BUSCA E RECUPERAÇÃO DE COMPONENTES EM AMBIENTES DE REUTILIZAÇÃO DE SOFTWARE

Regina Maria Maciel Braga Villela

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. Cláudia Maria Lima Werner, D.Sc.

Profª. Marta Lima de Queirós Mattoso, D.Sc.

Prof. Jano Moreira de Souza, PhD.

Prof. Marcos Roberto da Silva Borges, PhD.

Prof. Júlio César Sampaio do Prado Leite, PhD.

Profª. Ana Carolina Brandão Salgado, Docteur

RIO DE JANEIRO, RJ – BRASIL

DEZEMBRO DE 2000

VILLELA, REGINA M. M. BRAGA

Busca e Recuperação de Componentes em Ambientes de Reutilização de Software [Rio de Janeiro] 2000

X, 264 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2000)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Busca e Recuperação de Componentes
2. Reutilização de Software
3. Banco de Dados
4. Engenharia de Domínio
5. Infra-estrutura de suporte ao desenvolvimento de software
6. Projeto Odyssey
7. GOA++

I. COPPE/UFRJ II. Título (série)

Ao meu pai

e a minha mãe

Agradecimentos

Às Professoras Cláudia Werner e Marta Mattoso, por serem muito mais que orientadoras desta tese, mas também grandes amigas e incentivadoras do trabalho, mostrando e guiando os meus caminhos na vida acadêmica e compreendendo todos os meus momentos difíceis.

Ao Professor Jano de Souza, pela contribuição ao meu aprendizado no decorrer dos cursos de mestrado e doutorado.

Aos Professores Marcos Borges, Júlio Leite e Ana Carolina Salgado por participarem da banca examinadora da tese.

À Mônica Zopelari e à equipe de informática da Câmara Municipal do Rio de Janeiro, por colaborarem efetivamente na experimentação das propostas contidas nesta tese. Em especial ao Fábio Marx, Carlos Henrique Batista da Silva e Harlley Pacheco pela grande colaboração na especificação dos modelos do domínio do legislativo.

Aos amigos e parceiros Alessandreia de Oliveria, Marcelo Costa, Robson de Souza, Nelson Miller e Nicolaas Ruberg por acreditarem no trabalho aqui proposto.

Aos amigos do Projeto Odyssey, Márcio Barros, Leonardo Murta, Alexandre Dantas, Denis Silveira, Alexandre Correa, José Ricardo Xavier, Gustavo Veronesse e Hugo Vidal pelas contribuições, amizade e motivação.

Ao Marco e ao Dedé pelo incentivo, compreensão e grande amizade.

À amiga Fernanda, grande companheira de tantas viagens e incentivadora do trabalho.

À Rosa, Renata, Flávia e Fernanda Baião pelos ótimos momentos de amizade.

Aos amigos da COPPE pelo incentivo e motivação.

À Ana Paula e Patrícia por sempre estarem dispostas a ajudar.

À CAPES pelo apoio financeiro.

À Luciana e Rachel que, apesar de não aceitarem muito as minhas ausências, souberam compreender a importância deste trabalho na minha vida.

À minha mãe e ao meu pai, pelo sofrimento por conta das viagens, mas sempre incentivando os meus passos.

Ao Marcelo, por sempre incentivar e apoiar todos os meus projetos, compartilhando os bons e maus momentos.

Ao Pedro ou Carolina, cuja vinda foi adiada por conta deste trabalho.

A Deus, por permitir a conclusão de mais esta etapa importante da minha vida.

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

BUSCA E RECUPERAÇÃO DE COMPONENTES EM AMBIENTES DE REUTILIZAÇÃO DE SOFTWARE

Regina Maria Maciel Braga Villela

Dezembro/2000

Orientadoras: Cláudia Maria Lima Werner
Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

Esse trabalho apresenta uma abordagem para o uso da reutilização no desenvolvimento de software, através da especificação de mecanismos para o armazenamento, seleção e recuperação de componentes reutilizáveis, no contexto de um ambiente de reutilização de software. Tecnologias como agentes inteligentes, mediadores e ontologias do domínio foram utilizadas na especificação desta abordagem.

No entanto, para que o processo de reutilização possa ser realmente efetivo, não adianta termos mecanismos eficientes para o armazenamento, busca e recuperação de componentes, se não existem componentes reutilizáveis. Assim, esta tese também apresenta um processo de engenharia para a construção de componentes reutilizáveis, sendo estes componentes criados e/ou adaptados de maneira sistemática, com o objetivo principal de serem reutilizados em aplicações a serem desenvolvidas.

As propostas desta tese foram realizadas no contexto do Projeto Odyssey, em desenvolvimento na COPPE/UFRJ.

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

COMPONENT SEARCH AND RETRIEVAL IN A SOFTWARE REUSE ENVIRONMENT

Regina Maria Maciel Braga Villela

December/2000

Advisors: Cláudia Maria Lima Werner
Marta Lima de Queirós Mattoso

Department: Computer and System Engineering

This work presents an approach for effective reuse in software development, using some mechanisms for storage, selection and retrieval of reusable components. Ideas drawn from the field of intelligent agents, mediation and domain ontology are combined in this approach.

However, for providing an effective reuse process, the use of storage, search and retrieval mechanisms alone is not enough, if reusable components are not available. Therefore, this thesis also proposes a domain engineering process for a systematic creation/adaptation of reusable components within an application development.

The proposals of this thesis were done in the context of the Odyssey Project, under development at COPPE/UFRJ.

Índice

1 Introdução

1.1 Histórico

Em 1968, na conferência da NATO, que é considerada o marco de nascimento da engenharia de software, a palavra **reutilização de software** já surgia como uma das possibilidades de se superar a crise de software. O trabalho de MCILROY (1968) apresentado nesta conferência discutia uma proposta para a especificação de uma biblioteca de componentes reutilizáveis e a importância de se reutilizar componentes de software em contextos similares.

Na década de 70, pouco progresso e experimentação foram realizados na área (POYTON, LANERGAN, 1979). Como ponto significante, podemos ressaltar os projetos do departamento de defesa dos Estados Unidos com a linguagem de programação ADA. No início dos anos 80, a reutilização de software ganhou novo fôlego com os trabalhos conduzidos por FREEMAN (1980), (1983), (1984), cujo enfoque principal era na especificação de métodos e mecanismos para a operacionalização da reutilização de software, representação de componentes reutilizáveis e na organização de repositórios de componentes.

Ao longo dos anos 80 e 90, era esperada uma confirmação da reutilização de software como um grande avanço no aumento de produtividade no desenvolvimento de software. Trabalhos importantes foram desenvolvidos, com destaque para os trabalhos de NEIGHBOURS (1981), que foi o pioneiro na definição do termo **análise de domínio**, ARANGO (1988), cuja tese de doutorado foi uma contribuição importante para a sistematização do processo de criação de componentes reutilizáveis, através da especificação do que ele chamava de sistematização da análise de domínio, criando assim o termo **engenharia de domínio** e também PRIETO-DIAZ (1987), que, através da sua classificação por facetas, apresentou uma abordagem interessante e inovadora para a busca e seleção dos componentes reutilizáveis. Apesar destes trabalhos pioneiros e a conseqüente evolução dos mesmos, em termos práticos, houve uma evolução pequena na utilização efetiva da reutilização de software (KRUEGER,1992). Apesar das diversas pesquisas conduzidas na área, surgindo conferências importantes específicas para tratar sobre tema como o *Symposium on Software Reusability* (SSR) e a *International Conference on*

Software Reusability (ICSR), além de vários workshops e trabalhos apresentados em outros foros, ainda não se tem um consenso a respeito de qual a melhor abordagem a ser adotada.

Assim, passados trinta e dois anos desde o trabalho pioneiro de MCILROY (1968), vemos que a reutilização de software ainda não é uma prática comum entre os desenvolvedores. Mesmo com as novas tecnologias de desenvolvimento de software como a orientação a objetos evoluindo para componentização, objetos distribuídos, tecnologias de recuperação de informação baseadas em técnicas inteligentes, citando apenas algumas, a reutilização sistemática e organizada de software não é uma prática tão corrente.

Nossa suposição para esta questão é de que a reutilização de software até hoje não se tornou uma prática comum no desenvolvimento de software, dentre outros aspectos, pela falta de técnicas padronizadas e mais direcionadas para a construção, busca e recuperação de componentes reutilizáveis de forma que o reutilizador possa entender, selecionar e adaptar de forma facilitada o componente reutilizável ao seu contexto.

1.2 Definição do Problema

PRIETO (1991), apontava que o maior problema na reutilização de software em 1991 era a criação de componentes que pudessem ser reutilizados de forma facilitada em outras aplicações além daquela para o qual ele tinha sido originalmente definido. Ainda hoje este problema persiste, simplesmente porque as organizações/pessoas acreditam que se possa reutilizar componentes desenvolvidos para outras aplicações diretamente, sem nenhuma adaptação. Além disso, é comum que componentes desenvolvidos no contexto de uma dada aplicação "carreguem" com eles especificidades da aplicação, o que os torna muito difíceis de serem adaptados para outros contextos. PRIETO (1991) ressalta que este problema ocorre porque a construção de componentes reutilizáveis é realizada de maneira não sistematizada. Os componentes disponibilizados desta forma são difíceis de entender e adaptar para novas aplicações. Não se adota nenhum tipo de processo para a construção específica de componentes reutilizáveis e/ou adaptação de componentes desenvolvidos no contexto de alguma aplicação, para que estes possam ser reutilizados de forma facilitada. Assim, os desenvolvedores de software preferem criar seus próprios componentes do que reutilizar os disponíveis, dentre outras questões.

O entendimento é outra característica importante para promover a reutilização de um componente. KRUEGER (1992) ressalta que é necessário criar um mecanismo de abstração

que permita um melhor entendimento do componente reutilizável. A captura do relacionamento de uma dada abstração com um componente reutilizável é feita através de um mecanismo que Krueger denomina de "realização". Sem esta abstração, os desenvolvedores terão que imaginar o que cada componente faz, quando pode ser reutilizado e como pode ser reutilizado. Krueger considera componente reutilizável não só componentes de código, mas também estruturas de projeto, especificações, documentação, ou seja, todo produto criado durante um processo de desenvolvimento de software pode ser considerado componente reutilizável.

Neste contexto, podemos ter diferentes níveis de abstração para um dado componente, todos relacionados por ligações de "realização", sendo que um componente em um nível mais abstrato pode estar ligado através de ligações de "realização" a mais de um componente em um nível mais baixo. Acima de toda esta estrutura teríamos o nível mais alto de abstração que seria uma taxonomia de descrição dos componentes reutilizáveis disponíveis. Este mecanismo de abstração é também importante para facilitar a busca e seleção de um dado componente reutilizável. Sem sabermos como buscar e selecionar um componente, não podemos reutilizá-lo.

Outro ponto importante no contexto da reutilização de software ressaltado por Arango é a criação de "bibliotecas de componentes reutilizáveis" que permitam a identificação, seleção e busca de forma eficiente e facilitada (ARANGO, 1988). Diversas abordagens para a resolução deste problema já foram propostas na literatura, dentre elas (SEACORD, 1998), (MILLI *et al.*, 1995) (PRIETO, 1987) para citar apenas algumas. MILI *et al.* (1995) argumentam que para a recuperação de componentes reutilizáveis, devemos ter muito mais do que simplesmente um algoritmo de recuperação. Um dos principais aspectos que devem ser tratados por um mecanismo de recuperação de componentes é conseguir interpretar o problema, que está na cabeça do reutilizador e o qual ele espera que seja resolvido por algum componente reutilizável, e a solução, que é o componente reutilizável armazenado em um repositório. Além disso, atualmente, devemos lidar com questões tecnológicas como distribuição e heterogeneidade que é mais um aspecto a ser considerado na complexidade destes mecanismos de recuperação.

O armazenamento dos componentes é outro ponto a ser destacado. Um componente é um artefato de software e por isso é uma estrutura complexa e que necessita de mecanismos

adequados para seu armazenamento (BRAGA, WERNER, 1999). O armazenamento adequado destes componentes facilita posteriormente sua busca e recuperação, o que é importante para efetivar a reutilização. Assim, podemos resumir o problema atual da identificação, armazenamento, busca e recuperação de componentes reutilizáveis em quatro aspectos básicos: criação de componentes especificamente para serem reutilizados, entendimento destes componentes, tratamento de aspectos como distribuição e heterogeneidade e criação de uma semântica específica para facilitar a busca.

1.3 Motivação

ARANGO (1988) aponta três aspectos técnicos envolvidos na reutilização de software: i) **a aquisição de informação reutilizável**; ii) **sua representação**; e a iii) **reutilização da informação**.

Esta tese aborda os três aspectos mas como foco no item iii, ou seja, em criar mecanismos adequados para a efetiva **reutilização da informação**. Dentro deste item, podemos dizer que nos concentramos particularmente na criação de mecanismos para a identificação, seleção e recuperação de componentes reutilizáveis de maneira facilitada.

No entanto, concordamos com (HADJAMI *et al.*, 1995) quando dizem que o grau de reutilização de um conjunto de componentes depende fortemente, além da facilidade na busca por estes componentes, da qualidade dos mesmos.

Um dos aspectos relacionados à qualidade dos componentes é que estes devem ser criados e/ou adaptados de maneira sistemática, com o objetivo principal de serem reutilizados em aplicações em desenvolvimento. Para isso, conforme ressaltamos na seção anterior, é necessária a utilização de um processo de engenharia para a construção destes componentes, que no contexto desta tese, seguindo a conceituação definida por Arango, é denominado de **engenharia de domínio**.

Segundo KRUEGER (1992) e MILI *et al.* (1995), a abstração é um mecanismo importante para facilitar a busca por componentes. Mili ressalta que um reutilizador deve ter familiaridade com o vocabulário utilizado por um mecanismo de identificação, seleção e busca por componentes. Este vocabulário comum seria o conjunto de abstrações em mais alto nível descritas por Krueger, organizado em uma taxonomia. Esta taxonomia, reflete, em um nível alto de abstração, o conjunto de componentes passíveis de reutilização. Quanto mais específica for esta taxonomia, maiores serão as chances de reutilização.

Assim, uma melhor organização de conjuntos de componentes reutilizáveis de forma a facilitar a reutilização é a organização de bibliotecas de componentes divididas por domínio, pois desta forma, o universo de componentes a ser consultado é menor e as abstrações capturadas podem ser mais precisas. Esta taxonomia é denominada no contexto desta tese de **modelo de características**[1] e o processo utilizado na sua criação faz parte do processo de criação dos próprios componentes reutilizáveis, ou seja, a **engenharia de domínio**. Estes termos, para o leitor não familiarizado com a área, serão descritos em detalhe no capítulo 2.

No entanto, de nada adianta termos uma taxonomia, que permita um bom entendimento do domínio e identificação de abstrações de interesse, se, a partir desta taxonomia não conseguirmos "alcançar" os componentes a serem reutilizados. Desta forma, supomos ser imprescindível para o processo de reutilização a possibilidade de seguirmos as ligações "de realização" entre as abstrações e os componentes reutilizáveis. No contexto desta tese, estas ligações de "realização" são denominadas de "rastro". O estabelecimento destas ligações de rastro devem fazer parte do processo de especificação dos componentes.

Quando MILI *et al.* (1995) se referem a um vocabulário familiar ao reutilizador, devemos levar em consideração que a forma de **representar esta informação** também é importante. Para isso, devemos usar notações conhecidas pelos reutilizadores de forma que os mesmos possam identificar facilmente a informação requerida. Assim, a utilização de notações que seguem padrões pré-estabelecidos no desenvolvimento de software[2], como UML (BOOCH, 1998), facilitam a identificação dos componentes reutilizáveis, ao passo que se forem utilizadas notações não muito conhecidas do reutilizador, ainda teremos o "sobrecarga" de aprendizado da notação.

O armazenamento dos componentes é também uma característica importante para o processo de reutilização como um todo. SAMETINGER (1997) destaca que o armazenamento de componentes particionado por domínio de aplicação facilita a posterior reutilização dos mesmos. Outro ponto importante, atualmente, é que estudos genéricos na área de recuperação de informações em geral (BAESA-YATES *et al.*, 1999) sugerem que

---

[1] Este modelo de características é o modelo de *features* proposto pelo método FODA (KANG *et al.*, 1990) estendido, conforme será apresentado no capítulo 4.

[2] Neste aspecto específico, consideramos como prática no desenvolvimento de software a adoção do paradigma OO. Desta forma, consideramos a UML como um padrão de modelagem estabelecido e entendido pelos desenvolvedores.

devemos cada vez mais considerar o comportamento do usuário em relação às suas buscas passadas e atuais, de forma a melhorar suas buscas futuras. Para isso, existem diversas técnicas utilizadas em Inteligência Artificial para considerar este comportamento do usuário. Analisando a literatura disponível, pudemos notar que até o momento nenhum mecanismo de recuperação de componentes utiliza este tipo de técnica. Um mecanismo de recuperação de componentes deve levar ainda em consideração a característica distribuída e heterogênea dos componentes a serem recuperados, considerando a necessidade de busca por componentes remotos.

Desta forma, nossa principal motivação pode ser resumida na seguinte frase: Necessitamos de mecanismos eficientes para a busca e seleção de componentes reutilizáveis em um dado contexto. Para que estes mecanismos sejam eficientes, necessitamos da criação de taxonomias adequadas que facilitem esta busca e ligações semânticas entre as abstrações que formam a taxonomia e os componentes propriamente ditos, levando ainda em consideração a natureza distribuída e heterogênea dos componentes. Como premissa básica para que a reutilização de software seja eficiente, temos a qualidade do componente a ser reutilizado (KRUEGER, 1992) (MILLI *et al.*, 1995). Assim, necessitamos também da criação de um processo de engenharia de domínio que permita a especificação de componentes reutilizáveis de qualidade, atendendo as diversas fases do processo de desenvolvimento de uma dada aplicação, desde a fase de análise até a fase de codificação. Esta criação de componentes reutilizáveis é muito mais eficiente se particionada por domínios de aplicação (PRIETO,1991), (ARANGO, 1988), (COHEN, 1994), (GRISS *et al.*, 1998).

Na literatura pesquisada não foi encontrado um mecanismo que aborde estes aspectos de maneira abrangente e integrada, ou seja, levando em consideração a natureza possivelmente distribuída dos componentes, utilizando mecanismos adequados para o armazenamento destes componentes, especificando abstrações adequadas para a busca e recuperação dos componentes e além disso, lidando com a dicotomia entre o que o reutilizador tem de descrição do que seria seu problema e como a solução está representada na forma de componentes reutilizáveis (espaço do problema X espaço da solução).

Na literatura, são descritas abordagens integradas semelhantes à nossa proposta. A abordagem DRACO (NEIGHBOURS, 1981), (LEITE, 1994) para a construção de

componentes reutilizáveis se assemelha a nossa proposta nos seguintes aspectos: na especificação e construção de componentes reutilizáveis, englobando todas as etapas do processo de desenvolvimento de software; na captura de abstração e as ligações de "realização" destas abstrações, com o intuito de facilitar o entendimento e seleção de componentes reutilizáveis dos diversos níveis de abstração através de ligações de "realização"; e, na criação de uma taxonomia, no mais alto nível de abstração que facilite o entendimento, a seleção e a busca do componente. O diferencial da nossa proposta em relação a proposta DRACO é: na utilização de mecanismos para o tratamento específico do armazenamento de componentes, uma vez que o armazenamento dos componentes no DRACO é feito de maneira integrada ao ambiente como um todo ao passo que na nossa proposta, utilizamos um sistema de banco de dados para realizar este armazenamento; na utilização de mecanismos para recuperar componentes distribuídos; e na possibilidade de integração de componentes desenvolvidos por terceiros. Além disso, a abordagem utilizada pelo DRACO é uma abordagem generativa ao passo que a nossa abordagem é uma abordagem composicional.

O ambiente KBSEE (GOMMA, 1995) também possui uma abordagem semelhante a nossa proposta. O KBSEE utiliza diversos tipos de ferramentas para a efetivação da reutilização, através do uso de uma abordagem orientada a objetos. No entanto, para a utilização das ferramentas no desenvolvimento dos componentes de software, nos diversos níveis de abstração, são utilizadas diversas transformações na representação dos mesmos, o que compromete o desempenho do ambiente como um todo e sugere uma "fraca" integração entre estas ferramentas. Assim, o diferencial da nossa abordagem em relação ao KBSEE é na especificação de um conjunto de ferramentas integradas para o desenvolvimento de componentes reutilizáveis, a preocupação com a busca e recuperação eficientes destes componentes e a possibilidade de componentes de terceiros serem disponibilizados para o ambiente como um todo.

1.4 Objetivos

O objetivo principal desta tese é, portanto, definir e implementar uma infra-estrutura que permita a especificação, o armazenamento, identificação, seleção e busca de componentes reutilizáveis em um dado domínio de aplicação, de maneira integrada.

Portanto, nossa proposta de tese se concentra na criação de um processo que permita o desenvolvimento de componentes para a reutilização em todas as fases do desenvolvimento de software e na especificação e implementação de mecanismos para a identificação, seleção e busca de componentes reutilizáveis de acordo com a necessidades do usuário e que esta busca possa ser feita da maneira mais transparente e fácil possível. Além disso, especificamos os mecanismos para o armazenamento dos componentes desenvolvidos no Odyssey, utilizando para isso o gerente de objetos armazenados GOA++ (Mauro et al., 1997).

Partindo destes objetivos principais, iniciamos a investigação de quais mecanismos seriam mais adequados para criar este tipo de infra-estrutura. Levando em consideração a importância e necessidade de tal infra-estrutura e dos problemas levantados pelos pesquisadores na área (PRIETO, 1991), (KRUEGER,1992), (MILLI *et al.*, 1995), (ATKINSON, 1998), (SEACORD, 1998), (YEE *et al.*, 2000), identificamos alguns pontos onde o nosso trabalho deveria se concentrar:

i. Especificação de um conjunto de abstrações, segundo a definição apresentada em (KRUEGER, 1992), que permitisse uma melhor identificação de componentes reutilizáveis a serem selecionados;
ii. Utilização de notações familiares aos desenvolvedores/reutilizadores, para que a busca por componentes possa ser realizada de forma que o aprendizado de novas notações e/ou linguagens seja atenuado;
iii. Levar em consideração que o reutilizador deve ser auxiliado o máximo possível na busca por componentes reutilizáveis, considerando, conforme é ressaltado por MILI (1995), que o que ele deseja (problema) muitas vezes não está representado no conjunto de componentes reutilizáveis exatamente da mesma maneira (solução). Assim, é necessária a disponibilização de mecanismos que auxiliem o casamento entre o problema, que deve ser representado no formato e linguagem mais adequada ao usuário, e a solução, que deve ser representada de forma a poder atender com o máximo de precisão possível as necessidades dos usuários;
iv. Considerar os avanços tecnológicos atuais no desenvolvimento de software como agentes inteligentes, tecnologias de objetos distribuídos, desenvolvimento baseado em

componentes, Internet, entre outros, como possibilidades de auxílio ao desenvolvimento de tal infra-estrutura.

