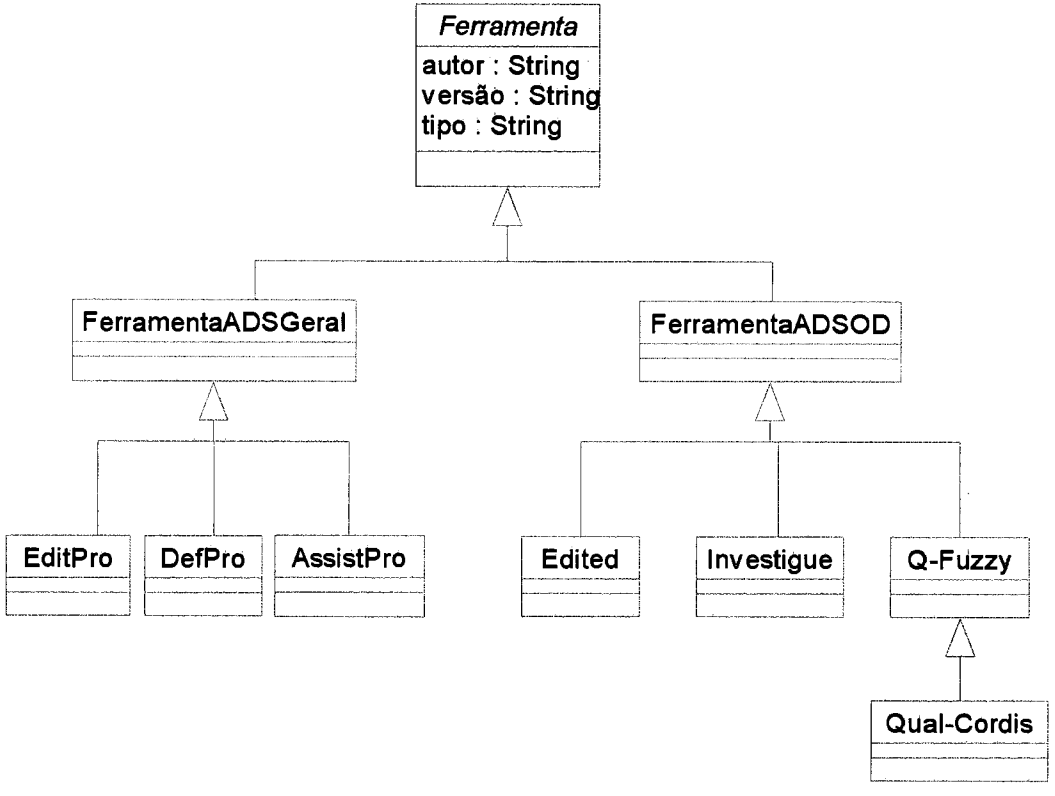

Figura A1.5 - Modelo de Classes do Pacote Ferramentas

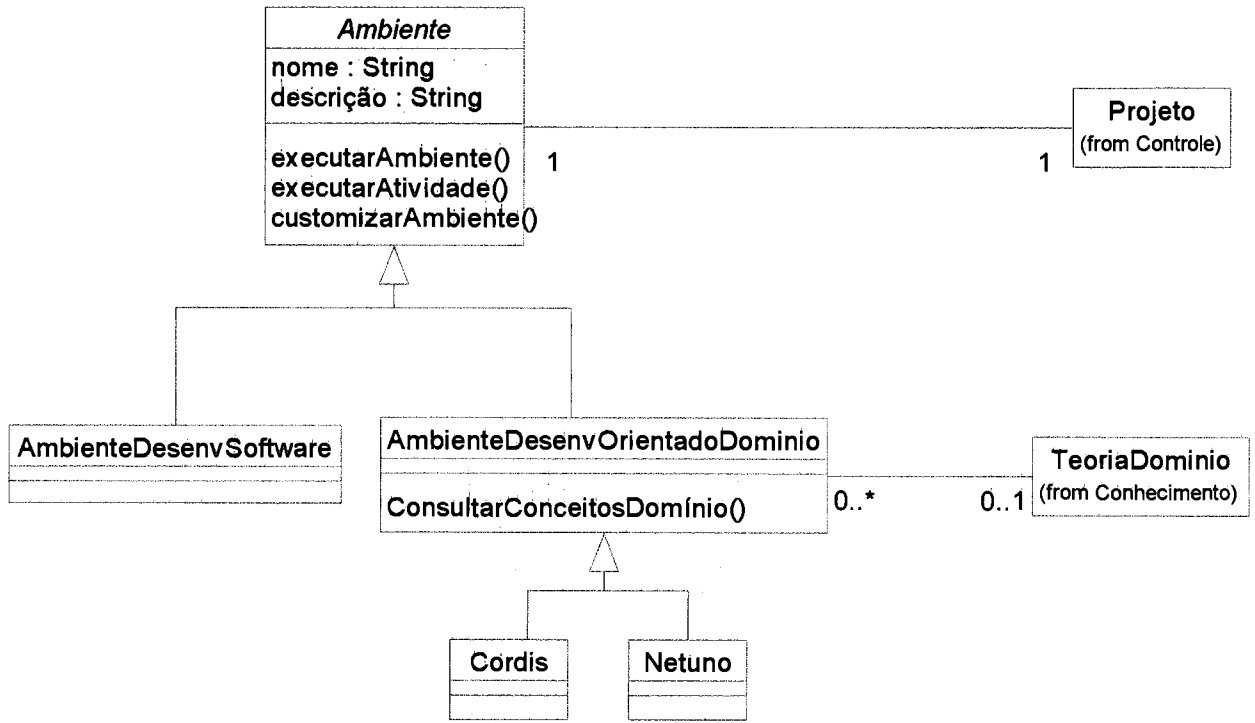

Figura A1.6 - Modelo de Classes do Pacote Ambiente

Anexo 2

Descrição de Conceitos da Teoria do Domínio de Cardiologia

Tabela A2.1 - Sub-Teoria de Patologia

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de instâncias
Patologia	Estudo da natureza das doenças e das alterações associadas da estrutura e função	Processo Inflamatório Malformação congênita Processo Degenerativo Processo Neoplásicos Processo carencial Processo infecicioso		
Processo Inflamatório	Processo decorrente da resposta à lesão do tecido envolvendo reações neurológicas vasculares, humorias e celulares dentro do local lesionado	-	Cardite reumática Pericardite	gravidade
Malformação congênita	Má formação originada no desenvolvimento embrionário	-	Comunicação inter-atrial Tetrologia de Fallot Comunicação inter-ventricular	manifestação
Processo Degenerativo	Processo resultante da perda das características normais do tecido, geralmente decorrente do envelhecimento	-	Aterosclerose	
Processo Neoplásicos	Processo resultante do crescimento de uma massa anormal de tecido com estrutura e função diferentes do tecido normal do qual se originou	-	Mixoma Metastases	origem
Processo carencial	Processo resultante de deficiências nutricionais	-	Beri-Beri (deficiência de vit B1)	

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de instâncias
Processo infecicioso	Processo resultante da ação de agentes infecciosos: virus, bacterias, protozoários, etc.		Doença de Chagas Endocardite bacteriana	microorganismo
Trauma	Processo resultante da ação de agentes mecânicos externos		Ferimento por arma Acidentes automobilísticos	
Etiopatogenia	Estudo de como os fatores interagem para produzir a doença	Etiologia Patogenia		
Etiologia	Causa de alterações no substrato anatômico. Geralmente se refere a fatores externos.	-	Infecção viral Ferimento	
Patogenia	Estudo do processo de evolução da doença		aumento da pressão arterial aumento do colesterol aumento da glicemia	

Tabela A2.2 - Sub-Teoria Anatomia do Coração

Conceito	Descrição	Sub-Classes	Instâncias (exemplos)	Atributos de Instâncias
Substrato anatômico	Estrutura normal do coração e dos vasos sanguíneos	-	Músculo cardíaco Endocárdio Pericárdio Sist. excito condutor Circulação sistêmica Circulação pulmonar	
Componente anatômico	Integrantes do substrato anatômico	Câmara cardíaca Artéria Veia Válvula Septo Sistema excito Condutor Parede		morfologia

