Ambientes de Desenvolvimento de Software Orientados a Domínio

Káthia Marçal de Oliveira[1], *Guilherme Horta Travassos*[1]
Crediné S. de Menezes[2], *Ana Regina C. da Rocha*[1]

[1]COPPE Sistemas / Universidade Federal do Rio de Janeiro
Caixa Postal 68511 – Cep 21945-970 - Rio de Janeiro
Fone/Fax: 0055 21 5902552

[2]Departamento de Informática — Universidade Federal do Espírito Santo
Av. Fernando Ferrari s/n – Cep 290060-900 – Vitória- ES

e-mail: {kathia, ght, darocha}@cos.ufrj.br , credine@inf.ufes.br

Resumo

Desenvolver software em um domínio não familiar para os desenvolvedores não é uma tarefa simples. Acreditando que o uso do conhecimento do domínio pode contribuir para redução das dificuldades no processo de desenvolvimento, e portanto, torná-lo mais produtivo, definimos Ambientes de Desenvolvimento de Software Orientados a Domínio. Estes ambientes tornam disponível o conhecimento sobre o domínio da aplicação numa representação simbólica utilizando ontologias do domínio e a identificação de tarefas possíveis de serem realizadas no domínio em questão. Ferramentas específicas do domínio introduzidas no ambiente tornam possível o uso desse conhecimento ao longo do processo de desenvolvimento. Nesse artigo, apresentamos a concepção desses ambientes no que se refere à definição e uso do conhecimento e à sua construção, de forma a prover facilidades que apoiem a orientação ao domínio.

Palavras-chave: Ambiente de Desenvolvimento de Software, I-CASE, Ontologia, Engenharia do Domínio.

Abstract

Developing software in a non familiar domain is not an easy task. We believe that the use of domain knowledge during the software development can render this process easier and increase productivity. To support this idea we defined Domain-Oriented Software Development Environments. Those environments make available knowledge about the domain in a symbolic representation by using domain ontology and the identification of potential tasks performed in the domain. Domain-specific tools introduced into the software development environment guide the use of this knowledge throughout software development process. This paper presents the concepts behind the definition and construction of Domain-Oriented Software Development Environments, discussing how this new family of environments can support the development of software products.

Key-words: Software Development Environments, I-CASE, Ontology, Domain Engineering.

1. Introdução

Um dos problemas cruciais no desenvolvimento de software e que afeta diretamente a qualidade do produto, é a identificação correta do que o sistema a ser desenvolvido deve atender, isto é, sua especificação de requisitos. Esta questão é ainda mais crítica quando os desenvolvedores de software trabalham num domínio que não conhecem. Neste caso, a produtividade no desenvolvimento também é prejudicada. Observamos essa situação nos últimos seis anos ao trabalharmos no desenvolvimento de diferentes tipos de produtos de software para o domínio de cardiologia em colaboração com a Unidade de Cardiologia e Cirurgia Cardiovascular da Universidade Federal da Bahia/Fundação Bahiana de Cardiologia (UCCV/FBC). Durante esse período, a equipe de desenvolvimento sofreu várias alterações de pessoal. Percebemos que, além dos conceitos e tarefas relacionadas às especificidades do projeto, cada novo integrante da equipe tinha também necessidade de entender os conceitos básicos de cardiologia antes de começar as suas atividades nos projetos.

Comparando este cenário a outras situações de desenvolvimento em diferentes empresas das quais participamos ao longo deste anos, percebemos que esse é um cenário comum em várias organizações, seja com a integração de novos desenvolvedores a equipes de desenvolvimento, ou com as empresas externas contratadas face à terceirização dos serviços, abordagem cada vez mais utilizada para o desenvolvimento de software. Devido à diversidade dos trabalhos, os desenvolvedores de empresas contratadas precisam conhecer um domínio diferente a cada novo projeto e, geralmente, este conhecimento está disseminado 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 à não disponibilidade do tempo necessário, ou às vezes à falta de mesmo interesse, para fornecer aos contratados o conhecimento do domínio.

Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD), definidos neste 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 (BROWN *et al*, 1992), mas incorporando um novo fator: o conhecimento de um domínio específico.

Iniciamos esse artigo (seção 2) apresentando as diferentes pesquisas realizadas em Ambiente de Desenvolvimento de Software (ADS) até nossa definição de Ambientes de Desenvolvimento de Software Orientados a Domínio. As duas seções seguintes (seções 3 e 4) se concentram nas principais características de ADSOD. Na seção 5, apresentamos como se da a construção de um ADSOD. Finalmente na seção 6, apresentamos as conclusões e perspectivas futuras desse trabalho.

2. De ADS a Ambientes de Desenvolvimento de Software Orientados a Domínio

Desde o surgimento dos ADS diferentes pesquisas tem sido realizadas. Algumas experiências de construção de ADS resultaram em ambientes de uso acadêmico e comercial. Essas experiências buscavam explorar características específicas, tais quais apoiar: a gerência de configuração e versões (BERNAS, 1995, CONRADI et al., 1994); o desenvolvimento orientado a objetos (BERNAS, 1995, GROTH et. al., 1995); a reutilização de software (TAYLOR *et al.*, 1995, CONRADI e KARLSSON, 1995, WERNER *et al* 1997); a colaboração e à cooperação (BANDINELLI et al., 1996, GRUNDY et al., 1995); e, finalmente, a modelagem e orientação baseada em processos de software (GARG e JAZAYERI, 1995, VASCONCELOS e WERNER, 1997, ARAUJO, 1998).

Com a disseminação das pesquisas em ADS, diferentes grupos fizeram propostas visando aprimorar e de certa forma evoluir o apoio ao desenvolvimento de software considerando características particulares do domínio para o qual os ADS são definidos e construídos. Como é comum nestes casos, a definição de processos de desenvolvimento, utilização de modelos e métodos, suporte ao armazenamento de dados e uso de ferramentas representam os aspectos fundamentais que se buscava identificar. Alguns dos principais projetos nesse sentido são os ambientes baseados em arquiteturas específicas do domínio (DSSA- *Domain-Specific Software Architecture)* (TAYLOR *et al.*, 1995), os ambientes baseados em conhecimento para apoio à engenharia de software (KBSE - *Knowledge-Based Software Engineering*) (GOMMA *et al.*, 1996) e os ambientes de projetos orientados a domínio (DODE - *Domain-Oriented Design Environment*) (FISCHER, 1994).

Arquiteturas de software específicas do domínio (DSSA) 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 em um mesmo 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 podem ser desenvolvidas.

Ambientes baseados em conhecimento para apoio à engenharia de software (KBSE) 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. A modelagem do domínio refere-se ao desenvolvimento de requisitos, da especificação e de uma arquitetura reutilizável, para uma família de sistemas em um determinado domínio de aplicação.

Ambientes de projetos (*design*) orientados a domínio (DODE) são sistemas complexos que apoiam as atividades de projeto em domínios específicos. DODE tem como objetivo apoiar as atividades *upstream* (transição do entendimento do problema para a etapa de especificação). Estes ambientes são caracterizados por um conjunto de componentes que consideram 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; catálogo de projetos realizados para permitir o estudo baseado em casos e reuso; entre outros. Uma característica importante desses ambientes é o seu aspecto evolutivo apoiado pela definição de um processo de construção e uso do ambiente em que o conhecimento é inicialmente modelado e continuamente evoluído e reorganizado.

Esses trabalhos, no entanto, pouco discutem 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 sugerindo a realização de adaptações. Os ambientes propostos por FISCHER (1994) apesar de abordarem o entendimento do problema, chamado de atividades *upstream*, não propõem nenhuma forma bem definida de organizar esse conhecimento. Além disso, DODE tem sido desenvolvidos 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 pouca relação com interface e interconexões (NING, 1994).