1.5 Justificativas

Analisando os objetivos apresentados e os pontos onde consideramos que o nosso trabalho pudesse trazer alguma contribuição, apresentamos a seguir algumas justificativas em relação aos mesmos.

Em relação à definição de um conjunto de abstrações (i) e a utilização de notações familiares ao desenvolvedor/reutilizador (ii), investigamos em um primeiro momento de que tecnologias teríamos disponíveis na própria área de reutilização de software. A criação de modelos de domínio se apresentou promissora, pela característica de capturar os principais conceitos e abstrações do domínio. Pesquisamos, então, diversos métodos de engenharia de domínio disponíveis na literatura como FODA ("*Feature Oriented Domain Analysis*") (KANG *et al.*, 1990), ODM ("*Organization Domain Modelling*") (SIMOS, 1996), OOSE (CHAN *et al.*, 1998), FODACom (GRISS *et al.*, 1998), entre outros. Todos estes métodos disponibilizam um modelo de domínio onde abstrações de mais alto nível no domínio, representando principalmente os conceitos importantes no domínio, são modeladas.

Pesquisamos também a área de recuperação de informação em geral (BAESA-YATES, FRAKES, 1992), (BAESA-YATES, RIBEIRO-NETO, 1999) e áreas atualmente relacionadas a esta tais como filtragem de informação na Internet (MOUKAS, 1997), ontologias (MENZIES *et al.*, 1998), (LUKE *et al.*, 1998), e especificação de metadados em bases de dados (WIEDERHOLD, 1997) (BAYARDO *et al.*, 1997). A área de recuperação de informação clássica pode ser descrita como o uso de técnicas para se coletar, representar, armazenar, organizar, acessar, manipular e apresentar informações que atendam a necessidade de um grupo de usuários (BAESA-YATES, FRAKES, 1992). A grande diferença da área de recuperação de informação da área de recuperação de dados é justamente a imprecisão da primeira, onde informações relacionadas podem ser recuperadas ao passo que na área de recuperação de dados, somente dados que atendam exatamente a consulta especificada utilizando uma expressão regular são recuperados.

Atualmente, uma grande parte das pesquisas em recuperação de informação está sendo direcionada para recuperação de informação na Internet (BAESA-YATES, RIBEIRO-NETO, 1999), envolvendo assim pesquisas relacionadas a modelagem, filtragem, classificação de informação, usos de metadados semanticamente ricos para a recuperação de informação, entre outros. O foco das pesquisas relacionadas a recuperação de informação nesta tese é justamente relacionada a recuperação de informação na Internet. Neste caso específico, observamos que a abordagem utilizada na maioria dos projetos é a de especificação de ontologias (metadados semanticamente ricos) e/ou uso de técnicas baseadas no perfil do usuário. Iniciamos, então, a investigação do que as ontologias poderiam contribuir para a especificação de modelos de domínio adequados para auxiliar na busca de componentes. Concluímos que as abordagens de modelo de domínio e ontologias (GUARINO, 1998) são bastante similares, sendo que os modelos de domínio, na sua concepção geral, têm maior liberdade em relação à representação, ao detalhamento e a variedade de ligações semânticas ao passo que um modelo ontológico possui uma estrutura mais rígida em relação à representação, justamente porque carrega um formalismo maior em relação a sua estrutura e representação.

Como o nosso objetivo principal é a busca por componentes, concordamos com outros autores (BAYARDO *et al.*, 1997) quando dizem que não precisamos de todo o formalismo da lógica para buscar informações distribuídas e heterogêneas, o que se aplica também à busca por componentes. Mais ainda, se obrigarmos os usuários a lidar com a sintaxe de linguagens formais para a especificação de ontologias, como KQML (FININ *et al.*, 1994) ou OntoLingua (ONTOLINGUA, 2000), um dos principais objetivos da nossa proposta, que é a facilidade para a especificação da busca, seria abandonado. Assim, optamos por representar a taxonomia das abstrações utilizando UML, por ser uma notação considerada padrão no desenvolvimento de software atual (OMG, 2000) e, portanto, mais familiar aos desenvolvedores. Desta forma, especificamos um modelo de abstrações do domínio o qual acreditamos que, para o nosso propósito de busca e seleção de componentes, seja adequado, permitindo inclusive, através dos relacionamentos entre diferentes modelos de domínio, a busca por componentes reutilizáveis em domínios correlatos.

Em relação ao auxílio no entendimento do problema e sua ligação com a solução (iii), partimos novamente para procurar na literatura metodologias que visassem a especificação

de componentes reutilizáveis não só no nível de código mas também em outros níveis de abstração. Os métodos de engenharia de domínio têm uma maior preocupação na definição de um modelo de domínio em mais alto nível de abstração, capturando as características comuns e variáveis do domínio (mapeamento do problema). No entanto, nenhum deles se preocupa em detalhar como seria a especificação e/ou adaptação de componentes reutilizáveis (solução), partindo-se das abstrações capturadas pelo modelo em mais alto nível de abstração. Todos possuem diretivas genéricas do que deve ser feito, mas não uma descrição detalhada de como fazê-lo. Assim, investigamos novos métodos para o desenvolvimento de aplicações baseado em componentes. Neste sentido, os métodos que consideramos mais promissores na área foram o RBSE (JACOBSON *et al.*, 1997) e Catalysis (D'SOUZA *et al.*, 1998). Estes dois métodos são baseados em avanços da tecnologia de orientação a objetos em direção à componentização de sistemas, adotando técnicas consideradas promissoras no desenvolvimento de software tais como o uso de padrões, principalmente padrões de projeto (GAMMA *et al.*, 1994), ênfase em tecnologias para distribuição, entre outros. No entanto, estes métodos dão ênfase ao desenvolvimento de uma única aplicação dividida em componentes e não na especificação de componentes que poderão futuramente ser reutilizados. Desta forma, tentamos unir métodos de engenharia de domínio, mais particularmente o método FODA (KANG *et al.*, 1990) com métodos de desenvolvimento baseado em componentes, principalmente o Catalysis (D'SOUZA *et al.*, 1998), criando assim o Odyssey-DE. Para verificar sua adequabilidade em um domínio real, realizamos um estudo de caso no domínio de processos legislativos. O detalhamento do Odyssey-DE é feito no capítulo 4 e sua utilização descrita no capítulo 6.

Considerando ainda a preocupação em auxiliar ao máximo o desenvolvedor na busca por componentes reutilizáveis (iii), fomos buscar na Inteligência Artificial (IA), mais especificamente na tecnologia de Agentes Inteligentes, e nos mecanismos de recuperação de informação na Internet (BAESA-YATES, RIBEIRO-NETO, 1999), soluções que poderiam ser adaptadas ao nosso problema. Técnicas como modelagem de usuário (FLEMING *et al.*, 1999), aprendizado de máquina (JENNINGS *et al.*, 1997), processamento de informação textual (SALTON, 1989) (BAESA-YATES *et al.*, 1999), entre outras, foram investigadas até chegarmos a um conjunto de técnicas que seriam adequadas ao nosso contexto. Assim, foi especificado um sistema multi-agente inteligente,

que, baseado no modelo de domínio, perfil do reutilizador, buscas passadas efetuadas por este reutilizador e técnicas de aprendizado, tenta auxiliar da melhor maneira possível o reutilizador em sua busca por componentes. Na literatura pesquisada para esta tese, não encontramos nenhum mecanismo de busca por componentes reutilizáveis com estrutura semelhante, onde se privilegia não só as abstrações apresentadas no modelo de domínio mas também o perfil do reutilizador, refinando desta maneira cada vez mais sua chance de sucesso. Com o objetivo de observar a utilização deste sistema específico, apresentamos no capítulo 6 um estudo de caso relacionado ao uso do sistema por um grupo de usuários.

Outro ponto importante é levar em consideração que, pela própria característica distribuída das organizações, que podem possuir unidades em locais diferentes e desta forma possuir componentes reutilizáveis desenvolvidos em unidades diferentes, ou que se pode querer reutilizar componentes desenvolvidos por terceiros e que podem estar armazenados remotamente, devemos possuir estruturas que permitam a identificação e busca destes componentes remotos (iv), de forma facilitada para o usuário. Neste sentido, pesquisamos as tecnologias disponíveis para a busca de informações distribuídas (ÖSZU, VALDURIEZ, 1999), metadados para recuperação de informações distribuídas (MENA, 1998), banco de dados federados, modelos globais e mediadores (WIEDERHOLD, 1995), entre outros. Analisando estas tecnologias e levando em consideração a divisão por domínios e a utilização de taxonomias também divididas por domínios (problema X solução), consideramos a tecnologia de mediadores a mais adequada para os nossos propósitos, pela possibilidade de divisão por domínios (WIEDERHOLD, 1997) e pela característica de se ter um modelo integrador para a busca, que no nosso caso específico é o modelo de abstrações do domínio. Este modelo integrador representa o espaço do problema e os componentes a serem recuperados representam o espaço da solução.

1.6 Evolução da Pesquisa

Esta tese foi originalmente motivada a partir do nosso trabalho no mestrado (BRAGA, 1995) que foi a especificação e implementação de um *framework* para automação de escritório. Esta experiência foi motivadora no sentido de apresentar a dificuldade de se expressar uma arquitetura que fosse genérica o bastante para auxiliar no desenvolvimento

de aplicações no domínio de automação de escritório e também a dificuldade de se encontrar o estilo arquitetural mais adequado para uma dada aplicação.

A partir desta constatação investigamos trabalhos relacionados a processos de desenvolvimento mais adequados para a criação de *frameworks*. O trabalho de mestrado de Alberto Cima (CIMA, 1996) detalha muitas questões com relação a este aspecto, principalmente pesquisas relacionadas à área de análise de domínio.

Consultando a literatura especializada (ARANGO, 1988), (GOMMA, 1995), (KANG et al., 1991), (GRISS et al., 1998), entre outros, não encontramos nenhuma infra-estrutura adequada para o apoio ao desenvolvimento de *frameworks*, desde a fase de análise até a fase de implementação. Assim, direcionamos nossos esforços para a especificação desta infra-estrutura, definindo modelos necessários, padronizações e ferramentas para a manipulação destes modelos (BRAGA, WERNER, 1999), (BRAGA *et al.*, 1998a), (BRAGA *et al.*, 1998b), (BRAGA *et al.*, 2000a).

O trabalho de especificação completo de uma infra-estrutura deste tipo seria muito abrangente para uma tese de doutorado. Assim, nos concentramos na especificação de um processo para a criação de componentes reutilizáveis e na especificação de mecanismos de armazenamento, busca e recuperação de componentes. A proposta original é atualmente um projeto de pesquisa de mais longo prazo, hoje denominado projeto Odyssey (WERNER *et al.*, 1999), (WERNER *et al.*, 2000).

Ao longo do nosso trabalho, pesquisamos intensamente a área de análise de domínio e suas metodologias de suporte. Verificamos que as metodologias até então disponíveis privilegiavam a captura dos conceitos comuns e variáveis do domínio e apenas disponibilizavam diretivas para a construção dos componentes reutilizáveis. Assim, vimos a necessidade da especificação de um processo de engenharia de domínio que privilegiasse o desenvolvimento de componentes em todas as fases. Surgiu, então, o Odyssey-DE (BRAGA *et al.*, 1999a), (BRAGA *et al.*, 1999b). Em relação ao suporte ao armazenamento, identificação, busca e seleção de componentes, foi especificada a ferramenta Odyssey-Search (BRAGA, 1999c), (BRAGA *et al.*, 2000 b), (BRAGA *et al.*, 2000c).

Um resultado interessante de nosso trabalho é que ele está em contínua evolução e já apresenta alguns frutos como uma dissertação de mestrado concluída este ano (MILLER, 2000), outra em fase de finalização (XAVIER, 2000) e três ainda em andamento. Todas

surgiram a partir de idéias e trabalhos levantados inicialmente nesta tese. As duas primeiras são trabalhos específicos na área de engenharia de software, sendo uma relacionada ao desenvolvimento de aplicação com reutilização (MILLER, 2000) e outro na área de arquiteturas de software (XAVIER, 2000). As três dissertações em andamento são dois trabalhos específicos na área de banco de dados, um na área de mediadores (PINHEIRO, 2000) e outro na área de metadados (OLIVEIRA, 2000), e um trabalho relacionado a agentes de filtragem de informação (COSTA, 2000).

1.7 Resultados

Acreditamos que este trabalho contribui no sentido de darmos mais um passo em direção à efetivação da reutilização no desenvolvimento de software, a partir dos seguintes aspectos:

- Definição de um processo para a criação de componentes reutilizáveis em seus diversos níveis de abstração e capturando as ligações semânticas entre os componentes criados em um nível com os criados em níveis inferiores a partir dos primeiros, utilizando o modelo de características que é capaz de modelar as abstrações que facilitam a reutilização;
- Especificação da arquitetura da infra-estrutura de apoio ao processo de criação de componentes reutilizáveis;
- Especificação e implementação de mecanismos para o armazenamento, identificação, seleção e busca por componentes reutilizáveis, de modo a auxiliar de maneira adequada o desenvolvedor a encontrar a informação desejada, utilizando para isso abstrações e notações familiares. No contexto desta infra-estrutura, podemos ressaltar o desenvolvimento dos seguintes serviços:
  - Mecanismo que permite o armazenamento dos componentes reutilizáveis em seus diversos níveis de abstração, além das ligações de "rastro" entre os diversos níveis;
  - Mecanismo que permite a busca por componentes reutilizáveis distribuídos e heterogêneos através de consultas às abstrações definidas no modelo de domínio;
  - Mecanismo que permite a busca de componentes reutilizáveis de outros domínios de aplicação que sejam adequados ao desenvolvimento de uma dada aplicação, sem que o usuário sequer conheça este domínio, uma vez que esta necessidade é capturada pela infra-estrutura de busca.

É importante ressaltar ainda que todos os aspectos relacionados ao armazenamento, busca e recuperação de componentes na infra-estrutura Odyssey foram especificados e implementados no contexto desta tese, seja através de pacotes específicos da infra-estrutura Odyssey, tais como os pacotes Goa, Agente (especificado no pacote Ferramentas) e Rule, que foram implementados em Java (JAVASUN, 2000), seja através da implementação da arquitetura de mediação, que foi implementada em C++ (INPRISE,2000).

Até o momento, não encontramos na literatura nenhum trabalho que reúna estes aspectos de forma integrada, facilitando assim o processo de reutilização de componentes de software. Contribuições pontuais permeiam a tese e serão apresentadas ao longo dos capítulos e sintetizadas nas conclusões.

1.8 Organização da Tese

Esta tese é composta de seis capítulos, além desta introdução.

No capítulo 2 são apresentados os principais aspectos envolvidos na engenharia de domínio e no desenvolvimento baseado em componentes. Além disso, é feita uma comparação entre as duas abordagens com o objetivo de identificar aspectos comuns entre elas e pontos onde se complementam.

O capítulo 3 enfoca as técnicas utilizadas na especificação de agentes inteligentes. São apresentados os tipos de agentes inteligentes atualmente considerados na literatura, detalhando principalmente agentes de interface, filtragem e recuperação, os quais são diretamente relacionados às propostas desta tese.

O capítulo 4 apresenta o processo de engenharia de domínio Odyssey-DE, detalhando suas etapas e atividades envolvidas. Além disso, é descrita a infra-estrutura Odyssey, que diz respeito ao ferramental utilizado na automatização do processo Odyssey-DE.

O capítulo 5 descreve a ferramenta de armazenamento, busca e recuperação de informações do domínio, Odyssey-Search. A ferramenta Odyssey-Search é um sistema multi-agente especificado no contexto da infra-estrutura Odyssey. São detalhados sua arquitetura, tipos de agentes utilizados e técnicas empregadas. Além disso, detalhamos a estrutura de armazenamento utilizada pela infra-estrutura Odyssey de forma geral e mais especificamente pela Odyssey-Search, para a recuperação de componentes reutilizáveis.

Esta estrutura de armazenamento é baseada na tecnologia de mediadores (WIEDERHOLD, 1995).

No capítulo 6 apresentamos dois estudos de caso, com o objetivo de verificar a adequabilidade das propostas apresentadas nos capítulos 4 e 5.

Finalmente, o capítulo 7 apresenta as conclusões e contribuições deste trabalho, bem como as perspectivas para pesquisas futuras.

2 Desenvolvimento Baseado em Componentes e Engenharia de Domínio

2.1 Introdução

Segundo PRIETO (1991), um dos maiores problemas na reutilização de software é a criação de componentes que possam ser reutilizados em outras aplicações além daquelas para as quais eles foram originalmente definidos. Como solução para este problema, Prieto aponta como passo fundamental a sistematização do processo de criação de componentes em um dado domínio, através de uma atividade denominada análise de domínio.

O termo Análise de Domínio foi introduzido pela primeira vez por NEIGHBOURS na década de 80 (1981). Sua definição clássica apresenta esta atividade como sendo "a de identificação de objetos e operações de uma classe de sistemas similares em um domínio de problema particular". Segundo este autor, o principal produto desta atividade é a definição de uma linguagem específica do domínio, formada por uma coleção de regras que relacionam objetos e funções.

Na interpretação dada por PRIETO (1991), a análise de domínio é uma atividade anterior à análise de sistemas e cujos resultados (modelos de domínio) suportam a análise de sistemas da mesma forma que os resultados da fase de análise de sistemas suportam a fase de projeto. Entretanto, podemos considerar a análise de domínio como apenas uma das atividades envolvidas na disponibilização de componentes de software realmente reutilizáveis.

Segundo KRUEGER (1992), componentes de software podem ser desde componentes em código fonte ou binário, passando por estruturas de projeto, especificações e documentação, entre outros. Assim, para que a reutilização possa ser realmente efetiva, deve-se considerá-la em todas as fases do desenvolvimento de aplicações, desde a fase de análise até a implementação. Desta forma, deve-se disponibilizar componentes reutilizáveis para todas as fases do processo de desenvolvimento das aplicações e, mais importante ainda, que estes componentes reutilizáveis sejam consistentes ao longo deste desenvolvimento, ou seja, um componente reutilizável utilizado na especificação da aplicação deve ser consistente com um componente de software a ser reutilizado na fase de implementação. Para que isso seja possível, é necessária a especificação de uma

sistemática de desenvolvimento destes componentes ao longo de todas as fases do processo. Sem esta sistematização na criação de componentes reutilizáveis, corremos o risco de disponibilizar componentes que não contemplem todos os conceitos relevantes do domínio e que não apresentem uma uniformidade de conceitos entre as diversas etapas de desenvolvimento. Esta sistemática é conseguida através da adoção de um processo de engenharia de domínio, conforme descreveremos na seção 2.2.

Segundo ARANGO (1988), a engenharia de domínio representa um enfoque mais sistematizado para a análise de domínio, sob uma perspectiva mais voltada para a construção de componentes, criando-se um processo completo para a especificação de componentes reutilizáveis (i.e., análise, projeto e implementação). As idéias apresentadas por KRUEGER (1992) são consistentes com a abordagem de Arango, uma vez que atesta a importância da abstração para facilitar a reutilização, sendo esta abstração especificada através da análise de domínio, mas também apresenta a importância da "realização" da abstração em componentes reutilizáveis que possam ser empregados no desenvolvimento de uma dada aplicação no domínio. Neste sentido, a especificação descreve o que a abstração faz, ao passo que a "realização" da abstração descreve "como" isso é feito.

Por outro lado, a corrente demanda por software cada vez mais complexo requer uma mudança na abordagem de desenvolvimento de software atual (COHEN, 1998). O desenvolvimento de software requer cada vez mais um alto grau de automação de suas atividades e controle do processo; a utilização de novas tecnologias como distribuição e componentização; a disponibilização de produtos de software que estejam em conformidade com os padrões adotados pelo mercado, citando apenas alguns aspectos. Com base nestas novas tecnologias, novos métodos de desenvolvimento de software baseado em componentes surgiram recentemente (JACOBSON *et al.*, 1997), (D'SOUZA, 1998). No entanto, este métodos preocupam-se com o desenvolvimento de aplicações baseadas em componentes e não com o desenvolvimento dos componentes para serem reutilizados. Esta constatação também é apresentada em (FORSELL *et al.*, 2000), que apresenta a análise de três métodos considerados baseados em componentes. Forsell chega a conclusão que, na verdade, os métodos iniciam com uma abordagem de desenvolvimento de componentes, que poderia ser considerada uma abordagem de desenvolvimento *para* reutilização, ou mais especificamente um processo de engenharia de domínio mas no meio

do processo há uma mudança para o desenvolvimento *com* reutilização, ou seja, uma abordagem de engenharia de aplicações. Assim, as abordagens acabam por mesclar os dois tipos de processos, não apresentando resultados satisfatórios em nenhum dos dois. Forsell ainda ressalta a importância de um suporte de infra-estrutura para os métodos, principalmente em termos de um repositório de componentes e técnicas de busca e recuperação dos mesmos.

Neste capítulo, apresentamos uma descrição genérica sobre a engenharia de domínio e sobre o desenvolvimento baseado em componentes (DBC) e as abordagens que dispomos atualmente na literatura. Apresentamos, ainda, alguns métodos de engenharia de domínio, pois acreditamos que os mesmos podem trazer contribuições significativas para o DBC. Através do estudo destas abordagens, supomos conseguir especificar uma abordagem híbrida, que reuna os avanços tecnológicos e considerações de projeto e implementação do DBC com as abstrações e a modelagem detalhada da engenharia de domínio.

2.2 Engenharia de Domínio (ED)

Nesta tese, estamos particularmente interessados no desenvolvimento de componentes. Desta forma, analisamos o processo de engenharia de domínio com o intuito de identificar o que o mesmo tem a contribuir para o processo de desenvolvimento de componentes no contexto de DBC.

O principal objetivo da engenharia de domínio (ED) é o desenvolvimento de componentes do domínio que possam ser usados (e reutilizados) no desenvolvimento de aplicações no domínio. Assim, a ED deve consistir de atividades que sistematizem a busca e representação de informações do domínio, de forma a facilitar ao máximo a sua reutilização.

2.2.1 Etapas da Engenharia de Domínio

Os métodos propostos na literatura (KANG *et al.*, 1998), (SIMOS, 1996), (KANG *et al.*, 1990), (CHAN *et al.*, 1998), (GRISS *et al.*, 1998) concordam que existem basicamente três etapas principais na ED[1]:

- Análise do domínio, que determina os requisitos comuns de uma família de aplicações com o objetivo de identificar as oportunidades de reutilização.

- Projeto do domínio: Utiliza os resultados da análise do domínio para identificar e generalizar soluções para os requisitos comuns, através da especificação de uma arquitetura de software do domínio. As oportunidades de reutilização identificadas na análise do domínio são refinadas de forma a especificar as restrições do projeto.
- Implementação do domínio: Transforma as oportunidades de reutilização e soluções do projeto para um modelo implementacional, que inclui serviços tais como: a identificação, reengenharia e/ou construção, e manutenção de componentes reutilizáveis que suportem estes requisitos e soluções de projeto.