Conceito	Descrição	Sub-Classes	Instâncias (exemplos)	Atributos de Instâncias
Câmara cardíaca	Cavidades existentes no coração	-	Átrio direito Átrio esquerdo Ventrículo direito Ventrículo esquerdo	superfície interna dimensão
Artéria	Vaso que conduz o sangue do coração para os tecidos, com exceção da artéria pulmonar que conduz dos tecidos para o coração.	-	Aorta Descendente Anterior Circunflexa Coronária direita Artéria pulmonar	diâmetro
Veia	Vaso que conduz o sangue dos tecidos para o coração, com exceção da veia pulmonar que conduz do coração para os tecidos.	-	Veia cava superior Veia cava inferior Veia pulmonar	diâmetro
Válvula	Estrutura que direciona o fluxo sanguíneo no coração	-	Válvula Tricúspede Válvula pulmonar Válvula mitral Válvula aórtica	circunferência do anel mobilidade
Septo	Estrutura de separação entre as cavidades	-	Septo inter-atrial Septo inter-ventricular	integridade
Sistema excito Condutor	Sistema responsável pela geração e condução do estímulo que conduz o batimento cardíaco	-	Nó sinusal Nó AV Feixe de His	integridade
Parede	Estrutura que delimitam externamente as câmaras cardícas	-	Pericárdio Miocárido Endocárdio	espessura mobilidade
Função	Função do coração em estado normal ou alterado	Fisiologia Fisiopatoloia		descrição da função
Função Normal (Fisiologia)	Estudo do funcionamento normal do coração e das suas estruturas e dos vasos sanguíneos	-	Contração Direcionamento de fluxo transporte nutrição	
Função Anormal (Fisiopatologia)	Estudo do funcionamento anormal do coração e das suas estruturas e dos vasos sanguíneos	-	Diminuição da contratilidade do músculo cardíaco	

Tabela A2.3 - Sub-Teoria de Diagnóstico

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de Instâncias
Diagnóstico	Identificação da patologia a partir da avaliação das evidências	Sindrômico Etiológico		
Sindrômico	Conjunto de evidências (sinais e sintomas) que o paciente apresenta		insuficiência cardíaca angina	
Etiológico	Determinação do fator(es) que causa(m) a doença		miocardite viral doença aterosclerótica	virus bacterias agentes químicos

Tabela A2.4 - Sub-Teoria de Terapia

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de Instâncias
Terapia	Tratamento aplicado em um paciente para uma determinada patologia	Clínica Intervencionista		
Clínica	Tratamento utilizando drogas ou orientações gerais (hábitos de vida, dieta)		Trombolíticos	droga utilizada dose duração de utilização
Intervencionista	Tratamento realizado no laboratório de cateterismo cardíaco ou cirúrgico.		Angioplastia Troca valvar Revascularização miocárdica Valvoplastia	tipo da cirurgia

Tabela A2.5 - Sub-Teoria de Avaliação Clínica e Exames Complementares

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de instâncias
Evidência	Característica manifestadas no paciente devido a presença da patologia	Quadro clínico Resultado de Exames		
Quadro Clínico	Conjunto de características clínicas referidas ou observadas no paciente	Identificação Sintoma Sinais Antecedente		
Identificação	Caraterísticas que identificam o paciente como indivíduo	-		idade sexo peso altura cor massa corpórea ocupação
Sintoma	Queixas ou informações referidas pelo paciente	-	Dispnéia Síncope Palpitação Diarréia	duração caracter intensidade fatores que aliviam o sintoma fatores que precipitam
Dor	Sintoma localizado	-	Dor toráxica Dor medio-esternal Dor dorsal	localização
Sinal	Características investigadas pelo médico através da inspeção, palpação, percussão e ausculta do paciente (exame físico)	Sinal visual Sinal tátil Sinal auscultório Sinal olfativo		
Sinal Visual	Características investigadas pelo médico através da inspeção visual		facies de sofrimento agudo palidez cianose	caracterização

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de instâncias
Sinal Tátil	Característica investigada pelo médico através da palpação	Pulso Abdome Edema Cardíaco		
Pulso	Características investigadas através da palpação dos pulsos		Caracterização do Pulso	frequência de pulso localização tipo do pulso amplitude de pulso
Abdome	Características investigadas através da palpação do abdome		Hepatomegalia Esplenomegalia Ascite	caráter dimensões
Edema	Aumento do volume resultante do acúmulo do líquido entre as células (intersticial).	-	Edema de MMI (membros inferiores)	localização intensidade caráter
Sinal Tátil Cardíaco	Sensação palpatória do batimento cardíaco		Precórido Ictus Frêmito	localização intensidade ciclo cardíaco
Sinal Auscultatório	Característica investigada pelo médico através da ausculta cardíaca ou percussão			caracter do som percebido intensidade do som percebido duração do som percebido
Ruído Habitual	Som normal na ausculta	Ruído Habitual Respiratório Ruído Habitual Cardíaco		
Ruído Habitual Respiratório	Som normal na ausculta respiratória		Som pulmonar	
Ruído Habitual Cardíaco	Som normal na ausculta cardíaca		Bulha 1 (B1) Bulha 2 (B2)	
Ruído Extra Habitual	Som anormal na ausculta		Ruído Extra Habitual Respiratório Ruído Extra Habitual Cardíaco	localização
Ruído Extra Habitual Respiratório	Som anormal na ausculta respiratória		Crepitos Roncos Sibilos	

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de instâncias
Ruído Extra Habitual Cardíaco	Som normal na ausculta cardíaca	Sopro Pressão Arterial	Click mesossitólico Estalido de abertura VM Atrito B3 B4	
Sopro	Som que transmite a sensação de sopro	-		relação com ciclo cardíaco
Pressão arterial		-		localização valor
Sinal Olfativo	Característica investigadas pelo médico pelo olfato	-	Diabete	característica do hálito
Antecedente	Característica pessoal e familiar	-	Antecedente Pessoal Antecedente Familiar	
Antecedente Pessoal	Antecedente referente ao próprio paciente (hábitos de vida)	-	tabagismo hipertensão arterial dislipidemia diabete obesidade stress pós-menopausa evento coronário prévio doenças prévias	fator temporal intensidade do antecedente
Antecedente Familiar	Antecedentes relativos aos parentes do paciente (doenças de caráter hereditário)	-	Diabete Evento coronário Cardiopatia isquêmica	idade em que ocorreu grau de parentesco
Resultado de Exame	Evidência obtida a partir da realização de um exame	-		
Exame Complementar	São exames solicitados pelo médico para ajudar ou refinar o diagnóstico. São utilizados ainda no acompanhamento terapêutico do paciente	Invasivos Não Invasivos		Medicamentos em uso característica

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de instâncias
Invasivo	Exames realizados através da introdução de algum instrumento no corpo do paciente para investigação	-	Cateterismo Cardíaco Estudo eletrofisiológico invasivo	dimensão das câmaras perviadade integridade contratilidade da parede
Não invasivo	Exames realizados sem a introdução de algum instrumento no corpo do paciente para investigação	Exame Laboratorial Baseado em Ultrasom Baseado em Sinal Baseado em Radiação		
Exame Laboratorial	Exames realizados a partir da análise do sangue urina , fezes ou fragmentos de tecidos do paciente	Dosagem de bioquímicos Avaliação hematológica Uroanálise Bacteriologia		substância avaliada
Bioquímica	Determinação da concentração de determinadas substâncias no sangue	-	Colesterol Eletrólitos Glicemia	taxa da substância
Hematologia	Determinação do número e forma dos elementos celulares integrantes do sangue	-	Eritrograma Leucograma	número de células forma das células
Sorologia		-		
Uroanálise	Determinação dos elementos presentes na urina	-	Características da urina	Concentração da substância elementos figurados
Bacteriologia	Pesquisa de agentes infecciosos	-	Presença de infecção	Microorganismo detectado
Baseado em ultrasom	Exames não invasivos baseados na emissão de ultrasom	-	Ecocardiograma transeofágico Ecocardiograma transtorácico Ecocardiograma de stress Ecocardiograma para cardiopatia congênita	estrutura avaliada forma da estrutura função da estrutura dimensão da estrutura
Baseado em Sinal elétrico	Exames não invasivos baseados na captura dos sinais eletrofisiológicos do coração	-	Eletrocardiograma Teste Ergométrico Holter	ritmo alterações no ritmo

Conceito	Descrição	Sub-Classes	Instâncias (exemplo)	Atributos de instâncias
Baseado em Radiação	Exames não invasivos baseados na emissão de radiação ionizante (raios-x ou raios gama)	-	Medicina Nuclear Raio-X	perfusão miocárdica
Tilt Test	Teste para avaliar a função autonômica do coração e da pressão arterial.	-		
Mapa	Registro da pressão arterial durante tempo prolongado (por exemplo 24 h)	-		valor da pressão arterial