A identificação da necessidade de apoio ao desenvolvimento orientado ao domínio (conforme descrito na seção 1) aliada às limitações dos ambientes convencionais para apoiar este tipo de desenvolvimento, levou à definição de Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD) (OLIVEIRA *et al.* 1999a). 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 poderão ser desenvolvidos.

A definição 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 anteriormente, 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 software.

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 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. No que se refere, à questão *(iii)*, deve-se considerar o estudo e a investigação do domínio nas diferentes atividades do processo de software apoiando a sua realização. A seguir apresentamos a Teoria do Domínio e como se dá a orientação ao domínio em um ADSOD.

3. 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. Ontologia (GRUBER, 1995) é uma especificação explícita de uma conceituação, ou seja, uma especificação explícita dos objetos, conceitos e outras entidades que assumimos existir em uma área de interesse, além das relações entre estes conceitos e restrições expressas através de axiomas. Uma ontologia do domínio (van

HEIJST *et al.*, 1997) expressa uma conceituação para um domínio particular, definindo restrições na estrutura e conteúdo do conhecimento desse domínio.

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 de forma a permitir 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 na interação com os 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 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 a 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, em geral, 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. 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. 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 HICKMAN *et al.* (1992). 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 a 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. Um exemplo de uma Teoria do Domínio para cardiologia pode ser encontrado em (OLIVEIRA *et al.*, 1999b).

3.1 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 (GÓMEZ-PÉREZ *et al.*, 1996, LÓPEZ *et al.*, 1999, GRUNINGER e FOX, 1995, USCHOLD e GRUNINGER, 1996). Cada um desses métodos oferece além da definição de um processo sistemático, características essenciais importantes na engenharia de um domínio: os aspectos de elicitação (USCHOLD e GRUNINGER, 1996), formalização (GRUNINGER e FOX, 1995) e documentação (GÓMEZ-PÉREZ *et al.*, 1996, LÓPEZ *et al.*, 1999). A Engenharia do Domínio em um ADSOD utiliza uma 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: identificação do propósito; definição; formalização; e, codificação. Atividades de apoio estão continuamente presentes em cada uma das etapas, e corresponde 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 de qualidade (USCHOLD e GRUNINGER, 1996). A integração com ontologias existentes pode ser uma atividade bastante comum quando existem várias sub-teorias a serem definidas. A Figura 1, apresenta o processo de Engenharia do Domínio.

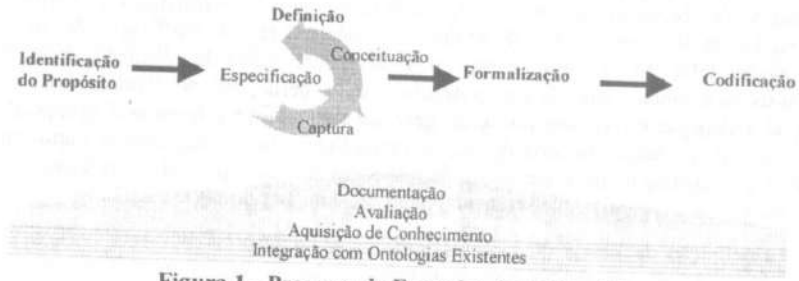

Figura 1 - Processo de Engenharia do Domínio

Na etapa de **Identificação do Propósito** é descrito o objetivo de apoiar o desenvolvimento de sistemas em um determinado domínio, limitando o seu escopo e descrevendo o cenário de sua utilização, que são geralmente problemas ou situações que mostram a utilidade de se ter orientação ao domínio.

A **Definição** da Teoria do Domínio é a etapa mais longa do processo e refere-se à modelagem do domínio propriamente dita. Esse etapa 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). 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 LÓPEZ *et al.* (1999). Não existe um ponto de início específico para a execução dessas atividades. A filosofia adotada deve ser *middle-out*, que significa capturar os termos mais significativos do domínio, procurar entendê-los e defini-los, especificar os requisitos e retornar para uma captura e conceituação mais detalhada.