2.2.2 Profissionais envolvidos com a Engenharia de Domínio

Em (SIMOS, 1996), são apresentados três grupos principais de profissionais que atuam ativamente em um processo de ED:

- Fontes: são os usuários finais que utilizam aplicações já desenvolvidas naquele domínio e os especialistas do domínio que provêem informações a respeito de conceitos e funcionalidades relevantes no domínio;
- Produtores: São os analistas e projetistas do domínio que capturam as informações das fontes e de aplicações existentes e realizam a análise, projeto e implementação/empacotamento dos componentes reutilizáveis;
- Consumidores: são os desenvolvedores de aplicações e usuários finais interessados no entendimento do domínio, que utilizam os modelos gerados nas diversas etapas da engenharia de domínio para a especificação de aplicações e/ou maior entendimento dos conceitos e funções inerentes ao domínio.

2.2.3 Fontes de Informação

Existem diversas fontes de informação disponíveis para a ED, cada uma com vantagens e desvantagens. A tabela 2.1, adaptada de (KANG, 1990), apresenta os principais tipos de fonte de informação disponíveis para a ED.

---

[1] Para o leitor não familiarizado com os conceitos inerentes a Engenharia de Domínio, no capítulo 4

Fonte	Vantagem	Desvantagem	Etapas da ED que a utiliza
Livros	• **Fonte segura de conhecimento teórico do domínio**	• **Apresentam uma visão ideal do domínio**	• **Bom para a construção do modelo de abstrações do domínio.** • **Fase de análise e de projeto**
Padronizações	• **Representa uma visão de consenso do domínio.** • **Geralmente utiliza uma nomenclatura amplamente aceita no domínio**	• **Deve ser validada para verificar sua aceitação no domínio**	• **Análise do domínio**
Aplicações existentes	• **A modelagem, se existir, é bem realista** • **Pode ser diretamente utilizada para se determinar os requisitos do usuário** • **A arquitetura pode ser abstraída para a arquitetura do domínio, se houver características arquiteturais comuns entre as aplicações.** • **Pode permitir a generalização e empacotamento de componentes implementacionais**	• **Se o domínio é dinâmico, as aplicações podem não representar mais a realidade do domínio.**	• **Todas as etapas**
Especialistas do Domínio	• **Permitem a captura mais consistente da dinâmica e da evolução do domínio**	• **Podem refletir uma visão muito particular do domínio**	• **Principalmente na fase de análise**

Tabela 2.1 – Principais Fontes de informação para ED

2.2.4 Produtos gerados pela Engenharia de Domínio

Um método de engenharia de domínio para ser efetivo deve prover representações específicas para documentar os resultados de cada uma das fases da ED. Estas representações devem, prioritariamente, representar o escopo/abrangência do domínio,

---

apresentamos uma descrição mais detalhada de um processo de ED.

descrever os problemas passíveis de serem solucionados através de componentes de software no domínio e prover especificações arquiteturais e implementações que solucionem os problemas encontrados no domínio. Assim, para cada uma das fases da engenharia de domínio, devem existir produtos específicos a serem disponibilizados.

Na fase de análise de domínio, devem ser disponibilizadas representações que capturem o contexto e a abrangência do domínio, explicitando seu interfaceamento com outros domínios. Nesta fase, ainda devem ser disponibilizados modelos que capturem os principais conceitos e funcionalidades inerentes ao domínio, gerando assim um ou mais modelos com abstrações do domínio (como recomenda KRUEGER (1992)). Este modelo de abstrações deve ser acompanhado de uma documentação que detalhe cada uma das abstrações e como estas podem ser mapeadas em requisitos de aplicações no domínio.

O projeto do domínio deve disponibilizar modelos que especifiquem a estrutura arquitetural a ser seguida pelas aplicações do domínio. As representações geradas devem prover modelos arquiteturais que auxiliem na especificação de arquiteturas específicas para cada aplicação. Pode-se ter vários modelos arquiteturais para um único domínio.

A fase de implementação do domínio deve disponibilizar componentes implementacionais que especifiquem as principais funcionalidades encontradas em aplicações do domínio. Estes componentes implementacionais devem estar em conformidade com o modelo de abstrações da fase de análise e com os modelos arquiteturais da fase de projeto, de forma que possam cooperar entre si para implementar todas as funcionalidades requeridas para uma dada aplicação.

2.3 Desenvolvimento baseado em Componentes (DBC)

Até bem pouco tempo atrás, o desenvolvimento da maioria dos produtos de software disponíveis no mercado era baseado em uma abordagem de desenvolvimento em blocos monolíticos, formados por um grande número de partes interrelacionadas, onde estes relacionamentos estavam na maioria das vezes implícitos. O desenvolvimento baseado em componentes surgiu como uma nova perspectiva para o desenvolvimento de software, cujo objetivo é a "quebra" destes blocos monolíticos em componentes interoperáveis, reduzindo desta forma a complexidade no desenvolvimento, assim como os custos, através da utilização de componentes que, a princípio, seriam adequados para serem utilizados em outras aplicações (SAMETINGER, 1997).

O desenvolvimento de software baseado em componentes tem como principal objetivo o desenvolvimento de aplicações pela composição de partes já existentes. Assim, o DBC é caracterizado por um conjunto de elementos, i.e., desenvolvimento de componentes (partes) independentes, composição de partes, interoperação[2], que o distinguem do desenvolvimento de software tradicional. Os componentes são partes operacionais de software que foram desenvolvidas de forma a desempenhar completamente suas funções. Assim, os componentes são empacotados a fim de prover conjuntos de serviços que são acessíveis apenas através de uma interface bem definida.

Este objetivo não é novo no desenvolvimento de software. Em 1968, McIlRoy (MCILROY, 1968 em SAMETINGER, 1997) já vislumbrava uma indústria de componentes de software reutilizáveis. Segundo HALL (2000), desde as primeiras linguagens de interconexão de módulos, em 1976, a idéia de arquitetura de software baseada em componentes já surgia. A diferença é que estas abordagens se baseavam essencialmente em código. Entretanto, vemos que o DBC para ser efetivo envolve muito mais do que isso (SAMETINGER, 1997). Somente recentemente o DBC está sendo considerado como uma técnica importante no desenvolvimento de software. Um conjunto de fatores despertaram um novo interesse em DBC, provendo a motivação necessária para se acreditar que este possa ser agora mais efetivo e realizado em larga escala. Dentre estes fatores, podemos citar (SAMETINGER, 1997), (HALL, 2000), (ATKISON *et al.*, 2000):

- Desenvolvimento da WWW e da Internet aumentaram o entendimento e preocupação em relação à computação distribuída. A WWW encorajou os usuários a considerarem um conjunto de serviços distribuídos no hiperespaço como sendo na realidade um sistema distribuído.
- A mudança de sistemas baseados em mainframes para sistemas baseados na arquitetura cliente/servidor levou os desenvolvedores a considerarem as aplicações não mais como sistemas monolíticos, mas sim como um conjunto de sub-sistemas interoperáveis.
- Padrões de infra-estrutura para construção de aplicações como as iniciativas do "Object Management Group" (OMG), através da "Object Management Architecture" (OMA) e

[2] Interoperação é a capacidade de dois ou mais componentes de software se comunicarem ou trabalharem em conjunto, independentemente da linguagem de programação em que foram escritos ou de seus domínios de execução.

dos produtos da Microsoft como o "Component Object Model" (COM) e seus derivados, entre outros.

O "Gartner Group" estima que por volta de 2003, 70% de novas aplicações desenvolvidas sejam especificadas como uma combinação de componentes reutilizáveis integrados. Esta reutilização em larga escala pode aumentar de maneira decisiva a qualidade, custos e tempo de entrega (ATIKSON *et al.*, 2000). No entanto, para que as previsões do "Gartner Group" possam ser cumpridas, ainda existe uma série de problemas a serem resolvidos em DBC, desde uma definição mais precisa e amplamente aceitável do que seria um componente, passando pelos problemas de interconexão de componentes em uma aplicação, interoperabilidade entre componentes e metodologias para o desenvolvimento baseado em componentes. Pelo que se pode notar nas discussões em recentes "workshops" sobre DBC (CBSE, 1998, 1999, 2000), ainda não se tem um consenso em nenhum destes tópicos, o que mostra a necessidade de um maior amadurecimento da área nos próximos anos para que em 2003 as previsões possam se concretizar.

Nas subseções a seguir, apresentamos uma visão geral do estado da arte dos diversos tópicos envolvidos no desenvolvimento baseado em componentes, apresentando ainda os pontos onde uma pesquisa mais intensa se faz necessária.

2.3.1 Componentes

A caracterização do que seria um componente em DBC ainda não é um tópico fechado. Cada grupo de pesquisa caracteriza da maneira mais adequada ao seu contexto o que seria um componente e, assim, não há ainda na literatura uma definição comum para este termo.

Desde o primeiro workshop de DBC em 1998, em Kyoto (BROWN e WALLNAU, 1998), até o mais recente, em Los Angeles, Estados Unidos (CBSE, 2000), passando por outras conferências mais específicas (SSR e ICSR), várias definições têm sido apresentadas. Cada uma delas ressalta um determinado aspecto de um componente. O que podemos notar é que as visões naquela época eram ainda muito diferentes umas das outras, o que ressaltava a novidade da área. Atualmente, podemos observar que já existem definições mais precisas e convergentes a respeito do que vem a ser um componente. A

definição de Sametinger (SAMETINGER, 1997), com algumas considerações em relação à arquitetura de software feita por Kazan (KAZAN, 2000), seria abrangente o suficiente para estabelecer uma definição satisfatória do que seria um componente em DBC:

"Componentes reutilizáveis são artefatos autocontidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras em conformidade com um dado modelo de arquitetura de software, documentação apropriada e um grau de reutilização definido."

Assim, já há um certo consenso de que um componente de software deve:

1. ser autocontido em termos da funcionalidade que ele provê.
2. descrever ou realizar uma função específica (KOZACZYNSKI, 1999) (SAMETINGER, 1997), (YACOUB *et al.*, 1999b);
3. estar em conformidade e prover um conjunto de interfaces bem definidas (CBSE, 1998, 1999, 2000) (SAMETINGER, 1997), (SZYPERSKI, 1998);
4. estar inserido no contexto de um modelo que guie a composição deste componente (arquitetura de software);
5. ter disponível uma documentação adequada (YACOUB *et al.*, 1999a), (SAMETINGER, 1997);
6. prover o grau de maturidade do componente, ou seja, em quantos projetos foi reutilizado, opiniões a respeito do componente, etc. (SAMETINGER, 1997), (YACOUB *et al.*, 1999b);

Ser autocontido significa que a função que o componente desempenha deve ser realizada por ele, de forma completa. Um componente deve também ser claramente identificável, de forma que possa ser encontrado de maneira facilitada seja qual for sua localização (remota ou não). Atualmente, este é um atributo muito importante pois cada vez mais a distribuição e interoperabilidade são características desejadas pelas aplicações. Assim, em um sistema distribuído, é importante que um componente seja identificado de maneira única, sem ambigüidades.

Um dado componente não é completamente independente dos outros componentes e do ambiente. O que determina como se dá esta dependência do componente em relação aos seus pares e ao ambiente que o cerca são suas interfaces. As interfaces de um dado componente determinam como este componente pode ser reutilizado e interconectado com outros componentes. Assim, as interfaces são cruciais para o mecanismo de composição, onde um componente exporta (provê) serviços que outro componente importa (requer). Dois componentes são conectados, portanto, se os requisitos requeridos por um componente são providos pelo outro componente.

SZYPERSKI (1998) define uma interface como sendo um conjunto de assinaturas de operações que podem ser invocadas por um cliente. Para cada operação, sua semântica deve ser especificada e esta especificação tem um papel duplo, que é servir tanto para os servidores que implementam a interface quanto para os clientes que irão usar esta interface. Esta abordagem é crucial para o DBC, pois a priori, servidores e clientes não precisam se conhecer. O que "mediará" a conexão será a interface que permite que as duas partes possam trabalhar em conjunto.

BROWN *et al.* (1998) apresentam como sendo importantes dois tipos de interface: interface funcional, que reflete o papel do componente no contexto de uma dada aplicação, e uma outra interface, denominada extra-funcional, que descreve restrições impostas pela arquitetura de software. Esta visão evoluiu no contexto da comunidade de DBC. Em (YACOUB, *et al.*, 1999a) é apresentado um modelo para a classificação de interfaces bastante interessante. Os autores dividem as interfaces em dois tipos: aplicação e plataforma. As interfaces de aplicação definem a interação do componente com outros componentes, de diferentes granularidades. HALL (2000) ressalta, ainda, que existem dois subtipos de interface de aplicação: interna, que são interfaces que permitem a conexão de componentes para formar um componente de maior granularidade, e interface externa que é visível em relação à arquitetura de software de uma dada aplicação. Neste sentido, Hall ressalta que a construção de componentes é um processo recursivo, com a composição de componentes de diversas granularidades. Interfaces de plataforma definem a interação com a plataforma onde cada componente pode ser executado. Este tipo de interface pode incluir chamadas ao sistema operacional, a tecnologia de hardware requerida para a execução do componente e os subsistemas de comunicação. Os autores ressaltam a importância deste

tipo de interface uma vez que determina a portabilidade do componente e como ele roda e executa em um hardware específico.

Outro ponto ressaltado pelos pesquisadores na área (SAMETINGER, 1997), (HALL, 2000) é a necessidade de se ter em alguns casos uma outra estrutura entre as interfaces para mediar a conexão. Esta estrutura é responsável por realizar conversões simples. Como exemplo deste tipo de estrutura podemos citar o barramento CORBA. SZYPERSKI (1998) apresenta ainda a importância das pré e pós condições para que dois componentes possam se conectar através de uma interface. Neste sentido, as interfaces podem ser vistas como contratos entre o cliente (requisitor) da interface e o provedor da implementação da mesma. Assim, para que o contrato possa ser fechado, ambas as partes têm que cumprir condições especificadas nas pré e pós condições da interface.

A documentação é também indispensável para a reutilização (SAMETINGER, 1997). Esta deve ser suficiente para que se possa recuperar um componente, avaliar sua adequabilidade para o contexto da reutilização, fazer adaptações (se este for o caso) e integrar o componente no seu novo ambiente. Outro conceito importante é o que os pesquisadores chamam de "grau de reutilização" do componente (YACOUB *et al.*, 1999a) (SAMETINGER, 1997). O grau de reutilização contém informações tais como: quantas vezes e onde o componente foi reutilizado (demonstrando assim sua maturidade ou não), quem é responsável pela manutenção do componente, quem é o proprietário do componente, entre outras.

Atualmente, uma das grandes preocupações no DBC é como especificar componentes de forma que os mesmos possam ser disponibilizados de maneira adequada para um grande número de usuários. Uma das pesquisas promissoras neste sentido é em relação ao uso de XML (*eXtended Markup Language*) para a publicação de componentes (LARSEN, 2000). O XML está sendo usado principalmente para a representação das interfaces dos componentes e com isso os pesquisadores acreditam que o problema de conexão de componentes heterogêneos poderia ser melhorado. Os pesquisadores argumentam que o uso de um padrão amplamente aceito pela comunidade garante a troca de informações de maneira padronizada e que o XML pode ser um mecanismo de representação a ser utilizado como padrão. Outro ponto onde as pesquisas relacionadas ao XML se mostram promissoras é em relação à documentação de componentes de forma que os mesmos

possam ser consultados por mecanismos de busca de forma mais eficiente (FAYAD *et al.*, 2000). Neste sentido, a publicação dos componentes seria feita através de uma descrição em XML. A utilização de XML em DBC é bastante promissora e este parece ser uma caminho de pesquisa forte em DBC nos próximos anos.

Sametinger (SAMETINGER, 1997) define alguns atributos que um componente deve ter de forma que sua classificação, busca e seleção sejam bem caracterizadas. Estes atributos complementam as características descritas acima, descrevendo com mais detalhes algumas delas. São eles:

- funcionalidade: a funcionalidade de um dado componente deve ser bem caracterizada, de forma que possa ser facilmente identificável em relação a sua utilidade em um dado contexto;
- interatividade: o quanto o ambiente e os outros componentes afetam o componente e a forma como ele se comporta;
- interação: como se dá a comunicação com outros componentes;
- concorrência: define entre outras coisas a ordem de execução dos componentes, se dois componentes podem executar concomitantemente, etc. O estudo da concorrência entre os componentes pode levar a ganhos de velocidade, entre outros aspectos;
- distribuição: principalmente com o advento da Internet, este atributo passou a ter grande importância no contexto de DBC;
- formas de adaptação: podemos ter duas formas de adaptação, a customização, que é a possibilidade de parametrização do componente, e a modificação que é a criação de uma nova versão do componente;
- controle de qualidade: deve-se prover mecanismos para garantir a qualidade de componentes .

Outra questão interessante é que, no início, a visão de um componente como um elemento somente como código era bastante forte e hoje (GOMMA, 2000) (GRISS, 2000), (FORSELL *et al.*, 2000) (HALL, 2000) esta visão está mudando e as pessoas começam a visualizar componentes em todos os níveis de abstração do processo de desenvolvimento. Atualmente, há um consenso em relação a dois grupos principais de componentes.

Szyperski (SZYPERSKI, 1998), WALLNAU (CBSE, 1999) classificam os componentes em:

- componentes de negócio, que são componentes que o domínio em si consegue reconhecer, tais como cliente, contas, etc;
- componentes de infra-estrutura, que são componentes de suporte aos componentes de negócio, tais como segurança, auditoria, tratador de mensagens de erro, etc.

Existe também um consenso em relação à importância dos componentes de negócio, apesar dos mesmos ainda não serem tão bem explorados. Esta abordagem se deve ao fato de que os pesquisadores na área concordam que o que irá dar retorno em termos de investimento para as empresas são os componentes que descrevem as funcionalidades do negócio da empresa.

2.3.2 DBC e a Orientação a Objetos

Um grande foco de debate ainda hoje em relação a DBC é a sua relação com a orientação a objetos (OO). Do ponto de vista do desenvolvimento OO, podemos considerar o DBC como uma evolução do mesmo. Em particular, a evolução se deve, principalmente, em termos de como se dá a construção de novos artefatos. No contexto do paradigma OO, quando um novo comportamento precisa ser definido, este é geralmente criado através da reutilização, ou herança, de um comportamento já existente no sistema, que é então especializado. Em muitas situações esta abordagem é benéfica. No entanto, podemos observar que a mesma tem algumas limitações. Por exemplo, para sistemas complexos, as hierarquias de comportamento herdado podem se tornar estruturas pesadas, difíceis de serem entendidas e cheias de interdependências que as tornam difíceis de serem modificadas. Assim, podemos considerar o DBC como uma evolução da OO no sentido de pregar construções onde as hierarquias de comportamento somente sejam permitidas no contexto de um componente. Este tipo de dependência não deve existir entre componentes, enfatizando-se neste caso o uso de agregações ao invés da herança.

A tecnologia de objetos é bastante adequada para DBC, uma vez que os conceitos importantes em DBC já são inerentes ao desenvolvimento OO, tais como encapsulamento e polimorfismo. Um dos pontos conflitantes entre as duas tecnologias é justamente o uso ou não da herança. DBC, em geral, prega o não uso da herança da forma que ela é usada em

OO. De acordo com HEINEMAN (1999), existem dois tipos de herança: essencial, que é a herança de um comportamento ou de uma característica externa visível, ou a herança acidental, que é a herança de parte ou toda a implementação de um objeto mais genérico. No desenvolvimento OO, a herança acidental, feita estritamente para herança de código, leva a um projeto pobre. Outro problema, ressaltado por Heineman e também por SZYPERSKI (1998), é que a utilização indiscriminada do mecanismo de herança leva a uma hierarquia de classes muitas vezes complexa, o que dificulta seu entendimento. Assim, se for necessária alguma modificação, o engenheiro de software tem que entender toda a hierarquia. Como a dificuldade deste entendimento é inerente, muitas vezes ao invés de modificar uma dada classe prefere-se adicionar uma nova classe, o que leva a herança acidental e ao projeto pobre.

No caso de DBC, isso é ainda mais grave, pois um dado componente deve ser conhecido apenas pelas suas interfaces externas. Assim, qualquer adaptação necessária deve ser feita idealmente através destas interfaces. Se houver a necessidade de utilizar herança será a do tipo essencial, ou seja, herança de interfaces.

Para tentar superar estas dificuldades, o DBC prega que o comportamento e as interfaces dos componentes sejam claramente especificados, sem ter que se preocupar com estruturas herdadas e possíveis modificações na mesma. Além disso, são providos mecanismos que permitem a conexão dos componentes de forma flexível, sendo que um componente pode ser substituído por outro similar a qualquer momento, sem haver a necessidade de modificações internas na estrutura dos componentes relacionados.

Apesar do DBC poder ser considerado uma evolução do desenvolvimento OO, isso não quer dizer que um componente tenha que necessariamente ser desenvolvido a partir da tecnologia OO. Um componente pode conter procedimentos tradicionais, ou pode ser desenvolvido inteiramente utilizando um abordagem de desenvolvimento estruturado, ou outra qualquer. O que importa para que o mesmo seja caracterizado como sendo um componente é que esteja em conformidade com a definição de componentes em relação as suas interfaces, arquitetura e funcionalidade, para citar somente os principais aspectos (SZYPERSKI, 1998).

2.3.3 Distribuição e Heterogeneidade

Devido, principalmente, ao advento da Internet e da mudança da arquitetura cliente/servidor para a idéia de pequenas aplicações que se comunicam pela WEB, formando uma aplicação distribuída, a importância da distribuição e interoperabilidade de componentes heterogêneos se disseminou pela comunidade de desenvolvedores de software. Esta abordagem é conhecida como *componentware* (SANT'ANNA, LEITE e PRADO, 1998) e as previsões apontam (KAPLAN, 1999) que este vai ser o grande mercado nos próximos anos no que tange o desenvolvimento de software. Com esta disseminação da abordagem de *componentware* criou-se uma certa confusão em relação ao que seria efetivamente o DBC. Muitos acreditam que o DBC seria puramente uma abordagem de *componentware*.

O *componentware* tem como objetivo o inter-relacionamento de componentes através de modelos de objetos ou similares. Neste contexto, modelos de objetos como o COM da Microsoft (Microsoft, 2000) e o CORBA da OMG (OMG, 2000) são utilizados, descrevendo como os objetos se comunicam uns com os outros, não se preocupando com a linguagem de programação, espaço de endereçamento, máquina ou sistema operacional utilizados. Modelos de documentos como o OLE 2 da Microsoft e OpenDoc do CILabs também são usados nesta abordagem. Como exemplo de utilização desta abordagem temos o desenvolvimento de componentes utilizando tecnologias como ActiveX, JavaBeans, como é o caso do projeto San Francisco, da IBM (IBM, 2000). A interoperabilidade em um ambiente distribuído é outra característica marcante da abordagem, permitindo componentes desenvolvidos em diferentes plataformas de hardware, sistemas operacionais e linguagens de programação se comunicarem através de interfaces bem definidas. Outro ponto importante é que apenas componentes binários são considerados, tais como componentes DCOM, JavaBeans e CORBA (PARISH *et al.*, 1999).