Tabela A2.5 - Tabela de Atributos de Instâncias

Conceito	Atributo	Descrição	Tipo	Faixa de valores	Unidade
Processo Inflamatório	Gravidade	Caracterização do processo inflamatório	caracter	–agudo –crônico	
Malformação congênita	Manifestação	Tipo da manifestação da mal formação congênita	caracter	–cianótica –acianótica	
Processo Neoplásicos	Origem		caracter	–primário –secundário	
Processo infecicioso	Microorganismo	Tipo de microorganismo responsável pela formação do processo infeccioso	caracter	–viral –bacteriano –fungico	
Componente anatômica	Morfologia	Descrição do aspecto morfológico	caracter		
Câmara cardíaca	Superfície interna	Descrição do aspecto da superfície interna da câmara cardíaca	caracter		
Câmara cardíaca	Dimensão	Tamanho da câmara (geralmente medido pelo diâmetro interno)	numérico		cm/ mm
Artéria	Diâmetro	Medida do diâmetro interno das artérias	numérico		cm/ mm
Veia	Diâmetro	Medida do diâmetro interno das veias			
Válvula	Circunferência do anel	Medida do diâmetro interno das válvulas	numérico		cm/ mm
Válvula	Mobilidade	Descrição da mobilidade da válvula, como por exemplo: se move bem, pouco, etc.	caracter		
Septo	Integridade	Definição da integridade (estar isento de anormalidades) do septo	Boleano	–S –N	
Sistema excito Condutor	Integridade	Definição da integridade (estar isento de anormalidades) do sistema êxito condutor			
Sistema excito Condutor	Espessura	Medida da espessura das paredes	numérico		

Conceito	Atributo	Descrição	Tipo	Faixa de valores	Unidade
Parede	Mobilidade	Descrição da mobilidade da parede.			
Clínica	Droga utilizada	Medicamento	textual		
Clínica	Dose	Quantidade / período de tempo	numérico		num/ dias
Clínica	Duração de utilização	Período de tempo da utilização	numérico		dias/ meses
Intervencionista	Tipo da cirurgia	Tipo da cirurgia	textual	-a céu aberto -no laboratório de cateterismo	
Identificação	Idade	Idade do paciente	numérico	0 - 150	
Identificação	Sexo	Sexo do paciente	caracter	-F -M	
Identificação	Peso	Peso do paciente	numérico		kg
Identificação	Altura	Altura do paciente	real		
Identificação	Cor	Critério para classificação racial	textual	-branca -morena -negra	
Identificação	Massa corpórea	Derivada do peso e altura (peso/altura^2)	textual		
Identificação	Ocupação	Ocupação atual (importância referente ao tipo de material que o paciente trabalha ou a própria característica do trabalho)			
Sintoma	Duração	Duração dos sintomas relatados (delta tempo)	numérico		dias, horas /min
Sintoma	Caracter	característica do sintoma	textual	p/ dor: -intensidade -aperto -opressão -peso -queimor ..	

Conceito	Atributo	Descrição	Tipo	Faixa de valores	Unidade
Sintoma	Intensidade	Graduação do sintoma	textual	–severa –moderada –leve	
Sintoma	Fatores que aliviam o sintoma	Fatores que quando adotados aliviam a queixa relatada	textual	–uso medicamento repouso –...	
Sintoma	Fatores que precipitam o sintoma	Fatores que quando adotados pioram a queixa relatada.	textual	–emoções –esforços –...	
Dor	Localização	Local da dor referida	textual	- torax - dorso - mandíbula ...	
Sinal Visual	caracterização	Descrição textual do aspecto (sinal) identificado no paciente	caracter		
Pulso	Freqüência de pulso	Número de batimentos cardíacos por minuto	numérico		bpm
Pulso	Tipo do pulso	Como se caracteriza o pulso	caracter	–arrítimico –filiforme –biferens –...	
Pulso	Amplitude de pulso	Intensidade percebida	caracter	–amplo –filiforme –normal – ...	
Pulso	Localização	Local que é identificado o pulso	caracter	–carotídeo –radial –cubital –femural	
Abdome	Caracter	Caracter percebido quando realizado o toque no abdome	caracter	–doloroso –não doloroso	
Abdome	Dimensões	Dimensões percebidas quando realizado o toque no abdome	numérico		

Conceito	Atributo	Descrição	Tipo	Faixa de valores	Uni-dade
Edema	Localização	Local do edema	caracter	Membros inferiores Membros superiores	
Edema	Intensidade	Intensidade	caracter		
Edema	Caracter		caracter	-mole -duro -fino -quente	
Sinal Tátil Cardíaco	Localização	Local de observação do sinal tátil	caracter		
Sinal Tátil Cardíaco	Intensidade	Vibração percebida			
Sinal Tátil Cardíaco	Ciclo cardíaco	Relação com o ciclo cardíaco	caracter	-sistólico -diastólico	
Sinal Auscultatório	Caracter do som percebido	Característica do som percebido na ausculta ou percussão	caracter	-grave -agudo	
Sinal Auscultatório	Intensidade do som percebido	Intensidade do som percebido na ausculta ou percussão	Caracter	- forte -	
Sinal Auscultatório	Duração do som percebido	Duração do som percebido na ausculta	numérico		min ou seg
Ruído Extra Habitual	Localização	Local onde foi detectado o ruído extra habitual	caracter		
Sopro	Relação com o ciclo cardíaco	Relação com o ciclo cardíaco	caracter	-sistólico -diastólico	
Pressão arterial	Localização	Localização onde foi identificado a pressão arterial	caracter		
Pressão arterial	Valor	Valor determinado para a pressão arterial	caracter		
Sinal Olfativo	Caracter de hálito	Hálito percebido pelo médico	caracter	-cetônico -alcoólico -...	
Antecedente Pessoal	fator temporal	Quanto tempo tem que o paciente teve ou tem o antecedente (delta tempo)	numérico		anos / mês

Conceito	Atributo	Descrição	Tipo	Faixa de valores	Unidade
Antecedente Pessoal	Intensidade do antecedente	Indicação da seriedade do antecedente no paciente	caracter		
Antecedente Familiar	Idade em que ocorreu	Idade que o familiar tinha quando ocorreu o antecedente	numérico	0-150	
Antecedente Familiar	Grau de parentesco	Parentesco do familiar	caracter	–1° –2° –3°	
Exame	Medicamentos em uso	Drogas que o paciente vem utilizando	caracter	–alfabloqueador –diurético –digital –nitrato –...	
Exame	Característica do exame	Característica geral do exame	caracter	–bom –regular –ruim	
Invasivo	Dimensões das câmaras	Diâmetro e volume	caracter		
Invasivo	Perviedade	Diz respeito a preservação do fluxo (se tem obstrução ou não)	caracter		
Invasivo	Integridade da estrutura	Ausência ou presença das falhas na estrutura	boleano	–presente –ausente	
Invasivo	Contratilidade da parede	Definição de como se apresenta a contração da parede	caracter	–bem –pouco	
Exame Laboratorial	Substância avaliada	Substâncias químicas pesquisadas	caracter		
Bioquímica	Taxa da substância	Valor da concentração da substância	numérico		
Hematologia	Número de células	quantidade de células por mm^3	numérico		num
Hematologia	Forma das células	Aspecto morfológico	caracter	–normais –esféricas –em forma de foice –...	
Uroanálise	Concentração da substância	Concentração da substância química encontrada	caracter		

Conceito	Atributo	Descrição	Tipo	Faixa de valores	Unidade
Uroanálise	Elementos figurados	Presença de células ou outros elementos	caracter		
Bacteriologia	Microorganismo detectado	Tipo de mecanismo detectado na realização de exames	caracter		
Baseado em ultrasom	Estrutura avaliada	Componente anatômico	caracter		
Baseado em ultrasom	Forma da estrutura	Aspecto morfológico	caracter		
Baseado em ultrasom	Função da estrutura	Descrição de como está o funcionamento da estrutura	caracter		
Baseado em ultrasom	Dimensão da estrutura	Descrição de como estão as dimensões da estrutura	caracter		
Baseado em Sinal elétrico	Ritmo	Características do ritmo cardíaco	caracter	–sinusal –atrial –ventricular –...	
Baseado em Sinal elétrico	Alterações no ritmo	Alterações no estado cardíaco norma;	caracter		
Baseado em Radiação	Perfusão miocárdica	Irrigação do músculo cardíaco (distribuição)	caracter	–normal –hipoperfusão de área	
Mapa	Valor da pressão arterial		numérico		
Mapa	Tempo	Tempo em que o paciente foi submetido o exame	numérico		

Anexo 3

Características de Qualidade Consideradas no QUAL-CORDIS

Este anexo contém o conjunto de características de qualidade que foram consideradas na ferramenta Qual-CORDIS, a customização de Q-*fuzzy* para o ADSOD CORDIS.

Para se chegar a este conjunto, foram considerados os seguintes grupos de características:

- **Características de Qualidade presentes na ISO 9126** (NBR 13596) (Tabela A3.1). Para efeito de padronização com o método Rocha (ROCHA, 1983), adotado na construção da ferramenta Q-*fuzzy* chamamos as características da ISO de fatores e as sub-características de sub-fatores. Além disso, organizamos as características da ISO, segundo os objetivos de qualidade definidos por Rocha.

- **Características Gerais de Qualidade** (Tabela A3.2). Neste grupo estão características comuns a qualquer software, que não estão presentes na ISO, mas que foram definidas a partir dos diversos trabalhos analisados.