A etapa de definição pode requerer várias iterações a depender do tamanho do domínio e da experiência da equipe que está realizando a Engenharia do Domínio. Em uma primeira iteração é identificado o conjunto de sub-teorias. Para isso, é realizado um *brainstorming* (USCHOLD e GRUNINGER, 1996) produzindo um conjunto de termos e frases potencialmente relevantes para o domínio. 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. É, então, identificada 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.

A partir daí, várias iterações são realizadas para a definição de cada sub-teoria, onde o grupo de conceitos é exaustivamente explorado e detalhado. Técnicas de elicitação de conhecimento como entrevistas, pesquisa bibliográfica 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: 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).

Após todas as definições é realizada a definição de restrições axiomáticas para os conceitos e sub-teorias. 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.

Finalmente é realizada uma última iteração para identificar as tarefas e mapear as tarefas com as sub-teorias definidas. A etapa de **Formalização** consiste em escrever a ontologia na linguagem de representação escolhida, ou seja, definir, explicitamente, conceitos e relações através de uma linguagem formal. 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.

A etapa de **Codificação** corresponde à implementação da formalização definida na etapa anterior. Com isso, a Teoria do Domínio passa a estar disponível, no ambiente, como módulos de conhecimento 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.

3.2 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 a 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 a 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 possuirem 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. 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 (ISO/IEC 12207, 1995). De forma geral, as principais atividades de desenvolvimento de software são: análise/definição do sistema, planejamento, análise de requisitos, projeto, codificação, teste e manutenção. 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 à 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 estabelecer 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 (como por exemplo nas atividades de análise e projeto). A Investigação do Domínio é apoiada por ferramentas específicas do domínio construídas utilizando a linguagem definida na Teoria do Domínio. Ferramentas acessam e atualizam com instâncias a Teoria do Domínio de acordo com a atividade específica que estão apoiando.

Um fator importante de ser considerado, na definição de processos de desenvolvimento de software é que um processo deve, geralmente ser particularizado para tipos de software, para a organização em que será 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 julgue 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.

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 é uma atividade sempre trabalhosa, e se torna mais complexa quando o engenheiro de software não tem conhecimento do domínio pois, além da identificação de quais são os requisitos do software, tem que entender cada um dos conceitos referidos pelos usuários, 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 (OLIVEIRA *et al.*, 2000a). A edição do conhecimento em KED é feita através da introdução de casos de pacientes pelos especialistas. A entrada de um caso é feita utilizando os conceitos da teoria de domínio de cardiologia, estabelecendo, dessa forma, uma linguagem familiar para os especialistas.

Na preparação para a elicitação de requisitos, ferramentas especialmente construídas para explorar o conhecimento do domínio podem mostrar os conceitos, descrições, relações e 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 na estruturação dos dados de acordo com a organização definida na Teoria do Domínio. Nesse sentido, WAND (1996) propôs uma correlação entre os conceitos ontológicos e os utilizados em modelagem de dados do tipo entidade-relacionamento (Tabela 2). De forma semelhante tem-se a correlação para a modelagem de objetos (PARSON e WAND, 1997) (Tabela 1). Além disso, a investigação do domínio permite a identificação das 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.

Tabela 1 - Relação 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 2- Relação 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

No que se refere às demais atividades do desenvolvimento de sistemas, um exemplo de apoio para a investigação do conhecimento do domínio é o uso de navegadores ou assistentes. Outras possibilidades estão sendo exploradas a depender de necessidades específicas.

5. Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio

Para permitir a construção de ADSOD é necessário um conjunto de ferramentas que apoie tanto sua definição (no que se refere a inclusão de Teoria de Domínio, tarefas e processo de software) quanto o seu uso durante o desenvolvimento de um software específico (no que se refere ao apoio a investigação do domínio utilizando a teoria do domínio definida).