Apesar destas características serem importantes e desejáveis em um componente, sua importância se deve, principalmente, pelo apelo do mercado por produtos na forma de código, ou seja, um componente que possa ser integrado ao código de uma dada aplicação e execute de forma adequada. Desta forma, em uma abordagem de desenvolvimento de componentes, este seria o produto final do processo, onde este componente começou a ser delineado através da especificação de uma dada funcionalidade importante em um domínio

de aplicação. Somente desta forma, é possível inferir que o componente será codificado (espaço da solução) de acordo com as necessidades do domínio (espaço do problema). A abordagem de *componentware* enfatiza o produto final sem se preocupar com o processo que foi utilizado para produzi-lo.

A utilização da abordagem de *componentware* pura para DBC esbarra ainda em alguns problemas como a falta de orientação em relação à escolha dos componentes mais adequados a um dado contexto e a melhor maneira destes componentes se relacionarem. Muitas vezes, a forma de composição escolhida pelo desenvolvedor não é a mais adequada, e nem os componentes escolhidos são os que melhor atendem às necessidades da aplicação, levando-se em consideração as características do ambiente e do domínio onde serão utilizados. Um forma de atenuar estes problemas é a especificação de uma arquitetura que apresente a interação entre os componentes a serem utilizados em um dado contexto, além de um maior detalhamento das funcionalidades dos componentes. Por outro lado, a abordagem de *componentware* tem uma característica que a torna primordial no processo de desenvolvimento atual, que é a facilidade de permitir que componentes heterogêneos sejam agregados.

Assim, devemos considerar o DBC em um contexto mais amplo, sendo o *componentware* apenas uma das tecnologias envolvidas. A fim de que uma abordagem consistente de DBC tenha sucesso, é primordial o entendimento dos conceitos que estão por trás das funcionalidades disponibilizadas pelos componentes e seus relacionamentos; a concretização do conhecimento em componentes reutilizáveis e o uso de um modelo que apresente o interfaceamento entre os componentes (modelo arquitetural).

Entre os padrões para interoperabilidade que surgiram no mercado nos últimos anos, ganharam maior destaque CORBA (OMG, 2000), DCOM (Microsoft, 2000) e RMI (JAVASUN, 2000). Como apelo para sua utilização, estes padrões propõem um alto nível de abstração para viabilizar a interoperação dos componentes através da utilização de linguagens intermediárias, geradores de código e linguagens para especificação de interfaces.

A utilização destes padrões realmente facilita a interoperação dos componentes. No entanto, resolve apenas um problema pontual na especificação de uma arquitetura de software para DBC, e mesmo assim, conforme KAPLAN *et al.* (1999) ressaltam, muitas

vezes trazem outros problemas, tais como grande acoplamento da aplicação com o mecanismo de interoperabilidade escolhido e não resolvem outros de maneira adequada, como somente lidar com componentes binários e a especificação da interface somente se preocupar com a sintaxe das operações.

Assim, apesar destes padrões de interoperabilidade serem uma tecnologia que facilita a interoperação dos componentes, esta por si só não pode ser considerada uma abordagem para DBC. Ressaltando este argumento podemos observar que os mecanismos de interoperação lidam somente com componentes binários encapsulados em um invólucro que possui as diretivas do mecanismo de interoperação. HAN (2000) argumenta que a interoperação dos componentes é dificultada também pela não compreensão das capacidades dos componentes. Como os mecanismos de interoperação consideram apenas o componente na sua forma binária, as capacidades ficam explicitadas nas descrições de suas interfaces. As abordagens atuais de definição de interfaces tratam somente questões sintáticas, como é o caso da IDL de CORBA. O autor argumenta que para se compreender de forma mais adequada um dado componente, dever-se-ia ter mais informações sobre o componente, tais como: a semântica dos elementos da interface; seu relacionamento, o contexto de uso e os atributos de qualidade. Uma abordagem interessante é proposta pelo autor, onde as assinaturas das operações são disponibilizadas pelo componente, incluindo também a semântica de interação de cada elemento da assinatura e restrições adicionais em relação ao uso destas assinaturas.

2.3.4 Arquitetura de Software para conexão de componentes

Um dos pontos em que existe consenso entre os pesquisadores na área de DBC é que um componente não pode ser visto de forma completamente independente de outros componentes com os quais se relaciona e de seu ambiente. Sendo assim, a arquitetura de software assume um papel de extrema importância, pois é a partir dela que podemos especificar de forma mais detalhada como se dá a interconexão entre componentes.

GARLAN *et al.* (1994) definem arquitetura de software em geral como sendo a especificação e o projeto das estruturas de controle e de organização de um dado sistema, incluindo-se os protocolos de comunicação utilizados para conexão dos elementos arquiteturais, sincronização, acesso aos dados, escalabilidade, desempenho, distribuição física e seleção entre alternativas de projeto. Neste contexto, um estilo arquitetural define

um vocabulário comum de tipos de componentes e conectores, e um conjunto de regras de como componentes e conectores podem ser combinados. É importante ressaltar que, na definição de GARLAN *et al.* (1994), os componentes e conectores arquiteturais são especificados de maneira abstrata, sem se levar em consideração o tipo de funcionalidade específica que um dado componente disponibilizaria. Exemplos de estilos arquiteturais são os estilos *pipe and filter*, *broker*, *model-view-controller*, entre outros. Desta forma, nenhum tipo de consideração em relação ao domínio de aplicação do componente é feita.

Segundo SZYPERSKI (1998), uma arquitetura de software para atender ao DBC consiste de um conjunto de decisões relacionadas à seleção de alternativas de projeto em relação à plataforma operacional, ao modelo de composição dos componentes e ao projeto de interoperação para o modelo de composição. Podemos notar, através desta definição, que o autor separa claramente a arquitetura de software para DBC em três aspectos principais:

1. A infra-estrutura para interconexão dos componentes, que na verdade nos remete à definição de arquitetura de software e estilos arquiteturais, utilizados por Mary Shaw e David Garlan (SHAW *et al.*, 1996);
2. A composição dos componentes em termos funcionais, o que alguns autores denominam de "*framework* de componentes" e outros de arquitetura do domínio (DIGRE, 1998). A idéia de composição estaria ligada à necessidade de ligação dos componentes para a especificação da funcionalidade desejada em um dado contexto;
3. A interoperação, introduzindo uma visão mais ligada à distribuição e heterogeneidade dos componentes. Neste contexto, a interoperação seria a capacidade de dois ou mais componentes de software se comunicarem ou trabalharem em conjunto, independentemente da linguagem de programação em que foram escritos ou de seus domínios de execução.

Todos estes pontos são muito importantes em DBC. No entanto, consideramos que uma arquitetura de software para DBC seria mais ligada à composição dos componentes e à infra-estrutura para esta conexão. A interoperação, item 3, trata-se de uma decisão arquitetural ligada à infra-estrutura de conexão, uma vez que para que a interoperação seja realizada, pressupõe-se a utilização de um mecanismo tal como um barramento CORBA,

RMI, entre outros. Este barramento é parte integrante da infra-estrutura para a interconexão de componentes.

A infra-estrutura necessária para a conexão dos componentes vem ao encontro das especificações de estilos arquiteturais definidas por SHAW e GARLAN (1996). Assim, esta infra-estrutura seria composta pelos serviços básicos, restrições e ligações semânticas que devem estar disponíveis em uma aplicação que siga um determinado estilo arquitetural. BUSHMMAN (1996) descreve os vários estilos arquiteturais por meio de padrões, apresentando os diversos serviços e ligações arquiteturais ligadas a um dado estilo. Por exemplo, um dos padrões apresentados por BUSHMMAN (1996, pp. 99-122) descreve uma configuração arquitetural para sistemas distribuídos. É feita toda uma discussão a respeito dos principais aspectos do padrão e, ao final, são apresentados modelos estruturais e comportamentais de como se daria a interação entre os elementos de uma aplicação que seguisse este estilo arquitetural. Assim, o que na verdade são apresentados são os serviços e a infra-estrutura necessária para que uma aplicação se comporte como um sistema distribuído. Não é feita nenhuma consideração em relação às funcionalidades que os componentes da aplicação devem disponibilizar. Este estilo arquitetural dominante é mais dependente da aplicação em si e do ambiente onde ela será executada. O que teríamos a mais em DBC seria o modelo de componentes do domínio ou frameworks de componentes que seria específico para um dado domínio. Derivar uma especificação de estilo arquitetural para um conjunto de componentes reutilizáveis de forma independente da aplicação que irá utilizá-los não é uma tarefa simples. Existem alguns domínios de aplicação mais estáveis onde se pode observar uma certa tendência em que todas as aplicações do domínio sigam um dado estilo arquitetural, como por exemplo, o domínio de telecomunicações, onde um estilo arquitetural distribuído é utilizado pela maioria das aplicações. Nestes casos, pode-se pensar em termos de um estilo arquitetural predominante, mas mesmo assim pode existir uma ou outra aplicação que não se encaixe perfeitamente no estilo arquitetural predominante.

A parte mais estável da definição dada por SZYPERSKI (1998) é a composição dos componentes em termos funcionais (item 2). Este framework de componentes é uma estrutura mais estável em relação ao domínio e não tão dependente das especificidades do ambiente em que a aplicação será desenvolvida e executada. Podemos considerar que este é

um modelo derivado a partir das especificações dos componentes, ou seja, dos modelos de análise que especificam os componentes. O que vai mudando é a forma de representação, indo de um alto nível de abstração, na fase de análise, até um detalhamento maior, nas fases de projeto e codificação. Este é o modelo que caracteriza o porquê de se reutilizar um dado componente no contexto de uma dada aplicação de um certo domínio e, mais importante ainda, como se dá a interação com os outros componentes para a especificação das funcionalidades necessárias à aplicação. Esta interação deve ser regulada por um conjunto de restrições, tais como a ordem de chamada de operações, pré e pós condições funcionais, entre outras. DIGRE (1998) considera esta parte da especificação arquitetural como a mais importante de todo o modelo de DBC.

A interoperação dos componentes, a terceira característica (item 3) da descrição de arquitetura de DBC dada por SZYPERSKI (1998), faz parte do estilo arquitetural escolhido (item 2). SAMETINGER (1997) define a interoperação de componentes como a habilidade dos componentes de se comunicar e cooperar sem levar em consideração diferenças de linguagem, plataforma de execução e interfaces. O autor ainda ressalta que a composição de componentes não implica, necessariamente, na sua interoperação. Assim, quem irá resolver os problemas de interoperabilidade será o estilo arquitetural escolhido.

Em (GARLAN *et al.*, 1994), são descritos alguns "enganos arquiteturais" (*architectural mismatches*) que podem ocorrer quando se constrói uma aplicação a partir de componentes. São descritas algumas categorias com possibilidade de ocorrer enganos:

- natureza do componente, incluindo suporte de infra-estrutura que o componente pode necessitar, modelo de controle de sincronização e modelo de dados;
- natureza dos conectores, onde problemas podem ser encontrados nos protocolos utilizados e tipos de dados a serem transmitidos pelos conectores;
- estrutura da arquitetura, que pode apresentar problemas em termos das interações entre os componentes;
- processo de construção, principalmente relacionados a problemas de heterogeneidade dos componentes.

Para tentar minimizar estes possíveis enganos, Garlan propõe algumas diretivas:

- utilizar uma descrição arquitetural padronizada e bem definida em termos de modelos e restrições;
- utilizar componentes previamente desenvolvidos para serem reutilizados, onde os pontos de conexão em suas interfaces estejam bem documentados e explicitados;
- utilizar invólucros (*wrappers)* e um software de conexão (*middleware)* sempre que problemas de compatibilidade de interfaces ocorrerem;
- tentar se basear em modelos arquiteturais de sucesso e que forem bem documentados (uso de padrões arquiteturais).

Assim, uma arquitetura de software para DBC, caracterizada pelo estilo arquitetural e por um modelo funcional de conexão de componentes, auxilia nas questões levantadas por Garlan, elucidando a natureza do componente, através do uso do modelo funcional, a natureza dos conectores, descrevendo-se os tipos de conexão necessárias para atender às necessidades funcionais dos componentes e também ao estilo arquitetural adotado; a estrutura da arquitetura, descrita pelo estilo arquitetural e o processo de construção, que engloba o processo de construção do componente como um todo.

2.3.5 Métodos para desenvolvimento de componentes

Como todo processo de desenvolvimento de software, é preciso prover uma sistemática (i.e. métodos) para o desenvolvimento baseado em componentes, devendo ser cuidadosamente planejada e controlada para ser efetiva. Assim, é necessária uma abordagem que enfatize a reutilização em todas as etapas do desenvolvimento.

Podemos dividir o processo de desenvolvimento em duas etapas: desenvolvimento **para** reutilização e desenvolvimento **com** reutilização (KRUEGER, 1992). Colocando esta abordagem sobre o prisma do desenvolvimento baseado em componentes, mudaríamos um pouco a nomenclatura e teríamos a mesma divisão: desenvolvimento de componentes e desenvolvimento com componentes. Em (LIM, 1998), é ainda considerada a importância do apoio de uma infra-estrutura para o processo como um todo.

Conforme visto anteriormente, o desenvolvimento de componentes pode ser visto como um processo de engenharia de domínio, que seria o desenvolvimento para

reutilização seguindo a nomenclatura utilizada por ARANGO (1988). Nesta abordagem, são três as etapas envolvidas: análise de domínio, projeto de domínio e implementação. Entretanto, alguns autores (SAMETINGER, 1997), (HALLSTEINSEN e SKYLSTAD, 1999) vêem o desenvolvimento de componentes como uma etapa posterior à engenharia de domínio. Esta visão se deve, principalmente, ao fato de que estes autores consideram componentes como itens implementacionais, sendo os modelos utilizados para a construção dos componentes relacionados a código considerados como documentação.

Ao considerar componentes muito mais do que somente código, uma abordagem de desenvolvimento de componentes, pode ser vista como um processo de engenharia de domínio completo, onde a preocupação, além de ser com a disponibilização de componentes em todos os níveis de abstração, não é com uma única aplicação, mas com uma família de aplicações. Assim, o grau de reutilização de um dado componente será muito maior e seu entendimento também, uma vez que este componente será disponibilizado tanto para especificação (mais fácil de ser entendido pelo usuário) quanto para código.

Por outro lado, o desenvolvimento com componentes deve ser visto como um processo de desenvolvimento de aplicações, onde a reutilização de componentes deve estar presente em todas as fases. Também neste ponto, podemos ver muita confusão em relação a como reutilizar os componentes e muitos dos métodos considerados para DBC não promovem a reutilização de componentes em todas as etapas, considerando esta reutilização principalmente para componentes implementacionais. Conforme ressaltam FORSELL *et al.* (2000), muitas mesclam o desenvolvimento de componentes com o desenvolvimento com componentes, fazendo as duas etapas ficarem misturadas. Com isso, não se enfatiza tanto o desenvolvimento de componentes genéricos, mas sim somente o compartimentar a aplicação em partes interoperáveis, sem a preocupação inerente de se construir componentes para serem reutilizados em contextos além dos quais eles foram originalmente construídos, uma vez que os componentes são construídos especificamente para serem utilizados na aplicação. Nenhum compromisso com flexibilidade de extensão, generalidade é utilizado. Assim, um processo detalhado e bem definido melhorará substancialmente o DBC.

Para termos um método de DBC que possa ser realmente efetivo, devemos buscar inspiração nos métodos essencialmente baseados em reutilização, tanto no que diz respeito ao desenvolvimento de componentes quanto ao desenvolvimento com componentes.

Existem atualmente alguns métodos de desenvolvimento de software baseados em componentes que se mostram promissores. O primeiro deles é baseado no trabalho de Ivar Jacobson, Martin Griss e P. Jonsson, descrito em (JACOBSON *et al.*, 1997). Este trabalho apresenta o desenvolvimento de software baseado em componentes de forma genérica. O método é baseado no paradigma OO e em padrões como UML e CORBA e utiliza casos de uso como ponto de partida para a identificação dos componentes reutilizáveis. O método, seguindo a característica básica do DBC, dá maior destaque à parte comportamental dos objetos. Outro método que vem despontando no contexto de DBC é o Catalysis (D'SOUZA *et al.*, 1998). O método Catalysis também é fortemente baseado no uso de objetos, *frameworks* e padrões, além de privilegiar o desenvolvimento de aplicações distribuídas. Neste sentido, o Catalysis consegue unir as duas tecnologias atualmente consideradas no desenvolvimento baseado em componentes, ou seja, a orientação a objetos e a computação distribuída.

No entanto, ainda está por surgir um método para DBC que privilegie, além da OO e computação distribuída, técnicas que permitam o empacotamento de componentes com o objetivo específico de serem reutilizados, levando em consideração a captura de abstrações que facilitem o entendimento dos componentes, e com processos bem identificados e detalhados tanto para o desenvolvimento de componentes (desenvolvimento **para** reutilização) como para o desenvolvimento com componentes (desenvolvimento **com** reutilização).

Além dos métodos, o apoio de um ambiente de suporte automatizado também é importante para o DBC. BROWN e WALLNAU (1998) apresentam alguns elementos básicos que devem ser considerados na utilização de ferramentas em DBC. Segundo estes autores, existem 4 elementos básicos que uma ferramenta automatizada para DBC deve apresentar: um repositório de componentes, que é utilizado para armazenar os componentes a serem utilizados no processo, uma ferramenta de modelagem baseada em componentes, que permita a descrição de uma aplicação completa em termos de seus componentes e suas interações, um gerador de aplicações, para a criação de aplicações baseadas em

componentes, e uma ferramenta de busca de componentes, para a busca e seleção dos componentes de interesse.

2.3.6 Repositório de Componentes

SAMETINGER (1997) conceitua repositório de componentes como sendo uma base preparada para o armazenamento, seleção e obtenção de componentes. Apesar do uso de repositórios de componentes ser considerado uma característica operacional, consideramos que o sucesso do DBC está intrinsecamente ligado aos mecanismos para a disponibilização de componentes reutilizáveis, e a utilização de um repositório é característica básica para isso. Assim, consideramos a utilização de um repositório de componentes característica tão importante quanto a distribuição e interoperabilidade, conforme iremos apresentar nos capítulos seguintes desta tese.

Para que esta recuperação seja efetiva, SAMETINGER (1997) ressalta a importância do armazenamento de informações adicionais relativas ao componente. A chance que um desenvolvedor tem de reutilizar um dado componente, ao invés de desenvolver um novo, depende da disponibilidade do componente em um repositório, de forma que este possa ser facilmente localizado e recuperado.

O autor distingue ainda três tipos de repositórios, a saber:

- Repositórios locais: armazenam componentes de propósito geral;
- Repositórios específicos a um domínio: armazenam componentes específicos, com escopo bem definido, podendo prover componentes alternativos para as mesmas tarefas;
- Repositórios de referência: auxilia na busca por componentes em outros repositórios, funcionando como uma espécie de páginas amarelas.

De acordo com SEACORD (1999), repositórios locais e centralizados que armazenam componentes genéricos historicamente falharam, principalmente por conta de serem repositórios centralizados e inchados. Outros autores reforçam também esta posição. BANKER *et al.*, em (SAMETINGER, 1997), argumentam através de constatações na prática, que o grau de reutilização não aumenta com o aumento do número de componentes disponíveis. Assim, a solução e evolução destes repositórios locais genéricos foi organizá-los por domínios de aplicação, o que diminui o escopo dos componentes a serem

consultados em uma busca. Entretanto, melhora os resultados da busca, uma vez que os componentes são mais focados em relação à chave de busca.

No entanto, com o advento da Internet e a disponibilidade de informações distribuídas e heterogêneas, permitiu-se que se tenha acesso a repositórios distribuídos, o que aumenta a gama de componentes disponíveis. Assim, uma abordagem que os autores na área vêem como promissora é a interconexão de repositórios através de um mecanismo como os chamados repositórios de referência. O sistema Agora (SEACORD, 1999) utiliza uma abordagem similar a de repositórios de referência. No entanto, os mecanismos de busca empregados, através de introspeção de componentes de código, é limitado tanto semanticamente quanto na abrangência, uma vez que somente recupera componentes binários CORBA e JavaBeans. Os próprios autores apresentam como limitador de sua abordagem a falta de informações descritivas acerca dos componentes, ou seja, adicionar mais semântica ao componente, de forma que a busca seja mais precisa. Atributos não funcionais também devem ser levados em consideração nesta abordagem, tais como confiabilidade, desempenho, disponibilidade, etc. Em (MERAD, 1999), é apresentada uma proposta que considera estes aspectos. Outros aspectos que devem ser tratados são unicidade do componente, *copyrights*, privacidade e comércio dos componentes via Internet.

2.3.6.1 Busca e seleção dos componentes

Quando se considera um repositório de componentes, uma questão importante é como se dá a busca e seleção destes componentes armazenados no repositório. Esta busca e seleção de componentes está intrinsecamente relacionada à qualidade dos mecanismos de classificação dos mesmos. Vários métodos de classificação de componentes já foram descritos na literatura. Dentre eles podemos citar: texto livre, classificação por palavras-chave (BARROS, 1995), classificação enumerada, faceta e pares de atributos.

A classificação por texto livre é a mais simples de todas. Não requer nenhum tipo de preparação, apenas a disponibilidade de uma documentação textual. A busca por palavras-chave também é bastante simples, bastando que o desenvolvedor do componente utilize um conjunto de palavras de forma livre, ou a partir de um vocabulário controlado. A utilização de texto livre e palavras-chave permitem casamentos perfeitos, ou no máximo aproximações baseadas nos radicais das palavras. No entanto, o uso deste tipo de busca por

si só tem um alto grau de inexatidão, o que resulta na recuperação de componentes inadequados.

A criação de hierarquias de termos, denominada classificação enumerada é outra forma utilizada para classificar componentes para a busca. Sua vantagem é que é fácil de ser entendida e utilizada. Como desvantagem, SAMETINGER (1997) cita sua inflexibilidade. Novos tópicos só podem ser inseridos em níveis mais baixos da hierarquia. Outro problema é a ambigüidade, uma vez que um dado componente pode pertencer a várias categorias.

A classificação por facetas (PRIETO, 1991) é outra abordagem bastante utilizada. As facetas são baseadas no vocabulário de um domínio particular, onde são consideradas perspectivas, pontos de vista e dimensões deste domínio. Os componentes são classificados segundo o conjunto de facetas. A vantagem da utilização do mecanismo de facetas é que relações complexas podem ser criadas combinando facetas com termos e modificações no esquema podem ser feitas sem modificar as facetas já existentes e suas ligações. A classificação por pares de atributos pode ser considerada um super conjunto da classificação por facetas onde mais combinações podem ser utilizadas. Uma das desvantagens do mecanismo de facetas é que utiliza a figura do bibliotecário para a criação das facetas ao passo que deveria ser sim um especialista do domínio.