- **Características de Qualidade de Sistemas Especialistas** (Tabela A3.3), definidas em (OLIVEIRA, 1995)

- **Características de Qualidade de Sistemas Hipermídia** (Tabela A3.4), definidas em (CAMPOS, 1994a).

- **Características de Qualidade de Software Educacional** (Tabela A3.5), definidas em (CAMPOS, 1994b).

- **Características de Qualidade de Sistemas de Informação Hospitalar** (Tabela A3.6), definidas em (CARVALHO, 1997).

- **Características de Qualidade do Prontuário Eletrônico** (Tabela A3.7), definidas em (CARVALHO, 1997) e (GRISOLIA *et al.*, 1999).

- **Características de Qualidade de Sistemas de Acesso Público para Educação de Pacientes** (Tabela A3.8), definidas em (VALLE *et al.*, 1997).

- **Características de Qualidade de Aplicações de Telemedicina** (Tabela A3.9), definidas em (LIMA, 1999).

Para se chegar ao conjunto final de fatores e sub-fatores de qualidade, organizado segundo o Método Rocha, foram selecionadas as características de qualidade pertinentes. Após a seleção, as características foram organizadas e compatibilizadas buscando-se uniformidade de terminologia.

Nas tabelas, a seguir, a coluna Q se refere a se a característica é de qualidade Interna (I) ou Externa (E). Os atributos de qualidade externa são aqueles percebidos pelos usuários finais sendo, portanto, avaliados por esse grupo. Os atributos de qualidade interna, por sua vez, são àqueles percebidos pelos desenvolvedores de software e, portanto, estes são os responsáveis pela avaliação. Essa classificação foi definida para identificar que atributos devem ser avaliados por quais avaliadores no ADSOD instanciado na automatização dos levantamentos de requisitos realizada pela ferramenta Qual-Cordis.

Tabela A3.1 - Características de Qualidade Presentes na ISO 9126 (organizadas segundo o Modelo Rocha)

Características ISO – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Eficiência	Comportamento em relação ao tempo	Atributos do software que evidenciam seu tempo de resposta, tempo de processamento e velocidade na execução de suas funções	I
	Comportamento em relação aos recursos	Atributos do software que evidenciam a quantidade de recursos usados e a duração de seu uso na execução de suas funções	I
Utilizabilidade	**Inteligibilidade**	Atributos do software que evidenciam o esforço do usuário para reconhecer o conceito lógico e sua aplicabilidade	E
	Apreensibilidade	Atributos do software que evidenciam o esforço do usuário para aprender sua aplicação (por exemplo: controle de operação, entradas, saídas)	E
	Operacionalidade	Atributos do software que evidenciam o esforço do usuário para sua operação e controle da sua operação	E
Manutenibilidade	Analisabilidade	Atributos do software que evidenciam o esforço necessário para diagnosticar deficiências ou causas de falhas, ou para identificar partes a serem modificadas	I
	Modificabilidade	Atributos do software que evidenciam o esforço necessário para modificá-lo, remover seus defeitos ou adaptá-lo a mudanças ambientais	I
	Estabilidade	Atributos do software que evidenciam o risco de efeitos inesperados ocasionados por modificações	I
	Testabilidade	Atributos do software que evidenciam o esforço necessário para validar o software modificado	I
Portatilidade	Adaptabilidade	Atributos de software que evidenciam sua capacidade de ser adaptado a diferentes ambientes especificados, sem a necessidade de aplicação de outras ações ou meios além daqueles fornecidos para essa finalidade pelo software considerado	I
	Capacidade para ser instalado	Atributos de software que evidenciam o esforço necessário para sua instalação num ambiente especificado	I
	Conformidade	Atributos do software que o tornam consonante com padrões relacionados à portatilidade	I
	Capacidade para Substituir	Atributos do software que evidenciam sua capacidade e esforço necessário para substituir um outro software, no ambiente estabelecido para esse outro software.	I

Características ISO – Confiabilidade Conceitual			
Fator	**Sub-Fator**	**Descrição**	**Q**
Funcionalidade	Adequação	Atributos do software que evidenciam a presença de um conjunto de funções e sua adequação para as tarefas especificadas	E
	Acurácia	Atributos do software que evidenciam a geração de resultados ou efeitos corretos ou conforme esperados	E
	Interoperabilidade	Atributos do produto de software que evidenciam sua capacidade de interagir com sistemas especificados	I
	Conformidade	Atributos do software que fazem com que ele esteja de acordo com as normas, convenções ou regulamentações previstas em leis e descrições similares, relacionadas à aplicação	E
	Segurança	Atributos do software que evidenciam sua capacidade de evitar acesso não autorizado, acidental ou deliberado, a programas e dados.	I
Confiabilidade	Maturidade	Atributos do software que evidenciam a frequência de falhas por defeitos de software.	I
	Tolerância a falhas	Atributos do software que evidenciam sua capacidade em manter um nível de desempenho especificado no caso das falhas no software ou violação nas interfaces especificadas	I
	Recuperabilidade	Atributos do software que evidenciam sua capacidade de restabelecer seu nível de desempenho e recuperar os dados diretamente afetados, em caso de falha, e o tempo e esforço necessários para tal	I

Tabela A3.2 - Características Gerais de Qualidade

Características Gerais - Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Facilidade de Utilização - conjunto de atributos referentes à facilidade de utilização do sistema pelos usuários finais	Interatividade	refere-se à característica do sistema possuir uma interface interativa, onde o controle esteja, na maioria dos casos, com o usuário mantendo um diálogo conciso e claro.	E
	Disponibilidade de auxílios	característica do sistema possuir informação de ajuda (*help*) disponível para seus usuários.	E
Reutilizabilidade - refere-se a característica do sistema, ou parte dele, em qualquer de suas formas, possa ser reutilizado por outros sistemas.	Modularidade	característica do sistema ser projetado e implementado através de uma estrutura de módulos independentes e particionados logicamente.	I
	Adaptabilidade	característica do sistema ser fácil de adaptar para permitir sua reutilização por outro sistema.	I
Implementabilidade - refere-se às características que tornam viável a implementação do sistema.	Viabilidade Econômica	viabilidade do sistema poder ser construído com uma relação custo/benefício aceita pelos usuários e desenvolvedores.	E
	Viabilidade Financeira	viabilidade do sistema poder ser construído mediante a disponibilidade do capital necessário para o seu desenvolvimento.	E
	Viabilidade tecnológica	viabilidade do sistema poder ser construído considerando-se a existência e a disponibilidade da tecnologia necessária para o seu desenvolvimento.	I
	Viabilidade de mão de obra	viabilidade do sistema poder ser construído, considerando-se a existência e a disponibilidade da mão de obra necessária para o seu desenvolvimento.	I
	Viabilidade de cronograma	viabilidade do sistema poder ser construído dentro do limite de tempo planejado, considerando possíveis ocorrências de imprevistos e com flexibilidade para introdução de atividades não projetadas e/ou contingenciais, mantendo a qualidade definida para o produto.	I
	Viabilidade social	viabilidade do sistema poder ser construído considerando-se o grau de satisfação de seus futuros usuários, bem como os impactos gerados sobre o sistema social ao qual servirá.	E

Características Gerais - Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**O**
Rentabilidade - refere-se aos benefícios financeiros e sociais obtidos com a utilização do sistema	Lucratividade	característica do sistema ter como conseqüência do seu uso um aumento na produtividade dos serviços prestados e na utilização dos recursos da instituição (produtos, serviços e insumos).	E
	Competitividade	característica do sistema oferecer benefícios para os pacientes e para a sociedade em geral, como conseqüência do seu uso.	E
Características Gerais – Confiabilidade Conceitual			
Fator	**Sub-Fator**	**Descrição**	**Q**
Integridade – refere-se às características que tornam o sistema capaz de enfrentar situações hostis.	Robustez	Característica do sistema ser capaz de enfrentar situações hostis, reagindo a elas sem perda de controle.	I
Fidedignidade - conjunto de atributos referentes às características que fazem o sistema corresponder aos requisitos especificados para seus objetivos.	Completitude	Característica do sistema e da sua documentação ser completo em relação a funcionalidade para qual foi projetado, possuindo apenas informações úteis e necessárias para atender os objetivos dos seus usuários finais.	E
	Consistência	Característica do sistema e da sua documentação ser consistente nas diversas partes que os compõe não possuindo informações contraditórias ou mesma informação com significados diferentes na várias partes que o compõe.	E