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 que garanta a inclusão da sub-atividade investigação do domínio. Essas ferramentas devem obedecer as características apresentadas nas seções anteriores em relação a integração do conhecimento.

No que se refere a utilização desses ambientes, algumas ferramentas típicas de um ADSOD devem ser oferecidas para apoiar a orientação ao domínio. Nesse sentido é necessário uma ferramenta para apoiar a atividade de construção, um assistente de aprendizado do domínio que permita a pesquisa de conceitos do domínio através da estrutura hierárquica definida na ontologia do domínio e de outros tipos de relações entre os conceitos; ferramentas para elicitação específicas ao domínio construídas utilizando a Teoria do Domínio; ferramentas de apoio à modelagem para auxiliar o desenvolvedor a reutilizar definições e a estrutura de organização dos conceitos definidos na teoria do domínio; ferramenta para registro do uso de conceitos em cada projeto; e ferramentas para evolução da teoria do domínio. Além disso, no que se refere a avaliação da qualidade, é importante que se considere atributos de qualidade específicos para o tipo de software do domínio em questão como por exemplo atributos de sistemas de informação hospitalar e telemedicina no caso da área médica.

Para construção dessas ferramentas e do ADSOD propriamente dito estamos reutilizando a infra-estrutura da Estação TABA, um meta-ambiente capaz de gerar, através de instanciação, ambientes de desenvolvimento de software adequados às particularidades de processos de desenvolvimento e de projetos específicos (ROCHA *et al.* 1990). A Estação TABA possui facilidades para a definição de processos, métodos e ferramentas CASE a serem utilizadas no processo de desenvolvimento. Estes elementos estão organizados em um modelo de integração do ambiente, permitindo que diferentes ambientes sejam definidos e instanciados. A estrutura da Estação TABA (TRAVASSOS, 1994) é composta de um conjunto de componentes integrados que englobam o controle de processos, interface com o usuário, reutilização, cooperação, suporte inteligente, entre outros. Para construção de ADSOD usamos a infra-estrutura de conhecimento proposta por (FALBO *et al.* 1999a) e implementada na Estação TABA, que considera o uso de ontologias em servidores de conhecimento.

A primeira versão da Estação TABA foi implementada em plataforma Unix/Solaris utilizando linguagem de programação Eiffel, que embora tenha contribuído significativamente para a compreensão dos conceitos e aquisição de conhecimento relativo à definição, projeto e implementação de ambientes, meta-ambientes e ferramentas CASE utilizando o paradigma da orientação a objetos, restringiu sua utilização e dificultou seu uso em diferentes instalações. Esta realidade aliada ao novo contexto de ambientes de desenvolvimento de software orientados a domínio que nos propusemos construir, nos levou a re-implementar a infra-estrutura básica da Estação TABA utilizando C++, o que estendeu seu uso para plataformas diferentes de Unix.

Dessa forma requisitos para construção de ADSOD foram incluídos na Estação TABA e novas facilidades foram implementadas. No que se refere a ferramentas para apoio a definição de um ADSOD, foram construídos um editor de Teoria do Domínio, um editor de tarefas e um editor de processos. *O Editor de Teoria do Domínio* permite a introdução do conhecimento do domínio no ambiente. Este editor utiliza um meta-modelo de classes para organizar o conhecimento de forma que os objetos do domínio deste modelo tornam-se classes nos ambientes instanciados. Desta forma, todos os conceitos e relações do domínio

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 que garanta a inclusão da sub-atividade investigação do domínio. Essas ferramentas devem obedecer as características apresentadas nas seções anteriores em relação a integração do conhecimento.