No entanto, até o momento, nenhum destes métodos se mostrou realmente eficiente para a busca e seleção de componentes. Um estudo empírico realizado por Frakes e Poulin (FRAKES, 1994) em relação aos métodos de classificação por palavras-chave, enumeração, atributos/valor e facetas mostrou que nenhum dos métodos é vantajoso em relação aos demais em termos de resultados. POULIN *et al.* (1995) mostraram ainda, através de experiências com vários mecanismos de classificação para busca por componentes, que a melhor técnica até então era a combinação dos mecanismos. No caso específico de sua experiência, que era baseada em componentes na forma de código, mecanismos de busca baseados em texto e a adoção de ordem hierárquica se mostraram mais promissores.

Assim, embora já bastante explorado este tema, ainda hoje investiga-se um mecanismo eficiente para a busca e seleção de componentes. Esta é um área de pesquisa proeminente em DBC. MILI *et al.* (1995) formalizam o problema da busca e recuperação por componentes da seguinte forma:

1. O espaço do problema;
2. O espaço do problema da forma que o desenvolvedor entende;
3. O espaço da consulta, que consiste da necessidade que o desenvolvedor conseguiu entender traduzida para uma consulta que possa ser entendida pelo sistema de busca.

Estes três pontos precisam ser o mais próximo possível um dos outros, de forma que os componentes certos sejam recuperados. Outro ponto importante é que atualmente esta busca deve ser a mais automatizada e transparente possível para o usuário, uma vez que, com a Internet, a busca por componentes se tornou uma atividade muito mais complexa devido à disponibilidade de um grande número de componentes desenvolvidos por terceiros e que podem estar distribuídos ao longo da rede. Assim, deve-se considerar uma abordagem onde se mescle técnicas consagradas de busca por componentes, como o mecanismo de facetas ou similar, busca por palavras-chave, entre outras, levando também em consideração técnicas que se mostram promissoras para busca na WEB, como técnicas de aprendizado, e a natureza distribuída dos componentes.

2.3.7 Outros tópicos de pesquisa em DBC

Existem ainda na literatura vários outros tópicos de pesquisa em DBC que estão começando a ser evidenciados. Apesar das pesquisas ainda estarem incipientes, estas se mostram bastante promissoras. Dentre eles estão a certificação da qualidade de componentes e a evolução e adaptação dos mesmos.

Certificação da qualidade de componentes

No CBSE 2000, um dos temas discutido foi a qualidade de componentes e mecanismos para se certificar esta qualidade. No entanto, ainda não se tem um consenso em relação a quais atributos de qualidade um componente deve ter. HAN (2000) define como atributos de qualidade os aspectos não funcionais de um componente, tais como segurança, performance e confiabilidade. SAMETINGER (1997) também define algumas propriedades que deveriam ser analisadas na avaliação da qualidade de um dado componente. São elas: uso de recomendações, realização de testes baseados em algum padrão de testes, clareza

conceitual (um componente deve ser claro e fácil de ser entendido), definições precisas para o grau de acoplamento e coesão.

Outro ponto importante na análise da qualidade de um dado componente é que não podemos esquecer da característica de composição de um componente. Assim, um ponto fundamental é analisar a qualidade de um componente em termos da arquitetura de software na qual ou nas quais ele está inserido. Já existem alguns trabalhos na área de análise de qualidade de arquiteturas de software como os trabalhos realizados no SEI com os métodos SAAM e sua evolução ATM (KAZAM, 2000).

SAMETINGER (1997) apresenta também uma primeira proposta para níveis de certificação de componentes. Estes níveis, segundo o autor, vão depender da natureza, freqüência de reutilização e importância do componente no contexto. Os níveis destacados por Sametinger são os seguintes:

- Nível 1: O componente é descrito por palavras-chave e é armazenado para recuperação automática. Nenhum teste é realizado e o nível de completude é desconhecido;
- Nível 2: O componente de código é compilado. Métricas são definidas e utilizadas para uma linguagem em particular;
- Nível 3: São realizados testes, dados e resultados dos testes são adicionados aos componentes;
- Nível 4: Um manual de reutilização é adicionado.

Evolução e adaptação de componentes

Outra preocupação que os pesquisadores estão levando em consideração em relação à adoção efetiva do desenvolvimento baseado em componentes, é a evolução e adaptação dos componentes a fim de que possam ser reutilizados em diferentes contextos. BRERETON (1999) ressalta que esta dificuldade é grande, principalmente, em relação à evolução do sistema, uma vez que esta responsabilidade de evolução é distribuída entre diversos autores e organizações proprietárias dos componentes.

Brereton apresenta alguns tópicos relacionados à evolução de componentes. Ele divide estes tópicos em três grandes áreas:

- Negócios: responsabilidade pela mudança (deve-se ter claramente o responsável pela mudança do componente e sua integração no sistema); risco de mudança (riscos inerentes à mudança, tais como análise do impacto de mudança de um componente em um sistema como um todo); pagamentos pela mudança; suporte a longo prazo.
- Gerenciamento: diretivas para mudança (o quanto a mudança é necessária); restrições para sistemas distribuídos (os componentes residem nos seus sites remotos e é preciso lidar com questões, tais como trocar um componente por uma nova versão sem notificar os usuários ou notificando-os; deixar os usuários escolherem se querem utilizar a versão antiga, novos usuários só utilizam a versão nova); documentação e descrição de componentes.
- Técnicas: atualizar um dado componente por um novo componente do mesmo fornecedor ou de fornecedores diferentes; adicionar e remover componentes; localizar, entender e avaliar componentes adicionais ou que vão ser trocados; determinar o impacto da mudança no sistema como um todo; avaliar os custos de testar novamente o sistema; avaliar o risco com a adoção de novos fornecedores.

SZYPERSKI (1998) e SAMETINGER (1997) argumentam que para minimizar estas questões de adaptação, um componente deve idealmente ser adaptado baseado apenas na sua interface externa. Assim, adaptações e evoluções necessárias seriam feitas apenas no âmbito da interface, podendo-se até criar novas versões de interfaces.

2.4 Métodos de Desenvolvimento Baseado em Componentes e de Engenharia de Domínio Existentes

Analisamos a seguir alguns métodos de ED e de DBC que se destacam na literatura. Esta análise é feita com base no *framework* de comparação utilizado no estudo de viabilidade do método FODA (KANG *et al.*, 1990), que apresenta três principais aspectos a serem considerados na avaliação de um método:

a) **processo**: como o método irá afetar uma dada organização que o adota; como é feita a gerência e a manutenção dos produtos; como é a representação do conhecimento do

domínio nos produtos e como os usuários podem aplicar efetivamente os produtos no desenvolvimento de aplicações no domínio;

b) **produto**: quais são os tipos de produtos gerados pelo método; como são representados e o quão aplicáveis estes são no desenvolvimento de aplicações no domínio;

c) **ferramentas para automatizar o processo**: disponibilidade de ferramentas automatizadas que auxiliem na utilização do método. Como estas ferramentas estão integradas, facilidade de uso das mesmas e sua robustez.

Os métodos analisados são os clássicos, tais como FODA (KANG *et al.*, 1990), ODM (SIMOS, 1996) e EDLC (GOMMA, 1995), dos quais foram derivados os principais métodos de ED disponíveis atualmente; métodos derivados diretamente do FODA, como FORM (KANG *et al.*, 1998) e FeatuRBSE (GRISS *et al.*, 1998); e outros métodos derivados de abordagens clássicas, como o OODE (CHAN *et al.*, 1998). Estes métodos possuem idéias interessantes que podem ser aplicáveis no contexto do DBC, principalmente para a fase de modelagem de domínio, servindo de guia posterior para o empacotamento dos componentes, que não são atividades bem desenvolvidas pelos métodos de DBC. Além destes, que foram desenvolvidos com o propósito de serem realmente métodos de ED, analisamos também alguns métodos de desenvolvimento de aplicações que representam abordagens de DBC, a saber, RSEB (JACOBSON *et al.*, 1997) e Catalysis (D'SOUZA *et al.*, 1998).

2.4.1 FODA (*Feature Oriented Domain Analysis*)

O método de análise de domínio FODA (KANG *et al.*, 1990) foi criado no contexto dos projetos de pesquisa do SEI (*Software Engineering Institute*), tornando-se um dos métodos de análise de domínio mais populares no início da década de 90. Os autores descrevem o FODA como um método para dar suporte à reutilização no nível arquitetural e funcional. No entanto, os estudos apresentados na literatura descrevem principalmente a modelagem de estudos de caso na fase de análise, dando ênfase à modelagem funcional, mas não apresentando com detalhes a parte arquitetural e nem as ligações entre os diversos modelos.

A abordagem adotada pelo FODA é bastante interessante do ponto de vista do DBC, uma vez que a captura das funcionalidades relevantes ao domínio auxilia na identificação

de que componentes, disponibilizando quais funcionalidades, estariam disponíveis no domínio. Outro ponto interessante, mas que não é muito ressaltado nos estudos de caso conduzidos, é a possibilidade de modelagem de características adicionais ao domínio como a modelagem dos ambientes operacionais disponíveis e as técnicas de implementação. Estas informações auxiliam o desenvolvedor na seleção das funcionalidades mais adequadas, levando-se em consideração o ambiente operacional e as questões implementacionais.

Processo	Descreve um processo de desenvolvimento de modelos para análise de domínio que pode se encaixar no processo de desenvolvimento de uma organização, uma vez que o tipo de modelo disponibilizado, (modelos hierárquicos, ER, DFD) são entendidos pela comunidade de desenvolvimento de software. No entanto, com o uso de novas tecnologias como OO, pode haver uma certa dificuldade de adaptação dos modelos do FODA (mais centrado no desenvolvimento estruturado) com os modelos das aplicações (se for utilizada uma abordagem OO). Outro ponto importante é que não existe nenhum ponto formal de validação, sendo esta validação feita de maneira ad-hoc pelos especialistas.
Produto	Os produtos disponibilizados são compatíveis com o desenvolvimento estruturado e desta forma entendidos pelos desenvolvedores. Nas etapas de especificação da arquitetura e implementação, não existe um detalhamento maior de como especificar os modelos desta fase.
Ferramental Automatizado	Não existem ferramentas específicas e integradas para o suporte ao desenvolvimento com o FODA. O conjunto de ferramentas 001(KANG *et al.*, 1991) foi utilizado em um dos estudos de caso do FODA. No entanto, o suporte dado pelo 001 é limitado, não suportando as ligações entre modelos, o desenvolvimento de aplicações no domínio e nem uma estrutura de armazenamento para os modelos.

Tabela 2.2 – Principais características do FODA

Uma crítica que pode ser feita ao método é que por ser a modelagem centrada na captura das funcionalidades do domínio, todos os modelos utilizados carregam esta característica. O método é dito ser muito preocupado com a geração de código, e não se preocupa com a devida ênfase na captura de todas as abstrações do domínio. A preocupação é somente com características funcionais das aplicações no domínio. Características ditas não codificáveis acabam por serem descartadas na modelagem do

FODA. Estas características (não codificáveis) também são importantes no contexto do domínio, principalmente para facilitar o reutilizador no entendimento dos conceitos e assim, consequentemente, facilitar o entendimento dos componentes, promovendo de forma mais eficiente a reutilização de componentes no desenvolvimento de aplicações no domínio. Além disso, o FODA não apresenta um detalhamento maior em relação à criação de componentes reutilizáveis. O método se atém à modelagem das características, mas não especifica como criar componentes.

2.4.2 ODM (*Organization Domain Modelling*)

O método de engenharia de domínio ODM (SIMOS, 1996) é, conforme ressaltado pelos próprios autores, um método altamente configurável e customizável. Podemos considerar o ODM um *framework* para a criação de métodos de engenharia de domínio particularizados, de acordo com as necessidades das organizações.

Processo	O ODM possui fases bem definidas em relação à engenharia do domínio. No entanto, por ser um processo genérico, não possui um detalhamento maior dos produtos gerados em cada uma das fases. Um aspecto interessante e que não é tratado com igual ênfase pelos outros métodos de ED é a fase de planejamento, onde é proposto um estudo para se analisar a viabilidade ou não de se realizar a ED naquele domínio. Possui também um enfoque bastante gerencial do processo de reutilização. Como os usuários irão aplicar os produtos da ED no desenvolvimento de aplicações irá depender da customização a ser realizada.
Produto	Os produtos gerados pelo método serão modelos de análise, arquiteturais e implementacionais. Como serão representados dependerá da customização a ser feita e de sua aplicação.
Ferramental Automatizado	O ODM em si não possui nenhum suporte automatizado. Em alguns projetos específicos foram criadas ou utilizadas algumas ferramentas de modelagem, mas não existe nenhum tipo de suporte efetivo para auxiliar no processo de entendimento e posterior reutilização de informações no domínio.

Tabela 2.3 - Principais características do ODM

Um aspecto interessante do método é a grande preocupação com o planejamento e organização do domínio. O ODM ressalta a importância da escolha correta de um domínio

onde será aplicada a ED, uma vez que o processo é caro e uma escolha errada pode comprometer o projeto em questão e novos projetos que possam surgir.

O ODM já foi aplicado em vários projetos mas sempre com a característica de ser particularizado para cada um deles. Esta particularização é complicada de ser realizada, pois envolve decisões que muitas vezes são difíceis de serem tomadas, tais como, que tecnologias se deve utilizar, qual a melhor representação a ser adotada, entre outros. Além disso, pela própria dimensão do ODM, ele é um método para ser aplicado em grandes organizações, onde um planejamento a nível gerencial é realizado.

2.4.3 EDLC (*Evolutionary Domain Life Cycle*)

O EDLC é um método de engenharia de domínio criado na Universidade de George Mason pelo pesquisador Gomaa e sua equipe. O EDLC adota uma abordagem gerativa de aplicações em um dado domínio, em contraste com a abordagem de componentes adotada pelo FODA.

Processo	A abordagem adotada pelo EDLC é baseada na especificação de múltiplos modelos que estão relacionados entre si. Por ser a abordagem baseada no paradigma OO, é facilmente reconhecida pelos desenvolvedores OO, o que facilita sua integração no processo de desenvolvimento das aplicações. No entanto, as representações disponibilizadas pelo ambiente de suporte podem dificultar a integração dos modelos no desenvolvimento de aplicações do domínio, a não ser que este desenvolvimento seja realizado totalmente no contexto do ambiente de suporte. A manutenção dos produtos gerados em múltiplas representações, que é como o ambiente de suporte realiza a verificação pode também representar um problema.
Produto	Os tipos de produtos disponibilizados, conforme descrito pelo método, são prioritariamente modelos de análise. Na literatura disponível a respeito do método não são especificados modelos arquiteturais. Um aspecto interessante é a verificação de consistência entre os diversos modelos.
Ferramental Automatizado	O suporte automatizado é feito através do ambiente KBSEE ("*Knowledge Based Software Engineering Environment*"). Um aspecto interessante do ambiente é o uso de mecanismos de inferência para a checagem de consistência entre os modelos. Um ponto fraco é a falta de integração das ferramentas, o que leva à necessidade de transformação em múltiplas representações. Outro ponto fraco é a falta de suporte para a fase de projeto e implementação.

Tabela 2.4 – Principais características do EDLC

Utiliza fortemente a orientação a objetos em seus modelos, inclusive incluindo uma representação gráfica dos mesmos. Outro ponto interessante do EDLC é a verificação de consistência que se faz entre os diversos modelos gerados. Com isso, se tem uma maior garantia da corretude e consistência dos mesmos. Para esta checagem de consistência, o EDLC usa uma base de regras e um mecanismo de inferência.

Além disso, o EDLC, através do seu ambiente de suporte automatizado, permite a geração dos modelos de análise adequados a uma dada aplicação, verificando possíveis inconsistências.

No entanto, o EDLC possui alguns pontos críticos, por ser baseado em uma abordagem gerativa. Sua aplicação fica limitada a domínios muito bem definidos e maduros, onde não existe uma evolução contínua do domínio. A bibliografia disponível sobre os projetos onde ele foi utilizado somente descreve a fase de análise, não havendo um detalhamento da fase de projeto. O próprio ambiente automatizado não possui suporte para a fase de projeto e implementação, o que é crítico em uma abordagem gerativa.

2.4.4 FORM (*Feature-Oriented Reuse Method*)

O método FORM (KANG *et al.*, 1998) é uma evolução do método FODA, onde um maior detalhamento de questões arquiteturais e implementacionais foi adicionado. O método ficou mais voltado para o detalhamento de como desenvolver componentes a partir das características identificadas. Para isso, utiliza técnicas como *templates*, geração automática de código, parametrização, entre outros, para o desenvolvimento dos componentes.

Novamente, a maior crítica a ser feita ao método, assim como o FODA, é a ênfase dada às funcionalidades do domínio em detrimento de outros aspectos, tais como captura de conceitos do domínio, características como desempenho, portabilidade, entre outros, que merecem destaque.

Entretanto, para um completo entendimento do domínio e suas abstrações, necessitamos de uma descrição mais detalhada não só das funcionalidades do domínio mas também de seus conceitos. Além disso, não existe um suporte automatizado para o método, o que dificulta sua aplicação.

Processo	Descreve um processo de ED completo, detalhando inclusive a fase de projeto e implementação do domínio. Os modelos especificados seguem uma abordagem OO e de componentização, o que facilita sua integração no desenvolvimento de software atual.
Produto	O principal produto disponibilizado pelo método é o modelo de características, onde as principais características (sob um prisma de construção de software) são explicitadas. Uma abordagem interessante são as regras de composição e o armazenamento de decisões tomadas em relação ao modelo de características. Os modelos arquiteturais e implementacionais não são bem detalhados. São apenas disponibilizadas diretivas para auxiliar na sua criação. As características selecionadas para uma dada aplicação seriam os guias para a criação da arquitetura. Nenhum detalhamento de como criar explicitamente a arquitetura e seus componentes, inclusive especificando interfaces, é dado.
Ferramental Automatizado	Não existe nenhum suporte automatizado. Pela complexidade na seleção das características e verificação manual de consistência (que é um dos pontos fortes do método) a aplicação do método se torna bastante difícil.

Tabela 2.5 – Principais características do FORM

2.4.5 OODE (Object-Oriented Domain Engineering)

O método de engenharia de domínio OODE (CHAN *et al.*, 1998) é centrado principalmente em dois pilares: a orientação a objetos e a distribuição. É um método desenvolvido internamente na BOEING para atender à demanda de seus projetos de desenvolvimento de software.

Segundo os próprios autores descrevem, o método é um apanhado de diversas abordagens de modelagem OO, como o uso de CRC *cards* (em CHAN *et al.*, 1998), OMT (RUMBAUGH *et al.*, 1991), entre outros, adaptadas para o contexto da engenharia de domínio. Em termos tecnológicos, o OODE utiliza abordagens avançadas para a modelagem e implementação de componentes no domínio, com o foco centrado na criação de arquiteturas de objetos distribuídos e heterogêneos, com o auxílio de *design patterns* (GAMMA *et al.*, 1994). Esta é uma abordagem bastante interessante, uma vez que a computação distribuída e a necessidade de cooperação entre aplicações heterogêneas é cada vez mais requerida.

No entanto, para um método de engenharia de domínio, apenas focar um estilo arquitetural, que, segundo a classificação de BUSHMANN (1996), seria o estilo

arquitetural *Broker*, é muito limitado, uma vez que em um dado domínio, reconhecidamente podemos ter aplicações que seguem outros estilos arquiteturais que não este. Desta forma, o método fica limitado a disponibilizar componentes que atendam aplicações que sigam esta arquitetura. Atualmente, apesar de ser uma tendência, poucas aplicações se baseiam na arquitetura *Broker* (OMG, 2000). Outro ponto crítico para a aplicação do método é a falta de ferramental adequado para sua execução, principalmente no que tange a especificação de componentes distribuídos e suas interfaces.

Processo	O OODE utiliza um processo em espiral para a especificação de componentes reutilizáveis. O processo é constituído de três fases: análise de domínio, projeto do domínio e instalação (entrega) dos produtos. Na literatura disponível sobre o método é dada pouca ênfase à fase de projeto e implantação. No entanto, pelos estudos de caso apresentados, podemos concluir que em todas as fases, produtos específicos são disponibilizados e validados através de testes.
Produto	São disponibilizados produtos reutilizáveis em todas as fases. Como o método utiliza uma abordagem OO, ele é facilmente integrado em processos de desenvolvimento de aplicações OO. No entanto, os produtos disponibilizados na fase de projeto e implementação podem não ser adequados para todos os tipos de aplicações, uma vez que o OODE utiliza uma abordagem de arquitetura distribuída.
Ferramental Automatizado	Não existe nenhum suporte automatizado específico para o método. Isto pode dificultar sua aplicação em domínios médios a grandes. O batimento dos modelos gerados com os padrões de projeto e de análise pode ser dificultado neste caso. Além disso, é necessário um suporte mais específico para a criação de arquiteturas distribuídas, principalmente no que tange ao interfaceamento das mesmas.

Tabela 2.6 – Principais características do OODE

2.4.6 RSEB

O RSEB (*Reuse-Driven Software Engineering Business*) é um método para o desenvolvimento de aplicações baseado em reutilização (JACOBSON *et al.*, 1997). Os autores partem da abordagem OO para a criação de um método com características de desenvolvimento baseado em componentes. Para isso, o método estende a abordagem OO

com construções, tais como "pontos de variação[3]" em casos de uso, uso de *frameworks* e *facades*[4], para permitir o desenvolvimento de aplicações baseadas em componentes previamente definidos.

O desenvolvimento de aplicações se baseia, então, no preenchimento dos "pontos de variação", instanciação de *frameworks* e especificação de contratos que permitam a reutilização de interfaces *(facades)*.

Conforme podemos perceber, o enfoque básico do método é no desenvolvimento de aplicações baseadas em componentes, não existindo uma sistemática para a engenharia de componentes, ou seja, para a construção dos componentes. Podemos notar algumas atividades semelhantes às da engenharia de domínio, mas não existe uma ênfase em ED. Além disso, não existe um modelo de abstrações, nos moldes do modelo de características, que guie e facilite a reutilização.

Processo	O RSEB como um processo para engenharia de aplicações baseadas em componentes é bastante interessante. No entanto, como um método baseado em reutilização, ele não atende a alguns requisitos. Seus produtos não são organizados de forma a serem identificados como reutilizáveis. Desta forma, é difícil para o usuário identificar os produtos a serem reutilizados.
Produto	Apesar de se ter uma ênfase na generalização de produtos de todas as fases, com o intuito de que estes sejam reutilizados, não existe uma sistemática de criação dos mesmos e, por este motivo, fica difícil garantir sua reutilizabilidade. Os modelos disponibilizados são casos de uso, modelos de classes, entre outros modelos OO, cuja ênfase na variabilidade é ressaltada.
Ferramental Automatizado	Conforme já ressaltado, não existe um suporte automatizado específico para o método. O que existem são ferramentas tais como o Rational Rose que podem ser utilizadas. No entanto, nem sempre estas ferramentas estão integradas.