Características Gerais – Confiabilidade da Representação			
Fator	**Sub-Fator**	**Descrição**	**Q**
Legibilidade - refere-se às características do sistema que o tornam de fácil compreensão.	Clareza	característica do sistema estar especificado, modelado e implementado da forma mais clara possível, isento de práticas que o tornem complexo e de difícil entendimento.	A
	Concisão	característica do sistema ter suas funções implementadas com a quantidade mínima de código e de ter sido especificado e modelado de modo não redundante.	A
	Estilo	refere-se à necessidade da especificação, modelagem e programas do sistema conterem elementos adequados de estilo, de forma que expressem seus objetivos e especificações de maneira simples, elegante, organizada e direta, considerando as padronizações e/ou recomendações estabelecidas para o produto.	I
	Correção da representação	característica do sistema estar correto do ponto de vista do uso da linguagem de especificação adotada, no que se refere a notação, semântica, sintaxe e formato de documentação.	I
	Uniformidade de terminologia	característica do sistema estar documentado com uniformidade de notação e padronização de termos técnicos.	A
	Uniformidade no grau de abstração	característica da documentação do sistema possuir um nível uniforme de detalhe, considerando-se um determinado estágio do desenvolvimento.	I
Manipulabilidade - refere-se às características que tornam possível, ao se analisar o sistema, encontrar, com facilidade, as informações desejadas.	Disponibilidade da documentação	característica do sistema ter sua documentação atualizada e pronta para uso quando necessário.	I
	Estrutura	característica do sistema ter sua documentação e código organizados, segundo uma estrutura hierárquica, que facilite lidar com sua complexidade.	I
	Rastreabilidade	característica do sistema possuir uma documentação e código que permitam a busca de informações através da sequência de agregação de detalhes de um determinado aspecto, desde sua visão mais geral até a mais detalhada, e vice-versa.	I

Tabela A3.3 - Características de Qualidade de Sistemas Especialistas

Sistemas Especialistas – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**O**
Manutenibilidade (sub-fatores a serem acrescentados a característica manutenabilidade da ISSO/IEC 9126)	Evolutibilidade	característica que o sistema deve ter de forma a permitir sua própria evolução, através de refinamentos sucessivos que representem o conhecimento de forma cada vez mais completa.	I
Adequação tecnológica - está relacionado às características e à adequação da tecnologia utilizada para construção do sistema.	Inferência	característica do sistema ter um mecanismo de raciocínio sofisticado.	I
	Grau de explicação	característica do sistema prover uma justificativa da solução gerada.	I
	Suporte	característica do sistema ser provido de técnicas auxiliares de suporte como, tratamento de incerteza, verificação da consistência e de concorrência e compilação de regras.	I
	Adequação da metodologia	característica do sistema usar procedimentos e métodos adequados para a representação do conhecimento.	I
	Adequação do ambiente de programação	característica do sistema ter sido implementado utilizando um ambiente de programação adequado às suas necessidades e que seja capaz de suportar futuras evoluções no sistema.	I
Rentabilidade (sub-fatores a serem acrescentados ao fator rentabilidade das características gerais)	Benefício social	Característica do sistema oferecer retorno social com sua utilização e/ou comercialização.	E
	Benefício no trabalho do usuário	Característica da utilização do sistema oferecer uma simplificação e melhoria nas condições de trabalho de seus usuários.	E
Implementabilidade	Codificabilidade	característica do sistema utilizar técnicas de representação do conhecimento, para	I

Sistemas Especialistas – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
(sub-fatores a serem acrescentados ao fator implementabilidade das características gerais)		codificação e atualização da base de conhecimentos, poderosas e flexíveis.	
Aplicabilidade - refere-se às características do sistema que o fazem adequado e útil no contexto em que está inserido.	Relevância	Característica da utilização do sistema ser importante para a realização da tarefa envolvida	E
	Utilidade	Característica da utilização do sistema ter consequências significativamente úteis.	E
	Justificabilidade	característica do sistema ter sua construção justificada pelo contexto em que estará inserido.	E
	Possibilidade	característica de ser possível a construção do sistema, considerando-se a existência de especialistas e as tarefas envolvidas	E
	Adequação	característica de ser adequada a construção do sistema, considerando-se a natureza, a complexidade e o escopo do problema.	E
Sistemas Especialistas – Confiabilidade Conceitual			
Fator	**Sub-Fator**	**Descrição**	**Q**
Fidedignidade (sub-fatores a serem acrescentados ao fator fidedignidade das características gerais)	Exaustividade	característica do sistema ter uma base de conhecimento com um nível de riqueza que permita, sempre, a geração de resultados confiáveis.	E
	Equivalência a especialista	característica do sistema ter um desempenho com confiabilidade equivalente a de um especialista, tanto em termos de escopo do problema como da solução gerada.	E

Tabela A3.4 - Características de Qualidade de Sistemas Hipermídia

Sistemas Hipermídia – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Manutenibilidade (sub-fatores a serem acrescentados ao fator manutenabilidade das características ISO)	Evolutibilidade	característica do sistema de oferecer facilidade evoluções do hiperdocumento	I
Facilidade de comunicação é a característica de um sistema de hipermídia ser oportuno e ameno ao uso, facilitando a comunicação com o usuário autor, durante todo o tempo em que este o utilizar.	Disponibilidade de Recursos	característica do sistema de possuir recursos para facilitar edição de informações	I
	Oportunidade	característica do sistema de possuir mecanismos para resultados e navegação eficiente em tempo hábil	I
	Estrutura Trabalho Cooperativo	característica do sistema de prover recursos para o trabalho cooperativo	I
	Amenidade de Uso	característica do sistema de oferecer mecanismos para navegação satisfatória	E
Reutilizabilidade (sub-fatores a serem acrescentados ao fator reutilizabilidade das características gerais)	Suporte de Base de Componentes	característica do sistema de possuir base de componentes candidatos á reutilização	I
Eficiência (sub-fatores a serem acrescentados ao fator eficiência das características ISO)	Eficiência de Troca Autor/Leitor	característica do sistema de executar a troca de modos com eficiência	E
	Eficiência de Integração com Outros Software	característica do sistema de integrar-se com outras facilidades do ambiente computacional	I
	Facilidades Oferecidas	característica do sistema de possuir outras facilidades para a autoria	I

Sistemas Hipermídia – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Rentabilidade (sub-fatores a serem acrescentados ao fator rentabilidade das características gerais)	Vantagens Oferecidas pelo Vendedor	Característica do sistema de possuir benefícios oferecidos pelo vendedor	E
Adequabilidade é a característica do hiperdocumento ser adequado ao ambiente proposto para o uso e se integrar com outros recursos utilizados, segundo a perspectiva do usuário final	Adaptabilidade ao Nível do Usuário	característica do sistema de permitir a livre navegação de acordo com necessidades e interesse do leitor	E
	Ausência de Erros na Navegação	característica do sistema de operar sem interrupção de suas funções	E
	Capacidade de Armazenamento das Interações	característica do sistema de assinalar nós visitados e trilhas percorridas	I

Tabela A3.5 - Características de Qualidade de Software Educacional

Software Educacional – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Eficiência (sub-fatores a serem acrescentados ao fator eficiência das características ISO)	Eficiência do desenvolvimento	característica que refere-se às ramificações alternativas para atendimento das necessidades pedagógicas do aluno /professor/ usuário.	**I**
Adequabilidade é a característica do sistema ser adequado ao ambiente educacional e se integrar com outros recursos utilizados	Integração	característica que refere-se à facilidade de entrosamento com outros recursos ou materiais educacionais.	**E**
	Adequação ao Ambiente Educacional	característica que refere-se à adequação ao ambiente educacional proposto pela escola	E
	Adequação aos Objetivos Educacionais	característica que refere-se à integração aos objetivos educacionais da escola	E
	Adequação ao Currículo da Escola	característica que refere-se à integração ao currículo regular da escola	E

Tabela A3.6 - Características de Qualidade de Sistemas de Informação Hospitalar

Sistemas de Informação Hospitalar (SIH) – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Flexibilidade - característica do SIH que torna possível o seu uso em diferentes situações, com um mínimo de modificações	Extensibilidade	Característica do SIH permitir extensões, com um mínimo de modificações, de modo a atender novas demandas de seus usuários.	I
	Adaptabilidade	característica do SIH que possibilita adaptações, com um mínimo de modificações, para atender a mudanças na maneira do usuário administrar e/ou realizar os serviços oferecidos.	I
	Independência dos dados	característica do SIH que possibilita a alteração da estrutura de armazenamento e do método de acesso aos dados do sistema, sem modificar a aplicação.	I
Facilidade de uso (sub-fatores a serem acrescentados ao fator facilidade de uso das características gerais)	Facilidade para a utilização clínica	característica do SIH permitir rapidez na entrada e consulta de dados, evitando assim interferências negativas na interação do médico com o paciente.	E
	Integração	Característica do SIH possuir mecanismos de apresentação e integração dos dados do paciente, embora estes dados possam estar distribuídos em diferentes locais.	I
	Uniformidade	Característica do SIH possuir interfaces padronizadas.	I
Rentabilidade (sub-fatores a serem acrescentados ao fator rentabilidade das características gerais)	Competitividade	Característica do SIH de, com sua utilização, criar vantagens competitivas com relação a instituições similares e/ou produzir novos serviços para a instituição.	E
	Valor Comercial	- característica do SIH referente ao seu potencial de comercialização para outras instituições.	**E**
	Valor de Marketing	Característica do SIH de, em função do seu uso, contribuir positivamente para a imagem da instituição.	E
Interoperabilidade- característica do SIH que possibilita o seu interfaceamento externo e interno.	Interoperabilidade interna	Característica do SIH possuir interfaces de qualidade, entre as suas várias partes e que estas possam ser, facilmente, adaptadas a novas necessidades.	I