No que se refere a utilização desses ambientes, algumas ferramentas típicas de um ADSOD devem ser oferecidas para apoiar a orientação ao domínio. Nesse sentido é necessário uma ferramenta para apoiar a atividade de construção, um assistente de aprendizado do domínio que permita a pesquisa de conceitos do domínio através da estrutura hierárquica definida na ontologia do domínio e de outros tipos de relações entre os conceitos; ferramentas para elicitação específicas ao domínio construídas utilizando a Teoria do Domínio; ferramentas de apoio à modelagem para auxiliar o desenvolvedor a reutilizar definições e a estrutura de organização dos conceitos definidos na teoria do domínio; ferramenta para registro do uso de conceitos em cada projeto; e ferramentas para evolução da teoria do domínio. Além disso, no que se refere a avaliação da qualidade, é importante que se considere atributos de qualidade específicos para o tipo de software do domínio em questão como por exemplo atributos de sistemas de informação hospitalar e telemedicina no caso da área médica.

Para construção dessas ferramentas e do ADSOD propriamente dito estamos reutilizando a infra-estrutura da Estação TABA, um meta-ambiente capaz de gerar, através de instanciação, ambientes de desenvolvimento de software adequados às particularidades de processos de desenvolvimento e de projetos específicos (ROCHA *et al.* 1990). A Estação TABA possui facilidades para a definição de processos, métodos e ferramentas CASE a serem utilizadas no processo de desenvolvimento. Estes elementos estão organizados em um modelo de integração do ambiente, permitindo que diferentes ambientes sejam definidos e instanciados. A estrutura da Estação TABA (TRAVASSOS, 1994) é composta de um conjunto de componentes integrados que englobam o controle de processos, interface com o usuário, reutilização, cooperação, suporte inteligente, entre outros. Para construção de ADSOD usamos a infra-estrutura de conhecimento proposta por (FALBO *et al.* 1999a) e implementada na Estação TABA, que considera o uso de ontologias em servidores de conhecimento.

A primeira versão da Estação TABA foi implementada em plataforma Unix/Solaris utilizando linguagem de programação Eiffel, que embora tenha contribuído significativamente para a compreensão dos conceitos e aquisição de conhecimento relativo à definição, projeto e implementação de ambientes, meta-ambientes e ferramentas CASE utilizando o paradigma da orientação a objetos, restringiu sua utilização e dificultou seu uso em diferentes instalações. Esta realidade aliada ao novo contexto de ambientes de desenvolvimento de software orientados a domínio que nos propusemos construir, nos levou a re-implementar a infra-estrutura básica da Estação TABA utilizando C++, o que estendeu seu uso para plataformas diferentes de Unix.

Dessa forma requisitos para construção de ADSOD foram incluídos na Estação TABA e novas facilidades foram implementadas. No que se refere a ferramentas para apoio a definição de um ADSOD, foram construídos um editor de Teoria do Domínio, um editor de tarefas e um editor de processos. *O Editor de Teoria do Domínio* permite a introdução do conhecimento do domínio no ambiente. Este editor utiliza um meta-modelo de classes para organizar o conhecimento de forma que os objetos do domínio deste modelo tornam-se classes nos ambientes instanciados. Desta forma, todos os conceitos e relações do domínio

ontologia considera conceitos que abrangem desde anatomia do coração até as evidências que um paciente apresenta e que são necessários para o diagnóstico e terapia. Foi, ainda, desenvolvida uma ferramenta específica do domínio para apoiar a atividade de elicitação de conhecimento (KED) (OLIVEIRA *et al.*, 2000a). A Figura 3 mostra a tela principal do CORDIS-SBC, onde do lado esquerdo tem-se o conjunto de atividades, e do lado direito o conjunto de ferramentas disponíveis para a realização de cada atividade. Na figura 3 é visualizado, também, o conjunto de ferramentas implementados para esse ambiente instanciado: uma ferramenta simples para aprendizado do domínio (Investigue), ferramenta para avaliação da qualidade (OLIVEIRA *et al*, 1999c) e a ferramenta para elicitação de conhecimento (KED). Finalmente, é mostrado a tela de consulta as informações sobre a atividade (tela Propriedades da Atividade) no que se refere ao conjunto de seus artefatos gerados, pré-atividades, sub-atividades e recursos humanos necessários).