Tabela 2.7 – Principais características do RSEB

Um ponto interessante do RSEB é a preocupação com a descrição arquitetural de forma detalhada. No entanto, não é feita uma descrição detalhada do que realmente seria

[3] Pontos de Variação são partes específicas onde se pode acoplar novas construções que modifiquem e/ou especializem a funcionalidade de uma dada construção.

[4] *Facades* são as especificações de uma interface, fachada externa de uma determinada construção, ou seja, o que é visível para as demais construções.

uma arquitetura de componentes para o RSEB. O que é apresentado é uma arquitetura para aplicações baseadas em componentes que é basicamente uma arquitetura em camadas.

Uma crítica ao RSEB é a falta de suporte automatizado para o método. Os autores argumentam que ferramentas CASE como o Rational Rose (RATIONAL, 2000) e repositórios de componentes como o da Microsoft (MICROSOFT, 2000) poderiam ser utilizados. No entanto, no caso do Rational Rose, a ferramenta não possui suporte para a modelagem dos pontos de variação, interfaces ou *frameworks*, o que seria a parte interessante do método e o que o difere das outras abordagens OO. Além disso, não existe uma perfeita integração entre o ROSE e o repositório da Microsoft, que além do mais só disponibiliza componentes Microsoft.

2.4.7 FeatuRSEB (*Featuring RSEB*)

Observando as deficiências do RSEB em relação ao suporte à engenharia de domínio e do FODA em relação à utilização de uma abordagem de modelagem mais atual, foi criado o FeatuRSEB, no contexto dos projetos de engenharia de domínio da TELECOM Itália (GRISS *et al.*, 1998).

Assim, o FeatuRSEB tenta unir as vantagens do FODA, que é a modelagem de abstrações do domínio, que representaria quais funcionalidades são importantes no domínio com as vantagens do RSEB, que é a descrição detalhada de como implementar estas funcionalidades.

O método foi aplicado na TELECOM Itália e, segundo os autores, obteve bons resultados. Novamente, a grande crítica é na ênfase em modelar apenas as funcionalidades abstratas do domínio. SIMOS *et al.* (1998) ressaltam este ponto, ou seja, a necessidade de modelar mais do que somente as funcionalidades do domínio, com um exemplo tirado do estudo de viabilidade do FODA (KANG *et al.*, 1990), que apresenta uma modelagem do domínio de gerenciamento de janelas gráficas. No exemplo, ele atesta que sem uma definição precisa do que é "janela", o que é "redimensionamento" e o que é "dinâmico", as funcionalidades do domínio "suporte ao redimensionamento de janelas" e "suporte dinâmico ao redimensionamento de janelas" e seu relacionamento de generalização/especialização ficam restritos a uma decisão do projetista e não podem ser comprovados e melhor entendidos pelo usuário.

Processo	Processo bem definido e detalhado para a engenharia de domínio, com uma preocupação grande de representar o que (modelo de características) e o como (modelo de caso de uso e seus desdobramentos). Utiliza abordagens atuais de engenharia de software como padrões e frameworks. Pela própria característica dos modelos, a integração com abordagens de desenvolvimento OO é facilitada.
Produto	Os produtos disponibilizados facilitam a reutilização quando da escolha de quais componentes serão reutilizados (modelo de características) e no detalhamento destes produtos, de forma que possam ser integrados ao processo de desenvolvimento de aplicações.
Ferramental Automatizado	Não existe nenhuma ferramenta que suporte o processo de desenvolvimento do FeatuRSEB. Os autores descrevem algumas ferramentas que poderiam ser utilizadas como o Rational Rose, *Platinum Plus* e o *Microsoft Repository*, mas nenhuma disponibiliza as funcionalidades requeridas pelo método e que o fazem um método de engenharia de domínio. Os autores também mencionam um ferramental em desenvolvimento denominado DEUS.

Tabela 2.8 – Principais características do FeatuRSEB

2.4.8 Catalysis

O método Catalysis é um método de desenvolvimento baseado em componentes completo, cobrindo todas as fases do desenvolvimento de um componente, a ser utilizado no desenvolvimento de uma dada aplicação, desde a especificação até sua implementação (D'SOUZA *et al.*, 1998).

Todos os modelos são descritos em detalhe e há uma grande preocupação com a especificação da arquitetura da aplicação, que na verdade seria a especificação da colaboração entre os componentes. É interessante ressaltar que esta descrição arquitetural não se prende a nenhum estilo arquitetural específico, como é o caso do OODE.

Em termos arquiteturais, o Catalysis descreve dois níveis, o nível da macro-arquitetura, que seria o nível de colaboração entre os componentes e a micro-arquitetura, que seria a descrição de como cada componente é projetado internamente.

Existe uma preocupação grande em relação à interface dos componentes. Esta abordagem é interessante uma vez que a possibilidade de colaboração entre os componentes para formar a arquitetura é dada pelas interfaces. Por isso, existe todo um formalismo para a definição das interfaces, com especificação de restrições, detalhamento da interface, entre outros, de forma que esta consistência possa ser verificada de maneira formal.

Um aspecto interessante do Catalysis é que, apesar de não ser um método de engenharia de domínio, ele apresenta um conceito de tipo, que seria a descrição de alguma funcionalidade da aplicação em termos apenas de seu comportamento externo. Esta definição vem muito ao encontro da definição de características que descrevem funcionalidades. No entanto, o Catalysis não especifica nenhum tipo de modelo de relacionamento destas funcionalidades, capturando as similaridades e diferenças entre as mesmas, como é feito no modelo de características. Este é o ponto onde o Catalysis não pode ser considerado um método de ED; ele não possui nenhum modelo que apresente as abstrações do domínio de forma a facilitar a reutilização. Além disso, apesar de detalhar bastante como seria o desenvolvimento de componentes, o método está mais preocupado com o desenvolvimento de uma única aplicação e não com uma família destas. Assim, os componentes não são especificamente empacotados para reutilização posterior.

Outro ponto interessante do Catalysis e que está em conformidade com alguns dos princípios da reutilização, segundo apresentado por KRUEGER (1992), é o mecanismo de "realização", onde uma descrição mais abstrata de um componente está ligada ao seu (s) detalhamento (s) em fases de projeto e implementação.

Processo	Fases bem definidas para o desenvolvimento de componentes, utilizando construções consagradas no desenvolvimento de software atual como padrões e frameworks. Bom detalhamento da fase arquitetural, com especial ênfase na parte de especificação formal das interfaces. Falta um comprometimento maior com a reutilização para que possa ser adotado como um processo de engenharia de domínio.
Produto	Não existe um modelo que apresente "o que" reutilizar. Só apresenta modelos de como são especificadas estas funcionalidades. No entanto, neste segundo aspecto, os modelos são bastante consistentes, inclusive com mecanismos de rastreabilidade entre os mesmos. Podem ser facilmente integrados em um processo de desenvolvimento OO.
Ferramental Automatizado	Como o método é muito badalado, algumas ferramentas específicas estão sendo disponibilizadas para o mesmo. Como exemplo, podemos citar o Cool-Gen.

Tabela 2.9 – Principais características do Catalysis

Na literatura disponível, consideramos o Catalysis como o melhor método de desenvolvimento baseado em componentes, pois o detalhamento do seu processo é bastante consistente. Este ponto de vista é comprovado pela utilização do Catalysis em diversas

abordagens de desenvolvimento de software, inclusive com ferramentas específicas para ele.

2.5 Conclusões

Existem algumas similaridades entre os enfoques de ED e o DBC. Ambos buscam o desenvolvimento de componentes que possam ser utilizados e reutilizados novamente no desenvolvimento de aplicações. No entanto, o que podemos verificar é que ao passo que a ED está essencialmente preocupada com a captura das abstrações do domínio e como estas abstrações estão relacionadas, de forma a compor uma infra-estrutura de reutilização que seja capaz de disponibilizar componentes reutilizáveis, o DBC está mais preocupado em como estes componentes podem cooperar para a especificação de uma dada aplicação.

Podemos notar claramente nos métodos de ED uma grande preocupação com a especificação de um modelo de abstrações, que capture as similaridades e diferenças do domínio. A concretização, ou melhor, segundo KRUEGER (1992) a "realização" destas abstrações, principalmente nos métodos mais antigos como FODA e ODM, não recebe muita ênfase, disponibilizando-se apenas algumas diretivas de como deve ser a fase de projeto e implementação. As metodologias de ED atuais mais representativas, tais como FeatuRSEB, FORM e OODE, já apresentam um detalhamento maior das etapas que tratam este aspecto específico, i.e., as fases de projeto e implementação, mas não apresentam a captura destas abstrações nas fases posteriores, além de não especificarem como estas abstrações são concretizadas em componentes reutilizáveis.

Nas seções 2.4.1 a 2.4.6 foram analisados e comparados os principais métodos existentes. Verificamos que nenhum deles apresenta a "realização das abstrações do domínio em componentes reutilizáveis. Essa constatação compromete a efetividade da reutilização. Acreditamos que para a reutilização ser efetiva, todas as etapas (análise, projeto e implementação) do processo de engenharia de domínio devem ser contempladas com igual ênfase, compreendendo assim desde a especificação do conhecimento do domínio até a concretização deste conhecimento em um componente implementacional.

Os métodos de DBC têm uma grande preocupação com o detalhamento dos componentes e seus relacionamentos para a especificação de aplicações. Por serem abordagens preocupadas com a especificação de aplicações, não têm uma preocupação

maior no processo de abstração dos componentes, com o objetivo claro de que estes possam ser reutilizados o maior número de vezes por aplicações similares.

Portanto, a união dessas duas abordagens poderia resultar em um método de Engenharia de Domínio voltado para a especificação de componentes (EC) que unisse o melhor das duas abordagens, ou seja, da ED viria a especificação de um modelo de abstrações capturando as similaridades e diferenças do domínio, e do DBC o detalhamento necessário para as fases de projeto e implementação, com grande preocupação com as interfaces dos componentes e arquitetura de software. O modelo de abstrações serviria de guia para a reutilização e posterior empacotamento dos componentes, além de permitir a especificação dos componentes em um alto nível de abstração, de forma que seu grau de reutilização fosse o maior possível.

Analisando as metodologias apresentadas na seção anterior, podemos concluir que um método de engenharia de domínio atual, contemplando o desenvolvimento baseado em componentes, deveria levar em consideração os seguintes pontos:

- Especificar um modelo de abstrações do domínio, que não só levasse em consideração aspectos funcionais das aplicações do domínio mas também considerasse conceitos, ligações com outros domínios correlatos, atores envolvidos e características não funcionais. Neste sentido, o modelo de abstrações poderia ser considerado uma ontologia[5] (GUARINO, 1998) do domínio. Este modelo serviria ainda como um guia para o processo de reutilização.
- Criar um mecanismo que permitisse a verificação de consistência entre os modelos e também verificasse a consistência na seleção de abstrações que não fossem conflitantes. Esta checagem de consistência é uma característica muito importante.
- Definir um mecanismo de ligação entre os modelos de abstração e os modelos de especificação dos componentes de forma que se pudesse identificar claramente que abstrações deram origem a que componentes e quais os conceitos ligados à especificação do componentes. Seria a ligação de "realização" que KRUEGER (1992) descreve.
- A definição dos componentes deveria seguir uma abordagem essencialmente DBC, que possui o detalhamento necessário para tal, com definição de interfaces, levando em

[5] Metadados semanticamente rico para a descrição precisa de uma determinada área de aplicação ou domínio.

consideração as restrições do estilo arquitetural adotado na colaboração entre os componentes.

- Necessidade de suporte automatizado para o processo de criação de componentes reutilizáveis, uma vez que as ligações entre os modelos, checagem de consistência, entre outros, são atividades complexas de serem realizadas manualmente.

Esta tese apresenta no capítulo 4 um processo de engenharia de domínio baseado em componentes que contempla, entre outros aspectos, a "realização" das abstrações em componentes reutilizáveis e o suporte automatizado ao processo .

3 Sistemas baseados em Agentes

3.1 Introdução

A pesquisa em agentes inteligentes é no momento considerada como um caminho promissor para o desenvolvimento de aplicações, principalmente aquelas relacionadas a sistemas distribuídos e inteligentes (JENNINGS, WOOLDRIDGE, 1997). A tecnologia de agentes inteligentes tem sido utilizada em diversos tipos de aplicações, tais como filtros para sistemas de mensagens, controle de tráfego aéreo, mecanismo para recuperação de informação na Internet, educação, entretenimento, comércio eletrônico, entre outros (HUHNS, SINGH, 1997). JENNINGS *et al.* (1998) ressaltam que, devido ao interesse e a intensa atividade de pesquisa em relação a agentes, várias definições existem para o que seria um agente e, consequentemente, um sistema baseado em agentes. Uma definição que está sendo considerada como adequada pela maioria dos pesquisadores na área seria a seguinte:

"*Um* ***agente*** *é um sistema computacional encapsulado, que é situado em algum ambiente e é capaz de ações flexíveis e autônomas no ambiente de forma a alcançar seus objetivos*". (WOOLDRIDGE, 1997)

Quando consideramos um agente como um sistema encapsulado, quer dizer que um agente deve ser uma entidade facilmente identificável, com interfaces bem definidas. Em relação a ser situado em algum ambiente, significa que o agente deve receber estímulos (informações) em relação ao estado do ambiente e agir, se for o caso, de acordo com os estímulos recebidos. Além disso, devem estar claros os objetivos que o agente tem que cumprir, tendo controle sobre seu estado interno e seus atos. Mais ainda, um agente deve ser reativo, ou seja, ter a capacidade de responder em tempo hábil às mudanças que ocorrem no ambiente para que possa alcançar seus objetivos, e também proativo, adotando oportunamente novos objetivos e tomando assim novas iniciativas. Além disso, alguns autores (JENNINGS, 1998) (WOOLDRIDGE, 1997) argumentam que uma característica essencial para o agente é que ele seja socializável, ou seja, que seja capaz de interagir, quando apropriado, com outros agentes e mesmo com humanos, com o objetivo de

completar sua tarefa ou auxiliar outros com suas atividades. Outras características como mobilidade e autonomia, também são ressaltadas por alguns autores.

HUHNS e SINGH (1997) ressaltam que um agente pode apresentar estas características em um maior ou menor grau, de acordo com o seu ambiente e a tarefa a ser executada, ou seja, de acordo com o ambiente e o objetivo do agente, uma ou mais características serão ressaltadas em detrimento de outras. Como exemplo, podemos citar os assistentes pessoais (*personal assistants*) cujo objetivo é auxiliar um usuário (humano) na realização de uma determinada tarefa. Neste caso específico, a característica da sociabilidade do agente é uma das mais acentuadas em detrimento, por exemplo, de sua autonomia.

A partir da conceituação de agente, podemos então definir o que é um **sistema baseado em agentes (SBA)**. Segundo JENNINGS (1998), um sistema baseado em agentes é uma aplicação onde a tecnologia chave utilizada é a tecnologia de agentes. Um SBA pode conter um ou mais agentes. Apesar da abordagem de múltiplos agentes ser mais interessante, existem casos onde a solução utilizando somente um agente é adequada. Os assistentes pessoais são um caso típico desta abordagem. No entanto, a abordagem de sistemas multi-agentes, mais conhecidos como MAS (*Multi Agent Systems*), é mais discutida atualmente, principalmente por conta do interesse crescente em sistemas distribuídos, que seria um contexto adequado para a utilização de MAS. Desta forma, um MAS é um sistema composto de vários agentes que interagem entre si, representando uma visão descentralizada do problema, com múltiplos focos de controle, múltiplas perspectivas ou interesses competitivos (JENNINGS, 1998). Como vantagens de um MAS temos a distribuição e possível concorrência na resolução de problemas. Assim, um MAS é um tipo de sistema perfeitamente adequado para o ambiente da Internet e por isso sua grande popularidade entre os pesquisadores das mais diversas áreas.

Um MAS pode então ser definido como sendo um sistema formado por agentes fracamente acoplados, que devem interagir para resolver um determinado problema, que os agentes sozinhos não teriam capacidade e nem conhecimento para resolverem (JENNINGS, 1998). Podemos ressaltar como principais características de um MAS:

- Cada agente tem informações incompletas do problema, tendo um ponto de vista limitado;

- Não existe um sistema global de controle;
- Os dados são descentralizados;
- A computação é assíncrona.

Um ponto importante em relação a um MAS é como se dá a interação entre os vários agentes que compõem o sistema. Vários padrões de interação já foram descritos na literatura (HUHNS, SINGH, 1997) (JENNINGS *et al.*, 1998) (WOOLDRIDGE, 2000), tais como cooperação (agentes que trabalham em conjunto com um objetivo comum), coordenação (é escolhido um agente que organiza os passos para a resolução do problema de forma a evitar interações não necessárias) e negociação (agentes estão em constante negociação para chegar a um acordo, que é aceito por todas as partes envolvidas). Para operacionalizar estas interações, várias abordagens são propostas, desde o uso de ACL (*Agent Communication Languages*), que são linguagens para comunicação entre agentes em um nível semântico, até à utilização de um software de conexão (*middleware)* tipo CORBA, cuja preocupação é mais sintática. Este último caso seria o que chamamos de um MAS simplificado.

Para mostrar a utilidade efetiva da tecnologia de agentes no desenvolvimento de novas aplicações, JENNINGS e WOOLDRIDGE (1997) ressaltam a habilidade de resolver problemas complicados de serem resolvidos por outras tecnologias e também a habilidade de resolver problemas de uma melhor maneira do que com as outras tecnologias. Os autores ressaltam que para que uma tecnologia seja útil no mercado computacional, ela tem que oferecer uma ou outra destas habilidades e a tecnologia de agentes atende a estes requisitos.

No que tange a resolução de novos problemas, a tecnologias de agentes é adequada para o desenvolvimento de novos tipos de aplicações como sistemas abertos, onde a estrutura e os componentes podem mudar ao longo do tempo, sistemas complexos, utilizando a modularidade (conjunto de agentes interoperantes) e a abstração (cada um sabe por si só desempenhar seu papel), sistemas para auxílio ativo ao usuário na solução de problemas, não sendo um mero executor de tarefas, mas também auxilie na escolha do melhor caminho. Analisando a tecnologia de agentes, podemos confirmar que este tipo de "atitude" é o que se espera de um software baseado em agentes.

Por outro lado, a tecnologia de agentes também pode melhorar a eficiência no desenvolvimento de software, provendo melhores maneiras de conceitualizar e implementar uma dada aplicação. Com a tecnologia de MAS, o controle, dados e conhecimento de uma aplicação ficam naturalmente distribuídos, permitindo a interoperação de componentes distribuídos para a realização de uma dada aplicação. Com esta descrição, pode parecer que a tecnologia de agentes veio para resolver todos os problemas do desenvolvimento de software atual. No entanto, isso não é verdade, uma vez que a tecnologia é nova e ainda sequer se chegou a um consenso sobre o que é e o que não é um agente, existindo muitos pontos problemáticos que dificultam a aplicação desta tecnologia, que por si só é bastante complexa. Dentre os pontos problemáticos citados na literatura, temos (JENNINGS, WOOLDRIDGE, 1997):

- Não existe um controlador do sistema: a tecnologia de agentes pode não ser apropriada para domínios onde seja necessário um controle global, ou respostas em tempo real, e onde *deadlocks* devem ser evitados. Para aplicações com estes tipos de restrições, a tecnologia de agentes não é adequada;
- Confiança e delegação: para os usuários ficarem confortáveis com a idéia de delegar tarefas aos agentes, estes têm que confiar nos agentes. Esta confiança vem com o tempo e, durante este período, o agente tem que demonstrar de todas as maneiras que ele é confiável, o que muitas vezes é difícil de se conseguir.

HUHNS e SINGH (1997) apresentam as principais características que um agente, um sistema de agentes e uma infra-estrutura de suporte a agentes devem prover, apresentando uma faixa de valores que podem ser considerados na classificação de um MAS. As tabelas 3.1, 3.2 e 3.3 apresentam estas características para agentes de forma isolada, sistemas baseados em agentes e infra-estrutura de suporte, respectivamente. Em relação às tabelas, as propriedades dizem respeito às características que um dado agente pode apresentar ou não. A faixa de valores é um maior ou menor grau de uma dada propriedade que o agente pode apresentar e o tipo de propriedade caracteriza se a propriedade é uma característica interna do agente, ou seja, tem a ver com suas funcionalidades internas ou se é uma característica externa, ou seja, se é relacionada com sua capacidade de comunicação com o meio externo.

3.2 Evolução da Tecnologia

A Inteligência Artificial como ciência já tem mais ou menos 40 anos de existência (BIGUS, 1997). No entanto, somente com o advento dos sistemas especialistas foi que a IA começou a ser empregada com maior ênfase na engenharia de software e considerada no desenvolvimento de aplicações comerciais.

Propriedades	Faixa de Valores	Tipo de Propriedade
Tempo de vida	Transiente até persistente	Interna
Nível de cognição	Reativo/Deliberativo	Interna
Paradigma de construção	Declarativa/Procedural	Interna
Mobilidade	Estacionário/Intinerante	Interna
Adaptabilidade	Aprendiz/Autodidata	Interna
Localidade	Local/Remota	Externa
Autonomia social	Independente/Controlado	Externa
Sociabilidade	Responsável/Membro de equipe/Altruístico/ Consciente	Externa
Colaboração	Cooperativo/Competitivo/Antagonista	Externa
Interação	Direta/Facilitador/Mediador	Externa

Tabela 3.1 – Características de Agentes

Propriedades	Faixa de Valores
Unicidade	Homogêneo/Heterogêneo (considera se o sistema é formado por um conjunto de agentes do mesmo tipo ou não)
Granularidade	Fina/Grossa (se os agentes são aplicações completas – granularidade grossa, ou não)
Estrutura de Controle	Hierarquia/democracia
Autonomia de execução	Independente/controlada

Tabela 3.2 – Característica de Sistemas de Agentes

Propriedades	Faixa de Valores
Autonomia de projeto	Plataforma/linguagem/arquitetura interna/protocolo de interação
Infra-estrutura de comunicação	Memória compartilhada/ baseada em mensagens/ ponto a ponto/ multicast/ broadcast/ síncrona/assincróna
Serviço de diretório	White pages/ yellow pages
Protocolos de mensagens	KQML/HTTP/HTML/OLE/CORBA/DCOM
Serviços de Mediação	Baseado em ontologias/ transações
Serviço de Segurança	Controle temporal/ autenticação

Tabela 3.3 – Características da infra-estrutura de apoio

A idéia de agentes autônomos começou a ganhar força, principalmente a partir da década de 70, com pesquisas relacionadas a IA simbólica, ou bases de conhecimento, ou modelos mentais(JENNINGS *et al.*, 1998). Ao longo do tempo, algumas críticas surgiram em relação a esta abordagem, surgindo então outro ramo que acreditava que os agentes deveriam ser implementados utilizando uma abordagem mais comportamental, devendo ser baseada em um histórico de comportamentos. Os pesquisadores em IA denominam a primeira abordagem de **deliberativa** e a segunda de **reativa**. Hoje em dia, os pesquisadores acreditam em uma abordagem híbrida, que una as abordagens deliberativa e reativa em camadas.