Sistemas de Informação Hospitalar (SIH) – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Implementabilidade (sub-fatores a serem acrescentados ao fator implementabilidade das características gerais)	Adequação da infra-estrutura tecnológica	Refere-se à adequação do hardware, da rede de comunicação de dados, do software básico e do sistema gerenciador de banco de dados utilizados para o desenvolvimento e operação do SIH, e a conformidade existente entre os elementos dessa infra-estrutura.	I
Adequabilidade - característica do SIH que se refere à sua capacidade para atender aos fins desejados.	Apoio à atividades médicas	característica do SIH ser capaz de apoiar a decisão médica fornecendo informações ajustadas às atividades de diagnóstico e terapêutica, bem como, no acompanhamento dos pacientes.	E
	Apoio à pesquisa médica	Característica do SIH prover informações que auxiliem em atividades de pesquisa médica.	E
	Apoio à comunicação	característica do SIH oferecer mecanismos que permitam a comunicação entre os diferentes profissionais responsáveis e/ou envolvidos no atendimento de um determinado paciente	E
	Disponibilidade da informação	característica do SIH ser capaz de disponibilizar as informações para os seus usuários, no local onde estas são necessárias.	E
	Oportunidade	- característica do SIH ser capaz de fornecer informações com a rapidez necessária para que ela seja útil.	E
	Atualidade	característica do sistema prover informações que representem o estado atual da instituição no instante em que foram produzidas.	E
	Apoio à decisão gerencial	característica do SIH prover informações que sejam usadas como conhecimento e fundamento para a tomada de decisão gerencial.	E
	Apoio a trabalhos de rotina	característica do SIH de auxiliar seus usuários na execução dos seus trabalhos de rotina, automatizando-os sempre que possível.	E
	Concorrência	característica do SIH de permitir a sua utilização por vários usuários ao mesmo tempo.	I
	Integração com os objetivos da instituição	característica do SIH de ter os seus objetivos alinhados com os objetivos institucionais, isto é, os objetivos do SIH decorrem e contribuem para os objetivos globais da instituição.	E
	Integração a procedimentos manuais	Característica do SIH de ter uma integração adequada entre os processos administrativos manuais e os procedimentos automatizados.	E

Sistemas de Informação Hospitalar (SIH) – Confiabilidade da Representação			
Fator	**Sub-Fator**	**Descrição**	**Q**
Manipulabilidade (sub-fatores a serem acrescentados ao fator manipulabilidade das características gerais)	Uso de mídia especializada	Característica do SIH utilizar mídia especializada para favorecer a divulgação, compreensão e absorção do conteúdo de sua documentação.	I
Sistemas de Informação Hospitalar– Confiabilidade Conceitual			
Fator	**Sub-Fator**	**Descrição**	**Q**
Fidedignidade (sub-fatores a serem acrescentados ao fator fidedignidade das características gerais)	Cobertura	Característica do SIH que indica a quantidade de funções implementadas, em relação as identificadas como necessárias na especificação do sistema, para que a instituição atinja os seus propósitos.	E
Integridade – característica do SIH que se refere à capacidade de preservação dos seus dados e processamento em situações anormais. (sub-fatores que devem ser acrescentados à característica confiabilidade da ISO/IEC 9126)	Auditabilidade	característica do SIH criar trilhas e histórico do processamento realizado, visando a posterior realização de auditoria.	I
	Sinalização	característica do SIH prover alertas para a entrada de dados que estão fora das normas especificadas ou para combinações de dados que indicam problemas.	I
	Privacidade	característica do SIH possuir mecanismos que garantam a privacidade de médicos e pacientes, com relação à utilização das informações médicas.	I
	Prevenção a contigências	característica do SIH referente à capacidade de retroceder, em caso de falhas ou danos, a formas mais simples de processamento até, no pior caso, permanecendo apenas com procedimentos manuais.	I

Tabela A3.7 - Características de Qualidade do Prontuário Médico Eletrônico

Prontuário Eletrônico – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Adequabilidade (sub-fatores a serem acrescentados ao fator adequabilidade das características de sistemas de informação hospitalar)	Suporte à dados multimídia	característica do prontuário permitir a inclusão e manipulação de dados multimídia[1], gerados durante o atendimento ao paciente.	I
	Apoio a decisões políticas	característica do prontuário prover informações que auxiliem a instituição e/ou os órgãos governamentais no controle da qualidade e da produtividade dos serviços de saúde bem como, no estabelecimento de políticas de saúde a nível local e nacional.	E
	Apoio à informação do paciente	característica do prontuário apresentar as informações de uma maneira que permita aos médicos utilizá-lo para propósitos explanatórios com o paciente.	E
	Apoio à educação médica	característica do prontuário apresentar as informações de maneira que permita a sua utilização com propósitos educacionais.	E
	Apoio à administração	característica do prontuário fornecer informações para a administração, por exemplo, no faturamento de contas e controle de materiais.	E
	Estrutura do prontuário	característica do prontuário ter o seu conteúdo organizado, segundo uma estrutura (por exemplo, registro médico orientado a problema, evento ou ao paciente), que facilite lidar com sua complexidade e facilite o acesso pelos seus usuários.	E
	Apresentação	característica do prontuário permitir a apresentação dos dados dos pacientes em vários formatos a depender da necessidade dos médicos e enfermeiros (por exemplo, resumos, gráficos, planilhas) ou de acordo com diferentes abordagens (registro médico orientado a problema, evento ou ao paciente).	E

[1]Os tipos de dados gerados durante a rotina clínica são do tipo texto, imagens (imagens geradas de ressonância magnética, angiogramas, medicina nuclear e tomografia computadorizada), sinais (ECG), gráficos, vídeo (ECO) e som (batimentos cardíacos).

Prontuário Eletrônico – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Adequabilidade (cont.)	Apoio ao registro do processo de decisão médica	Característica do prontuário possuir mecanismos para registrar o processo de decisão[2] durante a assistência ao paciente.	I
	Apoio ao registro do diálogo clínico	Característica do prontuário possuir mecanismos para registrar todas as requisições feitas pelo corpo clínico (exames, procedimentos, opiniões de outros clínicos etc.) com as suas respectivas respostas.	I
	Apoio ao registro dos procedimentos	Característica do prontuário possuir mecanismos para registrar os relacionamentos entre as doenças de um paciente e os seus respectivos tratamentos.	I
	Autenticidade	Característica do prontuário permitir que os médicos e enfermeiros possam incluir as informações no prontuário numa maneira considerada natural por estes[3].	E
	Estabilidade dos dados	Característica do prontuário possuir uma política para gerenciamento dos dados, ou seja, para a inclusão e retenção dos dados, garantindo, assim, o significado destes dados por toda a vida do prontuário[4].	I
	Integração	Característica do prontuário possuir mecanismos de apresentação e integração dos dados do paciente, embora estes dados possam estar distribuídos em diferentes locais.	I

[2]Em muitos casos, a evidência na qual uma decisão foi baseada é tão importante quanto a própria decisão para o entendimento da evolução do curso de uma assistência.
[3]O prontuário deve permitir, por exemplo, declarações conflitantes, negativas, duvidosas sobre diagnósticos ou outros tipos de inferências.
[4]Nesta política deve ser definido, dentre outros, quais os dados que perdem e os que não perdem a validade após a alta clínica.