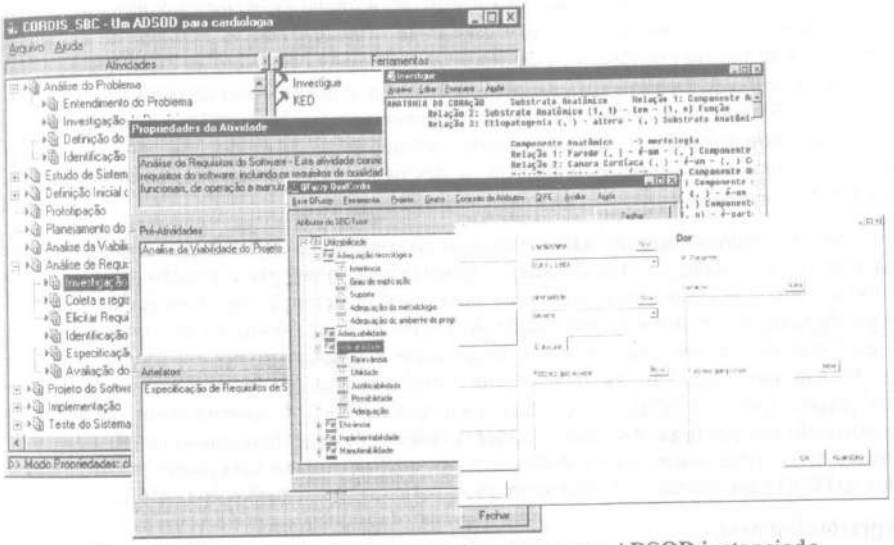

Figura 3 - Ferramentas de Desenvolvimento em um ADSOD instanciado

Ambientes instanciados pela Estação Taba apresentam o mesmo padrão de interface para a janela principal do CORDIS-SBC, e um conjunto de ferramentas específicas associadas ao processo definido. A construção de ADSOD na Estação TABA está apresentada em (OLIVEIRA *et al.*, 2000b) e está detalhadamente descrita em (OLIVEIRA *et al.*, 2000c).

6. Conclusão 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 de

projetos de desenvolvimento de software. Com este objetivo, definimos e construímos Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD).

Os ADSOD tem como objetivo apoiar os desenvolvedores no entendimento do domínio oferecendo conhecimento relevante sobre o domínio, ao longo de todo o processo de desenvolvimento. Neste artigo, apresentamos os aspectos teóricos de definição de um ADSOD no que se refere à modelagem do conhecimento e à orientação ao domínio ao longo do desenvolvimento de software. Este trabalho tem como contribuição a utilização de ontologias para a orientação a domínio no desenvolvimento de software, a definição e apoio a construção de ADSOD utilizando a infra-estrutura da Estação TABA.

Além do ambiente CORDIS-SBC apresentado neste artigo, outro ADSOD (GALLOTA, 1999) já foi definido e construído usando essa infra-estrutura. Este ADSOD, denominado NETUNO, apoia o desenvolvimento de software no domínio de Acústica Submarina sendo desenvolvido de acordo com as particularidades do GAS/IPqM (Grupo de Acústica Submarina do Instituto de Pesquisas da Marinha. De forma semelhante ao CORDIS-SBC, foi definida a Teoria do Domínio para Acústica Submarina e um processo de software específico para o GAS/IPqM. Foi, ainda, definido e implementado, no contexto desse ambiente, um navegador para investigação do conhecimento definido na teoria do domínio. Nosso próximo passo, é avaliação de uso desses ambientes no desenvolvimento de um sistema específico nas diferentes instituições.