Em relação à arquitetura de um agente, uma abordagem que é bastante utilizada é a denominada BDI (*Belif-Desire-Intention*). O B corresponde às informações que o agente recebe de seu ambiente. O D representa as opções disponíveis para o agente, ou seja, diferentes estados que o agente pode escolher e o I representa os estados que o agente escolheu e alocou recursos para eles. O mecanismo de "raciocínio" de um agente utilizando uma arquitetura BDI seria o seguinte: atualização constante de seus dados a partir das informações do ambiente; decisão sobre quais opções estão disponíveis e filtragem destas opções para determinar novas intenções, agindo de acordo com estas intenções. Este modelo BDI é utilizado para um único agente. Mais recentemente, com a computação distribuída e uso da Internet, surgiu um incentivo cada vez maior para as pesquisas em relação a sistemas multi-agentes (MAS).

A origem dos sistemas MAS advém dos estudos no campo da IA distribuída. As pesquisas em relação a MAS evoluíram ao longo dos anos, chegando à definição atual de vários agentes fracamente acoplados que trabalham juntos para resolver um dado problema. Estes agentes são autônomos e podem também ser heterogêneos.

Durante esta evolução, foram propostos vários modelos para a resolução de problemas utilizando MAS. O primeiro modelo proposto foi o modelo de atores, onde a idéia principal era ter componentes interativos e autônomos que se comunicavam através da passagem de mensagens assíncronas. Logo após o modelo de atores, foi proposto o modelo chamado Contract Net Protocol, permitindo a troca de tarefas entre agentes e realizando também um balanceamento de carga entre os agentes. A necessidade de sincronização entre os agentes e um planejamento prévio das ações dos agentes levou ao surgimento do modelo

Cooperativo. Como a resolução de conflitos entre os interesses dos vários agentes não era tratada pelo modelo cooperativo, surgiu a idéia dos modelos baseados em negociação.

Atualmente, as pesquisas relacionadas a MAS estão direcionadas para os chamados *middle agents* (WOOLDRIDGE, 2000), (JENNINGS, 1998) que seriam agentes responsáveis por encontrar outros agentes com capacidade de executar uma determinada tarefa. Outro foco das pesquisas mais recentes são as chamadas linguagens de comunicação entre agentes, as ADL's e o uso de ontologias na comunicação entre agentes. Além destas, pesquisas também estão sendo desenvolvidas no contexto de alocação de recursos para agentes. Assim, podemos observar que, apesar da idéia de MAS ser bem aceita pela comunidade de pesquisa, ainda há muito a ser feito para que a tecnologia se torne madura.

3.3 Áreas de utilização da tecnologia de agentes

Até bem pouco tempo atrás, a tecnologia de agentes era restrita somente ao contexto das universidades e laboratórios de pesquisa. Atualmente, principalmente por conta da Internet, aplicações baseadas em agentes estão sendo utilizadas para a resolução de problemas reais. No entanto, ainda há muito o que fazer para a tecnologia ser aplicada em larga escala.

JENNINGS *et al.* (1998) dividem as áreas de aplicação da tecnologia de agentes em dois grupos: aplicações industriais e comerciais. Até a explosão do uso da Internet, as aplicações industriais eram as que mais utilizavam a tecnologia de agentes. Hoje este panorama mudou, principalmente por conta das aplicações para gerência de informação e comércio eletrônico, sendo estas aplicações as principais responsáveis pelo grande interesse atual em agentes.

- Aplicações industriais: Foram as primeiras a utilizarem a tecnologia de agentes. Podem ser divididas nos seguintes tipos: manufatura, controlando as múltiplas fases do processo, com agentes que colaboram para o resultado final, telecomunicações, principalmente aplicando o modelo de negociação, controle de tráfego aéreo, e sistemas de transporte;
- Aplicações comerciais: O foco mais intenso de pesquisas atuais é em aplicações comerciais, principalmente aplicações voltadas para Internet, como o gerenciamento de informações e comércio eletrônico. Em relação ao gerenciamento de informações, existem pesquisas intensas na área, devido à grande massa de dados

disponibilizada pela Internet. Outro tipo de aplicação que vem utilizando fortemente a tecnologia de agentes é o de aplicações voltadas para o comércio eletrônico. Neste contexto, os agentes podem automatizar várias tarefas na compra e venda de produtos, podendo servir também como um apoio à decisão nesta compra e venda. Além destes dois tipos de aplicação, outro tipo de aplicação comercial que utiliza a tecnologia de agentes são as aplicações para gerência de negócio, onde são criados agentes responsáveis por cada parte do negócio e quando juntos, auxiliam na tomada de decisões gerenciais.

- Entretenimento: Devido ao retorno econômico que pode ser alcançado com este tipo de aplicação, cada vez mais aplicações nesta categoria têm sido desenvolvidas, principalmente relacionadas a jogos, cinema interativo e realidade virtual;
- Aplicações médicas: A área de sistemas especialistas foi largamente utilizada em aplicações para medicina. Atualmente, está acontecendo uma evolução natural deste tipo de sistema para a tecnologia de agentes, principalmente para unidades de terapia intensiva, onde especialistas de diversas áreas devem interagir. Neste caso, pode-se utilizar a tecnologia de MAS, onde cada agente é responsável por uma área de especialidade.

Apesar de um vasto campo de aplicação da tecnologia de agentes, ainda há muito o que se pesquisar, no sentido de evoluir o desenvolvimento de aplicações baseadas em agentes para uma prática de engenharia de software. Hoje em dia, os pesquisadores argumentam que os sistemas são criados de maneira ad-hoc e muitas vezes por especialistas na tecnologia de agentes e não especialistas em desenvolvimento de software. Isto acarreta o desenvolvimento de projetos pobres e de difícil manutenção. Além disso, os pesquisadores (JENNINGS, 1998) (HUHNS *et al.*, 1997) ressaltam que faltam ferramentas para auxiliar este desenvolvimento. Assim, conforme ressalta WOOLDRIDGE (2000), uma área que merece maior estudo é a de como a engenharia de software pode trazer contribuição para sistemas baseados em agentes.

3.4 Tecnologia de Agentes e Engenharia de Software

O objetivo maior da Engenharia de Software é a sistematização do processo de desenvolvimento de aplicações, de forma que mesmo aplicações complexas sejam

desenvolvidas com qualidade e custo aceitável. Atualmente, no desenvolvimento de software vem crescendo o uso de técnicas de desenvolvimento baseado em componentes (DBC), conforme descrevemos no capítulo 2. Um dos motivos que o DBC ganhou tanto destaque é porque prega o desenvolvimento de software como partes interoperáveis, o que para um ambiente distribuído como é a Internet, é perfeitamente adequado. Dentro deste contexto, podemos dizer que a próxima onda será o desenvolvimento baseado em agentes, com a diferença que, na tecnologia de agentes, os componentes terão atitude e poderão agir de forma autônoma.

Em DBC, conforme descrito no capítulo 2, ainda existem poucos métodos realmente adequados para a sistematização do processo. Mais escassos ainda são os métodos para o desenvolvimento de SBAs. WOOLDRIDGE (1998) ressalta que atualmente não existem técnicas e nem ferramentas adequadas para lidar com a complexidade dos SBAs. JENNINGS *et al.* (1998) ressaltam que foram realizadas algumas tentativas de adaptar técnicas de desenvolvimento OO e de DBC para ser utilizado no desenvolvimento baseado em agentes. Chegou-se à conclusão que com certeza o paradigma OO pode ser utilizado para a implementação dos sistemas. No entanto, os pesquisadores alertam que um agente não é um objeto (WOOLDRIDGE, 1998) (HUHNS, SINGH, 1997). Na verdade, um agente é uma entidade ativa e autônoma, devendo tomar suas próprias atitudes sem que nenhum outro agente tenha que ativá-la. Além disso, os pesquisadores argumentam ainda que o modelo de interação entre agentes é mais sofisticado, devendo ser um modelo baseado em colaboração ou negociação, conforme ressaltamos na introdução. Todas estas considerações com certeza podem ser implementadas com o paradigma OO, mas não fazem parte do paradigma em si.

A abordagem de DBC se mostra um pouco mais adequada e similar à tecnologia de agentes, uma vez que o conceito de componente como uma entidade capaz de realizar uma determinada tarefa, sendo auto-contida, é bem próximo do conceito de agentes. A diferença é que agentes devem obrigatoriamente possuir capacidade de raciocínio, no sentido em que devem ser capazes de responder a requisições a respeito dos serviços que disponibilizam. Componentes não são autônomos, não existindo a noção de reação, pró-ação ou comportamento social (JENNINGS *et al.*, 1998). Logo, um agente pode ser considerado um componente e pode ser implementado utilizando a tecnologia de orientação a objetos, mas o

contrário não é verdade, ou seja, nem todo componente pode ser considerado um agente e o mesmo se aplica a um objeto em relação a agentes.

Partindo do pressuposto que um agente pode ser implementado com a tecnologia de objetos, alguns pesquisadores começaram a utilizar metodologias de desenvolvimento OO para implementar SBAs. No entanto, observaram que os SBAs eram muito mais complexos que outras aplicações baseadas em objetos. Esta complexidade está justamente na modelagem do agente. WOOLDRIDGE (2000), atesta que, para especificarmos de forma adequada um agente, os seguintes aspectos devem ser explicitados:

- As crenças que o agente possui, ou seja, a informação que ele possui de seu ambiente;
- As metas que o agente tem que atingir;
- As ações que o agente deve realizar e os efeitos destas ações.

Assim, além das técnicas de desenvolvimento OO e mais recentemente DBC, temos que disponibilizar mecanismos que permitam a modelagem de aspectos específicos da tecnologia de agentes. Mais especificamente, os métodos OO e de DBC não conseguem capturar de maneira adequada o comportamento autônomo e flexível de um agente, as interações entre os agentes e a estrutura organizacional de um SBA.

WOOLDRIDGE *et al.* (1998) apresentam uma abordagem que tenta descrever aspectos característicos de um MAS. Apesar de não tratar todos os casos (o modelo de interação baseado em negociação não é abordado), esta abordagem é um primeiro passo em direção a um método específico para SBAs. O sistema como um todo é representado na abordagem como uma entidade denominada "Sistema". Esta entidade tem como significado uma "sociedade ou organização", considerando a aplicação a ser desenvolvida como uma sociedade de agentes. O próximo nível na hierarquia são entidades denominadas "papéis". Assim, um sistema é formado por um conjunto de papéis. Cada papel é descrito pelas seguintes características:

- Responsabilidades, que determinam as funcionalidades do papel. As responsabilidades são ainda divididas em dois tipos: propriedades vitais, que descrevem as atitudes que o papel pode tomar; e propriedades de segurança, que

descrevem as atitudes de reação do agente quando algo diferente do previsto ocorrer;

- Permissões: São as habilidades que o papel possui, mais especificamente, são os recursos de informação que o agente possui;
- Protocolos: definem a maneira que um papel pode interagir com outros papéis.

3.5 Principais tipos de Agentes

Existem diversas classificações para agentes na literatura. Algumas que consideram a mobilidade do agente, outras que os classificam de acordo com seu sistema de raciocínio, dividindo-os em reativos ou deliberativos. Outros classificam os agentes de acordo com seu grau de autonomia, aprendizado e cooperação que possuem. Atualmente, podemos encontrar na literatura uma grande quantidade de aplicações que, de uma forma ou de outra, utilizam os conceitos de agentes para implementar uma ou outra funcionalidade. O que fica mais evidente nestas aplicações é o papel que o agente desempenha. Assim, para um melhor entendimento do texto e por considerarmos a classificação dos agentes baseada em seu papel em uma dada aplicação a mais adequada para o contexto desta tese, utilizaremos a classificação apresentada por NWANA (1997), que divide os agentes em:

- Agentes de Colaboração: Agentes de Colaboração enfatizam a autonomia e a cooperação com outros agentes para a realização de suas tarefas em um dado ambiente. Uma característica marcante deste tipo de agente é a sua capacidade de negociação para que possa realizar suas tarefas. Geralmente, o aprendizado não é uma característica marcante. Também tendem a ser estáticos. Possuem a capacidade de resolver os seguintes problemas: problemas que não podem ser resolvidos por um único agente central, por conta de recursos limitados ou pela natureza distribuída do problema; e a necessidade de interoperação com sistemas legados.
- Agentes de Interface: Enfatizam a autonomia e o aprendizado, sendo que seu principal objetivo é colaborar com o usuário na realização de uma determinada tarefa, agindo como um assistente pessoal. Este tipo de agente age observando e monitorando as ações do usuário, aprendendo com esta observação e sugerindo

novas maneiras para o usuário realizar suas tarefas. A cooperação com outros agentes é mínima, não significando que não possa existir.

- Agentes de Informação: Têm como objetivos o gerenciamento, manipulação e coleta de informações de fontes distribuídas. Atualmente, este tipo de agente está sendo muito utilizado por conta da explosão de informações na Internet. Podem ser móveis, mas geralmente são estáticos, colaborando com outros agentes para seu acesso à informação distribuída. Geralmente, possuem uma base de conhecimento estática que utilizam para realizar suas tarefas. Podemos subdividir os agentes de informação em: agentes de filtragem e agentes de recuperação.
- Agentes Móveis: São agentes capazes de se mover ao longo de uma rede local ou remota, como é o caso da Internet. Seu principal objetivo é a realização de tarefas remotamente, em nome de seus usuários. As características marcantes são a mobilidade e a autonomia para a realização de suas tarefas. Um exemplo de aplicação onde utiliza-se agentes móveis é o comércio eletrônico, onde o agente iria até o site remoto para cotar preços e só retornaria para o usuário as informações de mercadorias de interesse, minimizando desta forma o tráfego em rede;
- Agentes Híbridos: São agentes que englobam características de mais de um tipo acima. Geralmente, este tipo de agente é construído em camadas. Outra característica marcante é que, por ser uma abordagem híbrida, é comum adotarem o paradigma deliberativo e o reativo em conjunto.

Alguns autores (NWANA, 1997) denominam um MAS como sistema heterogêneo baseado em agentes. Neste tipo de sistema, agentes de diversos tipos se comunicam através de uma linguagem de comunicação de agentes ou similar. Atualmente, a maioria dos SBAs utiliza esta tecnologia. A interação entre os agentes se dá da seguinte forma: os próprios agentes tratam de coordenar suas interações ou se valem de um agente "facilitador" que tem o papel de coordenar os demais. Atualmente, a interação através de um facilitador é a mais utilizada.

Detalhamos nas seções seguintes agentes de interface e de informação pela sua importância no contexto desta tese. No capítulo 5 apresentamos um MAS que utiliza os

conceitos de agentes de interface e agentes de informação e por este motivo uma descrição mais detalhada destes tipos de agentes se faz necessária.

3.6 Agentes de Interface

Agentes de Interface são agentes capazes de capturar as necessidades dos usuários e traduzir estas necessidades em ações no sistema a ser utilizado (JANCA, GILBERT, 1997). De acordo com FLEMING e COHEN (1999), existem atualmente três tipos de agentes de interface: **agentes autônomos**, que são capazes de automatizar certas tarefas em nome do usuário e **agentes colaborativos**, que agem mais como colaboradores do usuário, trabalhando em conjunto com o usuário para alcançar um objetivo, utilizando um diálogo com o usuário para determinar o curso de ação apropriado, e **agentes híbridos**, são autônomos mas em alguns casos específicos passam a ser interativos, podendo solicitar a ajuda do usuário para melhorar sua performance. Os agentes autônomos também são conhecidos como agentes passivos e os colaborativos como ativos.

Os agentes autônomos são os mais utilizados atualmente, uma vez que agem de forma transparente ao usuário, produzindo assim menos ambiguidade. JANCA e GILBERT (1997) contra argumentam dizendo que o usuário tem mais dificuldade de confiar neste tipo de agente, uma vez que o usuário não interfere mais ativamente na forma como o agente age. No entanto, por agir desta forma, em sistemas complexos, onde a interferência do usuário deve ser minimizada, os agentes colaborativos vêm ganhando cada vez mais espaço.

JANCA e GILBERT (1997) classificam ainda como agentes de interface, agentes com interfaces para aplicações, que seriam agentes com a capacidade de se conectar com uma dada aplicação. Com isso, uma aplicação que não utiliza o paradigma de agentes, passa a ter esta capacidade. Para que esta conexão entre a aplicação e o agente possa ser realizada, é necessária a construção de um adaptador. Apesar desta categoria de agentes ser bastante interessante, não estamos considerando nesta classificação. Nossa classificação considera agentes de interfaces mais voltados para assistentes pessoais.

O grupo de estudos sobre agentes do MIT (MAES, 2000), desenvolveu um modelo que pode ser considerado genérico para o desenvolvimento de agentes autônomos. Este modelo prega que um agente autônomo deve agir primeiramente observando seus usuários, usando inferência baseada em memória como forma de aprendizado. Para cada nova situação que

surge, o agente calcula a distância entre o seu estado atual e as situações passadas que ele tem armazenadas, usando um somatório de características relevantes. De acordo com a ação tomada pelo usuário na maioria das situações passadas, o agente seleciona uma ação para a situação corrente e calcula um fator de confiabilidade no resultado.

FLEMING *et al.* (1999) identificaram alguns pontos de deficiência neste tipo de agente, a saber: i) o agente não lida muito bem com situações ambíguas; ii) a falta de comunicação entre o agente e o usuário cria uma certa dificuldade para que o usuário entenda o agente e confie no mesmo; iii) aprendizado baseado em memória pode ser lento, pois pode requerer consulta a um grande número de situações passadas. Além destes, podemos acrescentar outros pontos tais como se a base de fatos relativos a situações passadas não estiver disponível, como o agente deverá agir. A atuação do agente deve ser bem focada em um domínio particular, pois se o agente for agir de forma genérica, atendendo a qualquer tipo de usuário, ele pode não formar uma base de situações passadas significante que resulte em uma boa predição, levando assim a uma atuação pobre por parte do agente. Por outro lado, na abordagem pura de agente colaborativo, o agente sempre questiona o usuário em relação aos caminhos que ele pode seguir. Este tipo de atitude por parte do agente pode aborrecer o usuário. Com isso esta abordagem pura não é considerada muito adequada. Em uma abordagem híbrida, o agente continua sendo autônomo, independendo do retorno explícito do usuário. No entanto, quando alguma ambiguidade surge, ou não existe uma base de situações passadas que permita ao agente consultar, o usuário é solicitado. A informação solicitada pelo agente é respondida pelo usuário, passando então a figurar na memória do agente. Este novo modelo de agente de interface tem sido usado com sucesso em alguns projetos, conforme atesta os experimentos descritos em (FLEMING *et al.*, 1999).

Um aspecto interessante dos agentes de interface é a capacidade de adaptação da própria interface. Esta capacidade é denominada de interface adaptativa (STAFF, 1997). Segundo LINO (1999), as interfaces adaptativas são caracterizadas por sua capacidade de modificação em resposta às características ou necessidades do usuário identificadas pelo sistema. Assim, a adaptação tem como objetivo tornar a interface mais amigável ao usuário, facilitando sua utilização e melhorando o desempenho do usuário em relação às suas tarefas. Esta adaptação é, geralmente, realizada com base nas diferentes características dos usuários e de seus comportamentos que se modificam ao longo do tempo. Assim, este tipo

de sistema está intimamente ligado à modelagem e categorização de seus usuários, o que é realizado através de técnicas de modelagem do usuário.

Segundo BROWNE *et al.* (1990), existem três formas distintas de realizar a adaptação da interface:

- O agente de interface coleta informações sobre o usuário e se altera na mesma sessão ou entre sessões;
- O agente identifica o usuário como pertencente a uma categoria e determina as alterações de acordo com a categoria identificada;
- A interface não é alterada, mas ocorre uma alteração no desempenho, através de um tratamento de erros mais eficiente.

Uma outra forma de adaptação é descrita em (STAFF, 1997), onde a adaptação é realizada na navegação do usuário por um hipertexto. Assim, o sistema adaptativo guia o usuário pelo hipertexto, recomendando aquelas ligações que levarão o usuário para informações relevantes a sua tarefa, ou, em um modelo mais aprimorado, até mudando os links de destino. Em ambos os casos de adaptação, é necessária a criação de uma base de informações a respeito do usuário, seja através de um modelo que capture informações acerca das preferências do usuário, ou seja por conta de um modelo que capture os padrões de comportamento do usuário para navegação. Neste último caso, consideramos que os mecanismos estariam mais ligados a agentes de informação, que descreveremos na seção seguinte. Portanto, um modelo de usuário é uma característica importante para a especificação de um agente de interface. Na seção 3.8 apresentamos as técnicas na especificação de modelos do usuário.

3.6.1 Sistemas de Agente de Interface Existentes

Existem algumas aplicações baseadas em agentes de interface descritas na literatura. Todas são baseadas na construção de um modelo de usuário para realizar a adaptação da interface. O sistema AVANTI (FINK *et al.*, 1996) é um exemplo de uso de agentes de interface. O AVANTI realiza a adaptação de conteúdo e da apresentação de páginas HTML, de acordo com o grupo a que o usuário pertence e de suas próprias características. Este projeto é voltado para a apresentação de dados de uma área metropolitana, como os da polícia, hospitais, transportes, entre outros, para diversos tipos de usuários: turistas,

moradores, agentes de viagem e outros. As interfaces são adaptadas de acordo com o tipo de usuário que está fazendo a consulta. A adaptação é baseada em uma base de regras montada a partir de informações do usuário (modelo do usuário) e na navegação realizada pelo usuário.

Outro sistema que utiliza agentes de interface para adaptação é o HyperContext (STAFF, 1997). Este sistema baseia-se no conteúdo dos documentos a serem visitados e no comportamento do usuário durante a navegação, para realizar a adaptação da interface. Utiliza um modelo baseado em um único usuário.

Uma característica destes dois sistemas é que os mesmos são baseados em agentes autônomos, ou seja, agem por conta própria sem a ajuda do usuário. Com isso, quando existem ambigüidades ou situações de conflito, o agente tem que selecionar as opções por conta própria, sem o auxílio do usuário. Este tipo de atitude pode levar a um comportamento inadequado por parte do agente.

O sistema PUSH (ESPINOZA, HOOK, 1996) é um sistema para apresentação de grandes volumes de informação de acordo com as necessidades e preferências do usuário. Em relação ao agente de interface utilizado, o sistema já apresenta uma abordagem híbrida, uma vez que observa o comportamento do usuário e infere novos caminhos, mas também aceita o *feedback* do usuário em relação a alguma situação de conflito em que o agente possa estar. Este tipo de abordagem é considerada mais interessante atualmente, uma vez que mescla a autonomia do agente com a colaboração do usuário em situações nas quais é difícil para o agente resolver.