Prontuário Eletrônico – Confiabilidade Conceitual			
Fator	**Sub-Fator**	**Descrição**	**Q**
Integridade (sub-fatores a serem acrescentados ao fator integridade das características de sistemas de informação hospitalar)	Imutabilidade	característica do prontuário possuir mecanismos que garantam que as informações que contêm, após incluídas, não sejam modificadas, tanto por pessoas autorizadas à sua utilização como pelas não autorizadas.	I
	Permanência dos dados	característica do prontuário possuir mecanismos que garantam que o seu conteúdo não seja apagado (parcial ou totalmente).	I
	Atributabilidade	característica do prontuário possuir mecanismos que garantam a identificação do autor das informações e de quem realmente entrou com uma determinada informação no sistema, quando e onde entrou	I

Tabela A3.8 - Características de Qualidade de Sistemas de Acesso ao Público para Educação de Pacientes

Sistemas de Acesso ao Público para Educação de Pacientes – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Facilidade de utilização (sub-fatores a serem acrescentados ao fator integridade das características de sistemas de informação hospitalar)	Facilidade de Localização da Informação	refere-se às características existentes no sistema que possibilitam, com facilidade, a localização dos diferentes assuntos pelos usuários-finais.	E
	Facilidade de Memorização	refere-se às características existentes no sistema que facilitam aos usuários finais, a memorização de informações importantes.	E
	Uniformidade	refere-se às características que fazem o sistema comportar-se sempre de forma similar, permanecendo compreensível e familiar para os usuários finais.	E
	Motivação	refere-se às características do sistema que tornam sua utilização motivadora para o usuário-final.	E
	Auxílio em situações de erro	refere-se à característica do sistema dar apoio aos usuários em situações de erro.	E
	Evidência de inicialização	Refere-se às características do sistema que tornam fácil a sua inicialização e/ou reinicialização.	E
	Conforto da estação de trabalho	refere-se à característica do sistema estar instalado em equipamento de uso confortável, que possua dimensões adequadas aos diferentes tipos de usuários, sem requerer dos mesmos desgastes físicos desnecessários.	E
	Privacidade	refere-se à característica do sistema estar instalado em local que garanta a privacidade dos usuários finais durante o seu uso.	**E**
	Localização	característica do sistema estar instalado em local visível e de fácil acesso.	E
Implementabilidade (sub-fatores a serem acrescentados ao fator implementabilidade das características gerais)	Relevância	refere-se à viabilidade de construção do sistema considerando-se a sua utilidade e importância para a transmissão de informações aos pacientes.	E

Sistemas de Acesso ao Público para Educação de Pacientes – Confiabilidade Conceitual			
Fator	**Sub-Fator**	**Descrição**	**Q**
Legibilidade (sub-fatores a serem acrescentados ao fator legibiidade das características de sistemas de informação hospitalar)	Precisão da informação	refere-se à característica do sistema fornecer, somente, informações precisas, ou seja, que estas não coloquem o usuário em situações de dúvida.	E
	Adequabilidade da informação	refere-se à característica do sistema fornecer informações adequadas a seus usuários, considerando-se os aspectos sociais, culturais, étnicos, bem como os diferentes graus de escolaridade e de faixa etária.	E
Sistemas de Acesso ao Público para Educação de Pacientes – Confiabilidade da Representação			
Fator	**Sub-Fator**	**Descrição**	**Q**
Legibilidade (sub-fatores a serem acrescentados ao fator legibilidade das características das características gerais)	**Clareza da informação**	Refere-se à característica do sistema apresentar a informação de forma clara fazendo uso adequado de cores, letras, imagens, vídeos e som.	E
Manipulabilidade (sub-fatores a serem acrescentados ao fator legibilidade das características de sistemas de informação hospitalar. Estes atributos referem se à facilidade com que as informações podem ser localizadas no sistema (pelos pacientes) bem como na sua documentação e código (pelos desenvolvedores/mantenedores)).	**Apoio à navegação**	refere-se à existência de facilitadores que apoiem a navegação através do sistema hipermídia.	E
	Lateralidade	refere-se à característica do sistema permitir que os usuários percorram caminhos alternativos de acordo com a sua necessidade e interesse.	E
	Disponibilidade de atalhos	refere-se à existência de "atalhos" que facilitem a busca de informações para usuários experientes ou com maior facilidade de uso do sistema.	E

Tabela A3.9 - Características de Qualidade de Aplicações de Telemedicina

Sistemas de Acesso ao Público para Educação de Pacientes – Utilizabilidade			
Fator	**Sub-Fator**	**Descrição**	**Q**
Adequabilidade - característica do sistema de telemedinina que se refere à sua capacidade para atender aos fins desejados	Identificação do Paciente	Característica do prontuário eletrônico permitir a identificação correta e idêntica do paciente em diferentes localidades	I
	Disponibilidade da informação	**O sistema deve ser capaz de localizar** e disponibilizar informações do paciente que estão disponíveis ao longo do tempo em qualquer das localidades envolvidas	I
	Oportunidade	O sistema deve fornecer rapidamente quais são as informações do paciente que estão disponíveis em cada localidade	I
	Autenticidade	Característica do prontuário permitir que os médicos e enfermeiros possam incluir as informações no prontuário numa maneira considerada natural por estes.	E
	Confidencialidade	Característica do sistema proteger as informações do paciente armazenando de forma criptografada e evitando exposição inadvertida das informações do paciente	I
	Apoio a consulta ou tratamento a distância	Característica do sistema disponibilizar para o médico remoto, no mínimo, as mesmas informações do paciente disponíveis para o outro médico que participa da telemedicina	I
Integridade (sub-fatores a serem acrescentados ao fator integridade das características de sistemas de informação hospitalar)	Atributabilidade	característica do prontuário possuir mecanismos que garantam a identificação do autor das informações e de quem realmente entrou com uma determinada informação no sistema, quando e onde entrou	I

Anexo 4

Questionários de Identificação do Perfil de Especialistas

Tabela A4.1 - QIPE para grupo de enfermeiras

Questão	Itens
1) Marque sua experiência ou as atividades (cargos) que já exerceu, ou exerce, na área de enfermagem.	❑ Chefe de setor ❑ Professor (universitário) na área de enfermagem. ❑ Participante da equipe de desenvolvimento de software para cardiologia ❑ Usuário de software médico (ex.: sistema de informação, prontuário médico, etc.) ❑ Outros: _____ (ex.: medline, internet, etc.)
2) Já participou do desenvolvimento de quantos sistemas de computação?	❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7
3) Já usou quantos sistemas médicos (ex: sistemas de informação, prontuário eletrônico, etc.)?	❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7
4) Que tipos de atividades já realizou no desenvolvimento de sistemas?	❑ Definição do sistemas ❑ Entrevistas de elicitação ❑ Elaboração de modelos (estrutura de dados, etc.) ❑ Elaboração de casos de teste ❑ Avaliação do sistema (análise de resultados e testes)
5) Como você classificaria seu entendimento sobre o produto de software em questão?	❑ Excelente ❑ Bom ❑ Médio ❑ Baixo ❑ Nenhum
6) Marque a opção que melhor classifica seu grau de escolaridade	❑ Enfermeira com doutorado ❑ Enfermeira com mestrado ❑ Enfermeira com especialização ❑ Enfermeira ❑ Auxiliar de enfermagem
7) Quantos anos de experiência como enfermeira?	❑ Nenhum ❑ Entre 1 e 5 ❑ Entre 5 e 10 ❑ Mais de 10

Tabela A4.2 - QIPE para grupo administrativo

Questão	Itens
1) Marque sua experiência ou as atividades (cargos) que já exerceu, ou exerce, na área de administrativa hospitalar.	❑ Diretor ❑ Gerente da equipe administrativa ❑ Chefe de Setor ❑ Recepcionista ❑ Membro da equipe financeira ❑ Outros: _____
2) Já participou do desenvolvimento de quantos sistemas de computação?	❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7 _____
3) Já usou quantos sistemas médicos (ex: sistemas de informação, prontuário eletrônico, etc.)?	❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7 _____
4) Que tipos de atividades já realizou no desenvolvimento de sistemas?	❑ Definição do sistemas ❑ Entrevistas de elicitação ❑ Elaboração de modelos (estrutura de dados, etc.) ❑ Elaboração de casos de teste ❑ Avaliação do sistema (análise de resultados e testes)
5) Como você classificaria seu entendimento sobre o produto de software em questão?	❑ Excelente ❑ Bom ❑ Médio ❑ Baixo ❑ Nenhum
6) Marque a opção que melhor classifica seu grau de escolaridade	❑ Doutorado ❑ Mestrado ❑ Especialização ❑ Terceiro grau ❑ Segundo grau ❑ Primeiro grau

Tabela A4.3 - QIPE para grupo de desenvolvedores de software

Questão	Itens
1) Marque sua experiência ou as atividades (cargos) que já exerceu, ou exerce, na área de computação.	❑ Gerente de Projeto ❑ Professor (universitário) de informática ou de cursos de informática de mais de 40 horas ❑ Analistas de sistemas ❑ Projetista ❑ Usuário ❑ Outra: _____
2) Já participou do desenvolvimento de quantos sistemas de computação?	Grande porte: ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7 _____
	Médio porte: ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7 _____
	Pequeno porte: ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7 _____
3) Já participou do desenvolvimento de quantos sistemas de computação na área médica?	❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7 _____
4) Em que fases do ciclo de vida de um produto de software já participou?	❑ Especificação de requisitos ❑ Projeto ❑ Codificação ❑ Teste ❑ Manutenção ❑ Outra: _____
5) Quantos produtos de software (especificações, projetos, sistemas, etc.) similares ao que está sendo avaliado já desenvolveu?	❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7 _____
6) Como você classificaria seu entendimento sobre o produto de software em questão?	❑ Excelente ❑ Alto ❑ Bom ❑ Médio ❑ Baixo ❑ Nenhum

Questão	Itens
7) Marque a opção que melhor classifica seu grau de escolaridade	❑ Doutorado (área) _____ ❑ Mestrado (área) _____ ❑ Especialização (área) _____ ❑ Terceiro grau (área) _____ ❑ Segundo grau
8) Marque os subitens (e a quantidade a eles referente), que dizem respeito a seu grau de treinamento em informática:	Cursos (até 8 h): ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 ❑ Maior que 7 _____
	Cursos (até 40 h): ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 Maior que 7 _____
	Cursos (mais de 40 h): ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 Maior que 7 _____
	Simpósios/Congressos: ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 Maior que 7 _____
	Publicações de artigos nacionais: ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 Maior que 7 _____
	Publicações de artigos internacionais: ❑ Nenhum ❑ Entre 1 e 2 ❑ Entre 3 e 7 Maior que 7 _____

Anexo 5

Modelo de Classes das Ferramentas Q-*fuzzy* e KED

As Figura A5.1 e Figura A5.2 mostram, respectivamente, o modelo de classes da ferramenta Q-*fuzzy* e KED.