Outros trabalhos que estão sendo realizados no contexto de ADSOD são: uma ferramenta para apoio à modelagem considerando a semântica de ontologia e relação com modelos entidade-relacionamento, uma ferramenta para apoio a definição de processo de software segundo normas e modelos de maturidade (MACHADO *et al*, 2000), um assistente que apoie a avaliação do processo nos ambientes instanciados e investigação do uso de ontologias de tarefas para modelagem de caso de usos. Um trabalho mais amplo foi iniciado visando definir um gerenciamento completo do conhecimento que extrapola o conhecimento do domínio, envolvendo normas organizacionais, melhores práticas de engenharia de software, relatos de experiências, entre outros. Este trabalho tem sido realizado como uma proposta de evolução dos ADSOD para ambientes de desenvolvimento orientados à organização (LIMA, 2000).

Agradecimentos

Agradecemos a Dr. Álvaro Rabelo Jr pelas sugestões a este trabalho e ao Dr. Augusto Ximenes pelo apoio fundamental na definição da ontologia para cardiologia. Agradecemos, também, ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela apoio financeiro ao projeto Ambientes de Desenvolvimento de Software Orientado ao Domínio.

Referências

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

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.

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

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

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.

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.

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 andKnowledge Engineering Conference*, pp. 170-175, Kaiserslautern, Alemanha, Jun.

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.

GALOTTA, C. "NETUNO: um Ambiente de Desenvolvimento de Software orientado ao Domínio de Acústica Submarina", In: Anais do Workshop de Teses em Engenharia de Software - XIII Simposio Brasileiro de Engenharia de Software, pp. 38-42, Florianópolis, Brasil, Out.

GARG, P., JAZAYERI, M., 1995, Process-Centered Software Engineering Environments, IEEE Computer Society Press.

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.

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.

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.

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

ISO/IEC 12207: Information technology - Software Life Cycle Processes, International Standard Organization, 1995.

LIMA, K., 2000 "Ambientes de Desenvolvimento de Software Orientados à Organização", Exame de Qualificação para Doutorado- Relatório Técnico COPPE/UFRJ, Março.

MACHADO, L.F, OLIVEIRA, K M E ROCHA, A.R, 2000, "Modelo para Definição de Processos de Software baseado na ISO/IEC12207, em Modelos de Maturidade e Características do Projeto", Terceiro Workshop Ibero-americano de Engenharia de Requisitos e Ambientes de Software (IDEAS'00), Cancun, México, abril.

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.

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.

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.

OLIVEIRA K., 1999, "Modelo para Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio"; Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, 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, "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.

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.; XIMENES, A. MATWIN, S. *et al.*, 2000a, "A Generic Architecture for Knowledge Acquisition Tools in Cardiology", In: Workshop on Intelligent Data Analysis in Medicine and Pharmacology – 14th European Conference on Artificial Intelligence, Berlin, Alemanha, Aug.

OLIVEIRA K., SANTOS, G., ZLOT F., *et al.*, 2000b, "A Estação TABA e Ambientes de Desenvolvimento Orientados a Domíio"; Caderno de Ferramentas do XIV Simpósio Brasileiro de Engenharia de Software, João Pessoa, Paraíba, Out.

OLIVEIRA K., SANTOS, G., ZLOT F., *et al.*, 2000c, "Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio na Estação TABA"; Terceiro Workshop Ibero-americano de Engenharia de Requisitos e Ambientes de Software, pp. 168-179, Cancun, México, Abr.

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

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.

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.

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.

TRAVASSOS, G.H.; ROCHA, A.R.C.da, 1994, "Um modelo para Construção e Integração de Ferramentas", VIII Simpósio Brasileiro de Engenharia de Software; Curitiba, Paraná; out.

USCHOLD, M., GRUNINGER, M., 1996, "Ontologies: principles, methods and applications", *The Knowledge Engineering Review*, Vol 11:2, pp 93-136.

VASCONCELOS e WERNER, 1997, "Software Development Process Reuse Based on Patterns", The 9th International Confereence on Software Engineering and Knowledge Engineering, pp. 97-104, Madrid, Spain, Jun.

WAND, Y, 1996, "Ontology as a foundation for meta-modelling and method engineering", *Information and Sofware Tecnhology*, No 38, 281-287.