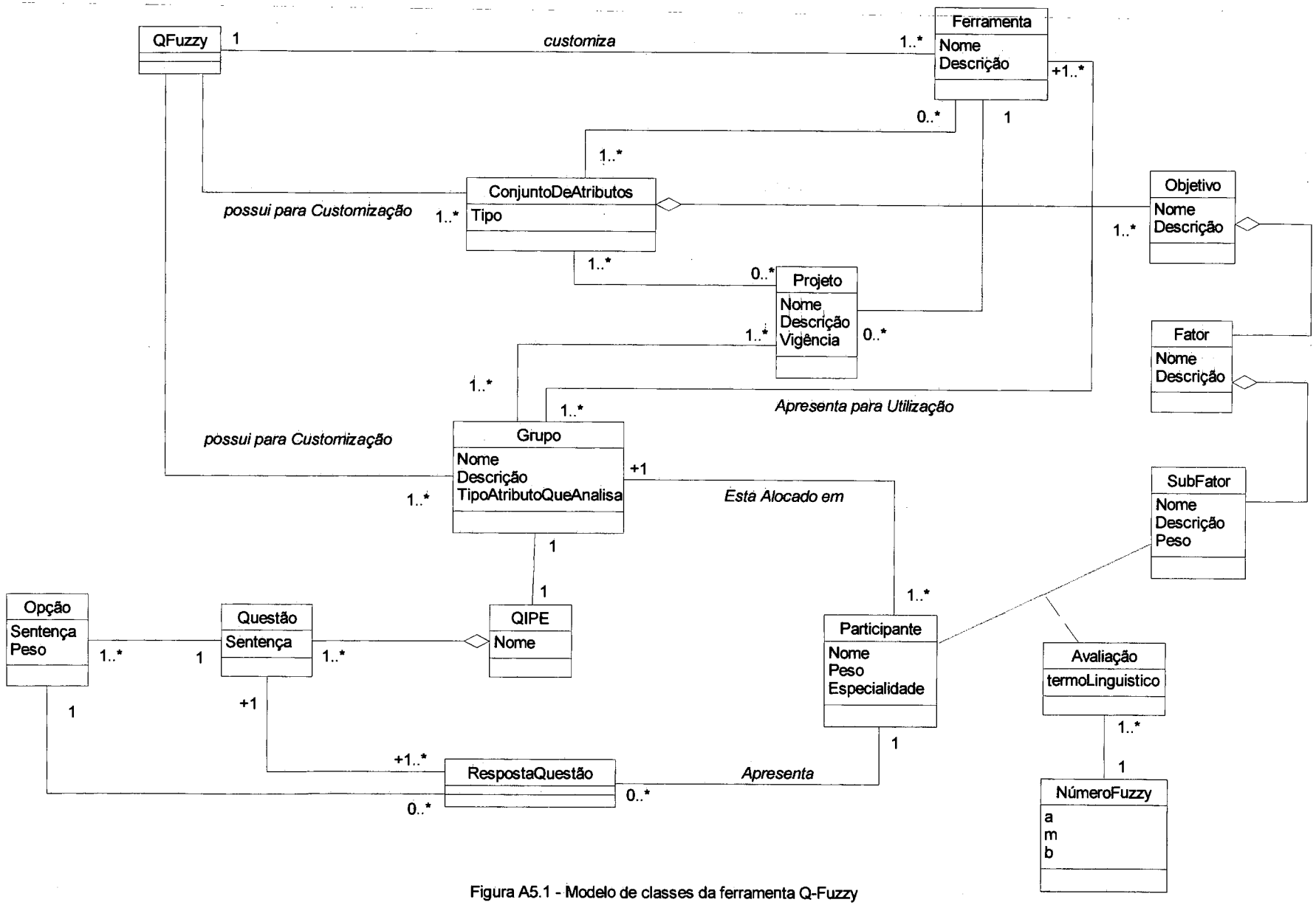

Figura A5.1 - Modelo de classes da ferramenta Q-Fuzzy

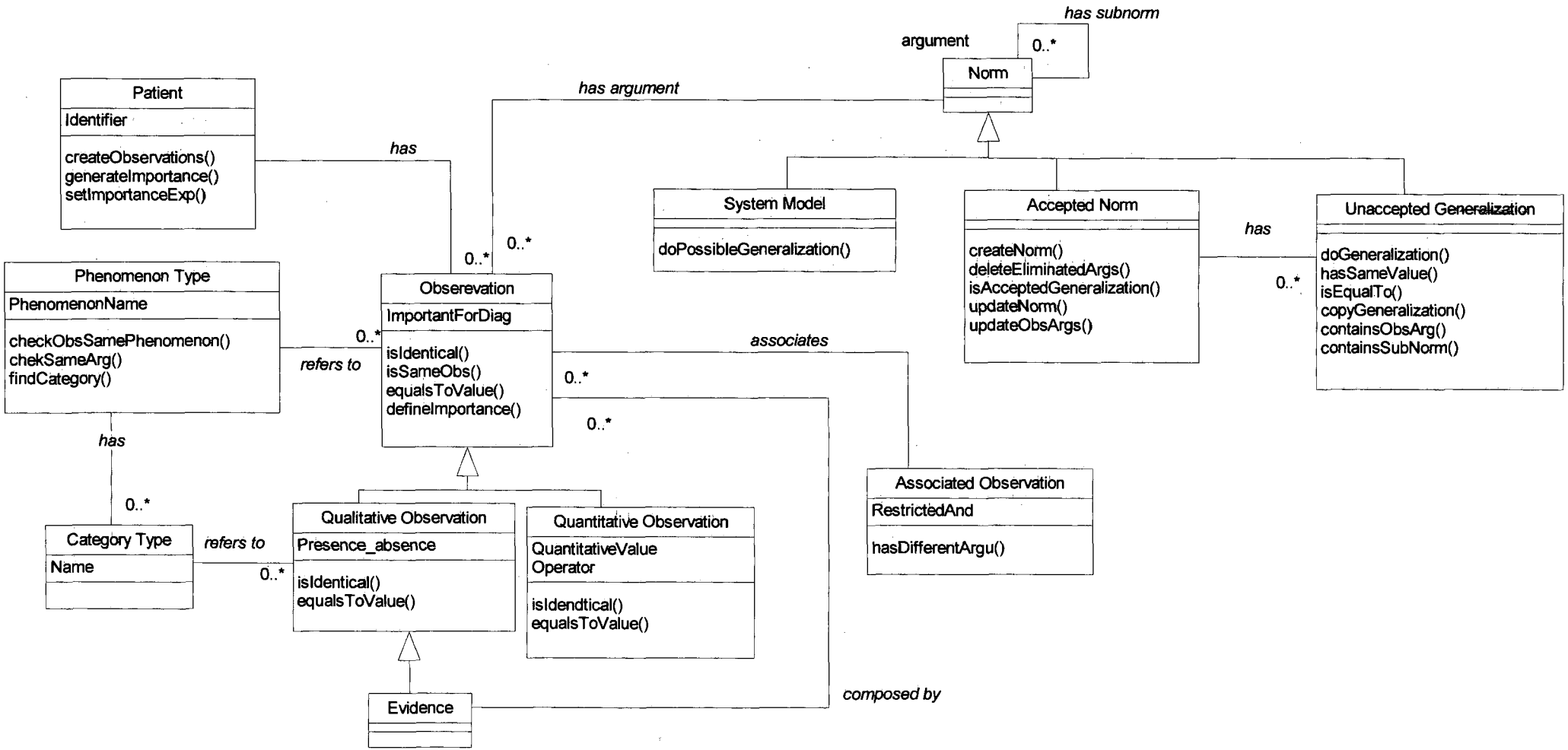

Figura A5.2 - Modelo de classes da ferramenta Ked