3.7 Agentes de Informação

O armazenamento e recuperação de dados por pessoas e organizações passou por modificações substanciais ao longo dos últimos anos até a recente disseminação da Internet, o que transformou a visão clássica de base de dados, de uma coleção centralizada de dados homogêneos em coleções distribuídas e heterogêneas. Um dado não é mais uma entidade estruturada e pode ser quase qualquer tipo de objeto representável, por exemplo, um desenho, um som, um texto, um registro, etc (PALAZZO, 1996). A disseminação das redes de computadores tornou informações deste tipo disponíveis a qualquer um. Em conseqüência, um número muito grande de repositórios de informações se encontra à disposição de todos, ainda que apenas uma quantidade restrita desta informação seja de

interesse para um determinado usuário. Assim, atualmente, são necessárias técnicas cada vez mais sofisticadas para a recuperação destas informações. Em particular, são utilizadas técnicas oriundas da área de recuperação de informação (BAESA-YATES, RIBEIRO-NETO, 1999) e recuperação de informações em bases de dados distribuídas, como é o caso, por exemplo, da arquitetura de mediadores (WIEDELHOLD, 1995).

Até bem pouco tempo atrás, a área de recuperação de informação era mais ligada à recuperação de textos não estruturados, geralmente não armazenados em bases de dados. Devido a distribuição e heterogeneidade das informações, atualmente as técnicas de IR podem e devem ser aplicadas a qualquer tipo de dado, esteja ele armazenado onde estiver (BAESA-YATES, RIBEIRO-NETO, 1999). Analisando as necessidades dos sistemas de IR, os pesquisadores apontam a tecnologia de agentes como promissora para lidar com os problemas detectados, como veremos a seguir.

FININ *et al.* (1998) apresentam uma lista de dez problemas atualmente encontrados em sistemas de IR: 1) Necessidade de soluções integradas; 2) Recuperação de informação distribuída; 3) Indexação e recuperação eficiente e flexível; 4) Expansão de termos para expandir ou refinar a busca; 5) Interfaces e navegação; 6) Filtragem e roteamento; 7) Recuperação eficiente; 8) Recuperação de dados multimídia; 9) Extração de informação; 10) Retorno do Usuário (*Relevance Feedback)* (busca de acordo com as preferências do usuário). Estes autores ressaltam que a maioria destes problemas podem ser resolvidos através da tecnologia de agentes de informação. Particularmente, pela literatura disponível em agentes de informação, identificamos que agentes de informação são capazes de resolver os problemas 1,2,4,5,6,7,9 e 10. Os problemas 3 e 8 e, parcialmente, o 4 também poderiam ser resolvidos mas com o acoplamento de novas funcionalidades.

Um agente de informação que tenha como objetivo resolver os problemas listados acima, deve prioritariamente ser capaz de:

- Se adaptar a seus usuários e ao conteúdo a ser disponibilizado;
- Possuir uma única interface para acesso a múltiplos repositórios de dados.

Entendemos ainda que um agente de informação deve ser capaz de localizar, recuperar e integrar informações. E, ainda, procurar por informações de maneira pró-ativa em fontes de dados distribuídas, evitando intervenções do usuário, sempre que possível.

Além disso, o agente só deve disponibilizar informações que sejam efetivamente do interesse do usuário. PAEPCKE *et al.* (1999) denominam este tipo de agente de "coletor de informações de valor", onde o valor da informação está relacionado a sua relevância.

Logo, para que um agente de informação seja capaz de executar todas estas tarefas, ele deve prover duas características básicas:

- Filtragem de informação;
- Recuperação de informação.

A Figura 3.1 a seguir, adaptada de PAEPCKE *et al.* (1999), apresenta como seria uma arquitetura básica de um sistema de IR.

Atualmente, as técnicas tanto para filtragem quanto para recuperação estão cada vez mais sofisticadas. Assim, um agente de informação pode ser visto como um MAS, composto de dois tipos de agentes: agente de filtragem e agente de recuperação.

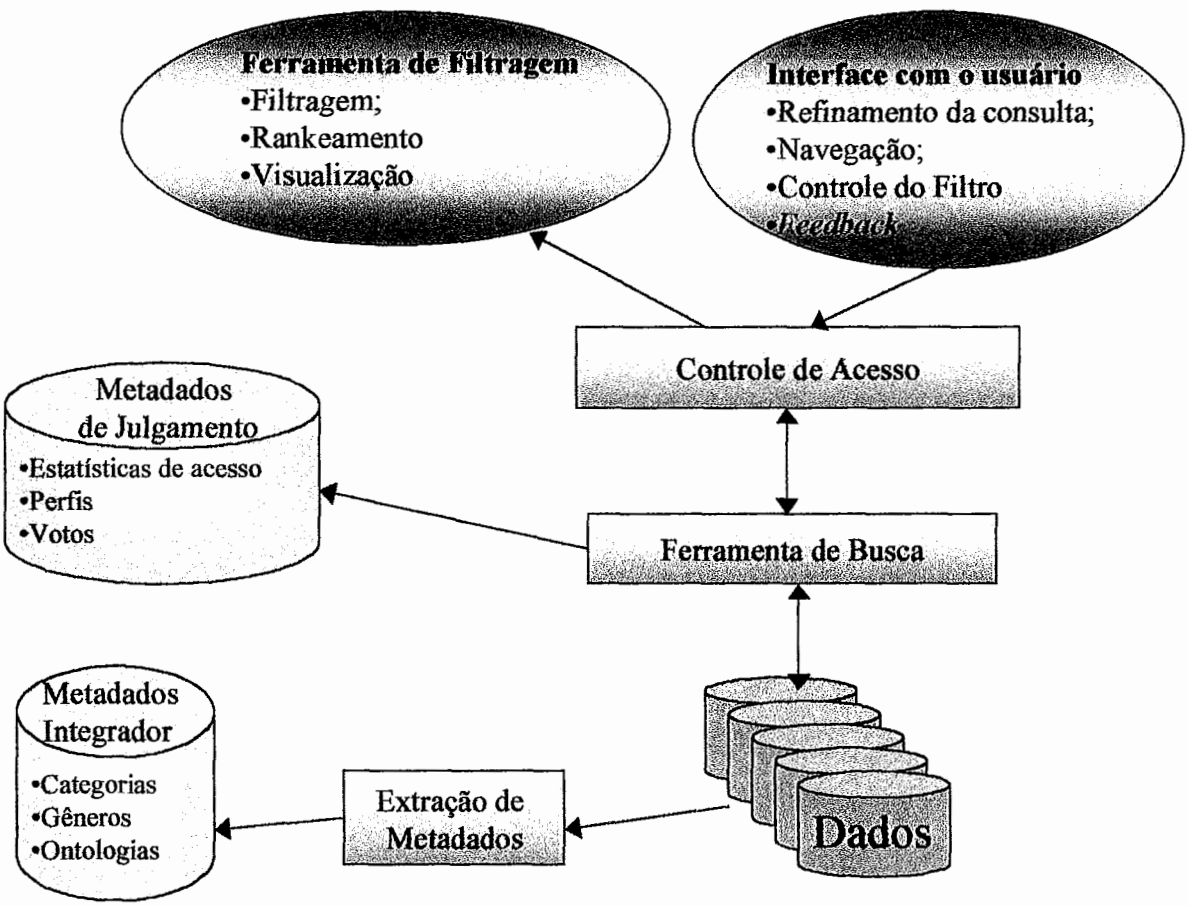

Figura 3.1 – Arquitetura básica de um sistema de IR (adaptada de PAEPCKE, 1999)

3.7.1 Agente de Filtragem

Sistemas de filtragem de informação, segundo MOUKAS (1997), têm como principal objetivo descartar informações que não atendam às especificidades do filtro. Assim, fica claro que não é tarefa de um filtro a localização dos dados. Esta localização é responsabilidade de um sistema de recuperação de informação.

As primeiras técnicas de filtragem eram baseadas na interação explícita do usuário, ou seja, os usuários avaliavam a informação recuperada e o sistema de filtragem se baseava nesta avaliação para buscas futuras. Mais recentemente, começaram a surgir técnicas com uma proposta mais automatizada para esta filtragem. Algumas abordagens analisam a estrutura da coleção, como as ligações entre documentos na Internet, considerando mais relevante aquele com maior número de *links* que apontam para ele. Outras técnicas observam como os usuários acessam as coleções, usando por exemplo, modelos de interesses dos usuários, ou assumindo que documentos mais acessados são os mais adequados (PAEPCKE *et al.*, 1999).

As técnicas de filtragem de informação são utilizadas para vários propósitos, seja para selecionar documentos recuperados de uma ou mais bases de dados; para guiar usuários quando os mesmos estão navegando por uma rede de informações, ou para disseminar informações de maneira seletiva. Este último item é considerado promissor e é chamado por muitos de "informação sob demanda", que seria a distribuição de informação de maneira seletiva e sem aborrecer os usuários. Atualmente, um dos itens considerados desejável em um banco de dados é um mecanismo que avise aos usuários interessados da chegada de novas informações ou mudanças ocorridas, sem que ele tenha que explicitamente requisitá-las (YEE *et al.*, 2000).

MOUKAS (1997) e PAEPCKE *et al.* (1999) descrevem três categorias principais de sistemas de filtragem de informação, a saber:

1. Filtragem baseada em conteúdo: Tem como objetivo a extração de informações úteis a partir do conteúdo de documentos. Utiliza técnicas de extração de metadados para coletar ou deduzir metadados a respeito das coleções ou documentos. Na Figura 3.1 esta categoria está ligada à formação do repositório de metadados integrador. As técnicas baseadas em conteúdo podem ainda ser divididas em: i) Análise dos dados: os metadados são derivados através da análise de dados (documentos) individuais em uma

coleção. Para isso, utiliza-se algoritmos chamados popularmente de "caça-palavras" e algumas variações destes. ii) Análise de coleção: Esta técnica não analisa um documento ou dado particular dentro da coleção, mas sim todos os dados da coleção e suas ligações. Neste contexto estão os algoritmos que analisam o número de ligações que apontam para um dado documento na Internet. iii) Análise do contexto da informação: Determina o contexto onde a informação está situada. O contexto pode ser desde a instituição de origem da informação, passando pela data da informação até o assunto de que trata a informação. Uma vez determinado este contexto, os documentos são classificados em grupos (*clustering technique*). iv) Análise baseada em *tags* contidas nos documentos: Esta categoria é baseada na intervenção humana em criar *tags* nos documentos para que depois possam ser analisados com o objetivo de determinar o valor do documento em um dado contexto. Algumas abordagens utilizam mecanismos de indexação de documentos HTML, XML, baseando-se nos seus respectivos *tags* HMTL.

2. Filtragem Social, também denominada de filtragem colaborativa ou baseada em ação. O sistema de filtragem utiliza respostas do usuário e classificação por diferentes usuários para filtrar as informações. Este tipo de sistema não se preocupa em analisar ou entender o conteúdo do documento. Esta abordagem usa as impressões do usuário sobre o documento para classificá-lo em relação a um dado usuário. Este índice gerado é individual para cada usuário e pode ser utilizado por usuários com interesses similares. A diferença entre as abordagens de filtragem social são baseadas em se é realizado ou não um julgamento explícito por parte do usuário. Esta categoria gera, na Figura 3.1 os metadados de julgamento. Assim, podemos dividir as técnicas em duas categorias: as que usam julgamento explícito e as que usam julgamento implícito.
   2.1. Julgamento Explícito: Nesta categoria, os usuários avaliam explicitamente documentos e coleções. Este julgamento explícito é a origem do termo retorno do usuário (*relevance feedback)*, onde uma busca é realizada e o usuário, de maneira explícita, indica qual a opção mais relevante. A partir desta escolha, as próximas buscas serão refinadas. Podemos ainda dividir as técnicas de julgamento explícito em: i) Filtragem baseada em retorno do usuário: Nesta categoria se encaixa o retorno do usuário baseado em um único usuário e o retorno do usuário

colaborativo, onde são considerados os julgamentos de um grupo de pessoas e não de um único usuário. ii) Filtragem baseada em expressões: Este tipo de filtragem é baseado na construção de uma expressão que irá filtrar as informações. Sistemas de mensagens de correio eletrônico utilizam muito este tipo de filtro. iii) Filtros Sintetizadores: É similar aos filtros baseados em expressões. No entanto, utiliza uma abordagem com menos intervenção por parte do usuário. O usuário apenas define a tarefa que deseja realizar e o mecanismo de filtragem irá escolher entre as tecnologias de filtragem qual a melhor para aquele caso específico.

2.2. Julgamento Implícito: A crítica em relação ao julgamento explícito é que o filtro requer a participação do usuário. Para tentar resolver este problema foram criados filtros com julgamento implícito, usando uma maneira mais automatizada de derivar estas informações. Geralmente, este tipo de filtro se vale da observação das ações do usuário. Podemos ainda subdividir as técnicas usadas nesta categoria em duas: i) Julgamento a partir do comportamento de um conjunto de usuários: Geralmente, esta abordagem utiliza históricos de acesso, gerando estatísticas a respeito dos acessos. Assim, os documentos mais acessados são considerados os de maior valor. ii) Julgamento a partir de um único usuário: A grande vantagem desta abordagem em comparação com a anterior é que na anterior o filtro funciona apenas se existir um julgamento prévio de outros usuários e nesta não é necessário um julgamento prévio. O julgamento é feito observando quais documentos o usuário acessa e observações a respeito do que o usuário faz com os documentos que ele acessa.

A utilização destas técnicas na filtragem de informação é ainda um assunto novo, apesar de já existirem diversos aplicativos desenvolvidos, sendo sua utilização na prática ainda pequena. No entanto, apesar da pouca experiência com estas novas técnicas para filtragem de informação, já existem algumas abordagens (FLEMING *et al.*, 1999) que parecem demonstrar a relativa eficácia na combinação das técnicas baseadas em conteúdo, julgamento implícito e explícito. O que se deve levar em conta é o contexto e o propósito do filtro.

Outro problema levantado por LINTON (1999), e até agora não muito bem resolvido é a necessidade de uma representação semântica mais precisa para a busca baseada em conteúdo. Uma solução que se apresenta é a proposição de padrões para metadados na Internet como o XML e RDF. No entanto, para que estes padrões sejam realmente efetivos, é necessário o desenvolvimento de ontologias que utilizem estas representações, ou seja, o desenvolvimento de uma rede semântica relacionada ao contexto ou domínio onde se encaixa a informação.

3.7.2 Sistemas de Agentes de Filtragem Existentes

Na literatura encontramos diversos sistemas de filtragem de informação, principalmente voltados para Internet. O que os diferencia é a maneira como realizam a filtragem de informação. Cada um deles utiliza um tipo de técnica e algoritmo para realizar a filtragem.

O sistema LIRA (BALABANOVIC, SHOHAM, 1995) realiza uma busca por documentos de interesse para um dado usuário na Internet. Esta busca é baseada no conteúdo dos documentos e nas preferências dos usuários. É utilizado um algoritmo que classifica os documentos pelo radical das palavras encontradas no mesmo. Além disso, o usuário pode avaliar os documentos explicitamente através de notas. O Letizia (LIEBERMAN, 1995) também é um sistema que auxilia o usuário na navegação web, indicando documentos de interesse para o usuário baseado no conteúdo dos mesmos e no contexto que se encontram.

O sistema Amathea (MOUKAS, 1997) utiliza uma abordagem de múltiplos agentes para busca e filtragem de informação na Internet. O Amathea baseia-se na busca realizada por mecanismos de busca convencionais como AltaVista, Yahoo, entre outros e depois filtra a informação retornada pelos mesmos. Para a filtragem da informação, utiliza técnicas de aprendizado de máquina. Outros sistemas que seguem a mesma linha são Webmate (CHEN, SYCARA, 1998), Arachnid (MENCZER, BELEW, 1998), Webface (HAN *et al.*, 1998) e Webwatcher (ARMSTRONG *et al.*, 1995), variando na forma como montam o perfil do usuário e classificam os documentos.

Analisando estes sistemas, podemos concluir que a abordagem de análise do conteúdo dos documentos é a abordagem mais usada para a filtragem, utilizando técnicas de aprendizado de máquina. Alguns pesquisadores (FRAGOUDIS, 1999) argumentam que

o uso de técnicas de aprendizado de máquina somente são adequadas, nestes casos de filtragem de informação na WEB, quando os interesses dos usuários são constantes por um longo período de tempo, pois caso contrário não se tem tempo hábil para montar dados de treinamento, necessários para o uso de algoritmos de aprendizado de máquina, realmente efetivos. Assim, para que estes mecanismos de busca sejam realmente efetivos, os mesmos devem ser particularizados por grupos de interesse ou domínios de interesse, pois neste caso, os dados de treinamento serão efetivos.

Alguns sistemas, como Amathea, WebMate, Webface utilizam mecanismos de busca convencionais para realizar a recuperação dos documentos a serem filtrados. Os outros sistemas realizam a busca por si próprios. A vantagem de se utilizar mecanismos de busca convencionais para a recuperação de documentos é que estes mecanismos mantêm um índice grande de links a serem acessados. No entanto, esta busca pode não ser tão efetiva, uma vez que a forma de se encontrar os documentos é através de algoritmos simples de "caça-palavras" e, mesmo posteriormente sendo realizada a filtragem, documentos realmente interessantes podem não ser apresentados.

Em um contexto diferente, existem também algumas propostas interessantes. Em Motta (MOTTA, 1999) é apresentado um ambiente de recomendação e filtragem da informação para apoio a equipes de trabalho baseado no enfoque cooperativo. A filtragem de informação é baseada na Teoria da Situação e Lógica do Fluxo de Informação. A abordagem utiliza técnicas interessantes para a filtragem de informação, principalmente levando em conta as particularidades do trabalho cooperativo, o que não é contemplado pela maioria dos trabalhos.

O uso de mecanismos de busca mais particularizados por área de interesse, onde possa se ter um índice de ligações interessantes para um dado contexto parece ser adequado, sendo estas ligações adicionadas por especialistas no domínio da aplicação. Uma limitação deste tipo de mecanismo é que como depende de uma atividade prévia de publicação de links por especialistas, se estes não quiserem publicar seus dados, o mecanismo de filtragem não terá como atuar sobre os mesmos.

3.7.3 Agentes de Recuperação

Um sistema de recuperação de informação, de forma geral, tem como objetivo principal a busca de informações muitas vezes imprecisas e relacionadas, estejam elas onde

estiverem. Um sistema de recuperação de dados, conforme descrito na introdução desta tese, é geralmente, ligado a uma base de dados homogênea, que é acessada para a recuperação de seus dados através de técnicas clássicas de Sistemas de Banco de Dados, como, por exemplo, uma linguagem de consulta baseada em SQL. Até bem pouco tempo atrás, quando se tinha que acessar várias bases de dados, este acesso era feito separadamente, com consultas específicas para cada uma das bases, de acordo com seus metadados. Este acesso era normalmente realizado de forma manual, onde um administrador de dados identificava possíveis fontes de dados e formulava consultas para cada uma delas. De posse dos dados retornados por cada uma das bases, o administrador era responsável pela integração dos dados provenientes das várias fontes, corrigindo possíveis inconsistências.

Atualmente, com o aperfeiçoamento das tecnologias de comunicação, o surgimento da Internet e o grande volume de informações disponíveis, os sistemas de recuperação de dados evoluíram e já permitem a recuperação de dados distribuídos e heterogêneos de forma transparente para o usuário. Além disso, a possibilidade de recuperação de dados relacionados e com um certo grau de incerteza aproximam estes sistemas dos sistemas de recuperação de informação (BAESA-YATES, RIBEIRO-NETO, 1999). No entanto, a maioria dos sistemas existentes ainda está em fase de evolução e ainda não existe um consenso de qual a melhor tecnologia a ser utilizada (KANTORSKI, 2000). Segundo ainda KANTORSHI (2000), existem enfoques e abordagens distintos para o acesso integrado a informações provenientes de banco de dados autônomos, localizados em ambientes diferentes de hardware e software, como os propostos em sistemas de banco de dados federados (SHETH *et al.*, 1990), sistemas *multidatabases* (BRIGHT *et al.*, 1992), sistemas de banco de dados heterogêneos (CHUNG, 1990) e mediadores (WIERDERHOLD, 1995). Estes enfoques variam, basicamente, no que se refere à existência (STRAUCH, 1998) ou não de um esquema global de metadados, onde toda a informação disponível é mapeada. As abordagens que utilizam o esquema conceitual global baseiam-se na comparação dos diversos esquemas conceituais dos bancos de dados envolvidos, na identificação de equivalências e conflitos de representações, e na eliminação das diferenças de representação adotadas localmente.

A solução baseada em mediadores propõe a criação de módulos de software que ocupam um nível explícito e ativo entre aplicações de usuários e recursos de dados. Contudo, uma das críticas que se faz ao uso de mediadores é que, por formarem uma camada adicional entre aplicações e base de dados, podem apresentar problemas de perda de desempenho com relação a aspectos de implementação. No entanto, quando é necessária uma transparência em relação à distribuição e heterogeneidade dos dados, a tecnologia de mediadores se apresenta bastante interessante, propondo um mecanismo efetivo para o acesso a dados heterogêneos e distribuídos a partir de um modelo comum, que é o modelo integrador (metadados) do mediador. Alguns autores argumentam que estes mecanismos integradores apresentam uma visão limitadora dos dados e que podem existir casos onde a identificação da origem das informações e a perda causada por algum aspecto da consulta seja explicitada. Uma maneira de minimizar os impactos destas perdas é apresentá-las, explicitamente para o usuário.

Atualmente, devido ao dinamismo necessário às aplicações existentes, os repositórios de informação estão em constante evolução. Assim, a tecnologia de mediadores apresenta uma abordagem interessante, uma vez que permite a adição ou retirada de fontes de informação sem comprometer a estrutura como um todo (FINI *et al.*, 1998). A seguir, apresentamos a tecnologia de mediação com mais detalhes. Dois conceitos que consideramos importantes em uma estrutura de mediação são: os metadados para a recuperação das informações distribuídas e heterogêneas e o mecanismo de integração das diversas camadas da estrutura de mediação. Descrevemos nas seções seguintes o estado da arte nestas duas tecnologias, ou seja, o uso de ontologias como metadados e o uso de padrões de interoperabilidade.

3.7.3.1 Mediadores

Levando em consideração as necessidades de acesso às informações por parte de um agente de recuperação de informações, devemos considerar como uma possível solução a utilização de uma camada que permita a integração de diferentes bases de informações, distribuídas e heterogêneas, de forma que as informações recuperadas sejam apresentadas ao agente de filtragem de forma consistente e em um formato adequado, sendo o agente de filtragem somente responsável pela filtragem da informação. Uma camada como esta é denominada por Wiederhold de "mediador inteligente", ou seja, uma camada que faz a

mediação entre as bases distribuídas e com modelos de dados heterogêneos, utilizando metadados e mecanismos de interoperação semanticamente ricos (WIDERHOLD, 1995). Mediadores para a recuperação de informações são considerados por alguns pesquisadores (WIDERHOLD, 1997) (BAYARDO, 1997) como agentes inteligentes para recuperação de informação. A característica principal que permite que a abordagem seja considerada uma abordagem relacionada a agentes, estaria justamente no uso de metadados sofisticados para recuperação e sua capacidade de interoperação.

Os primeiros mediadores que surgiram foram especificados para aplicações específicas ou para prover a extensão de serviços de banco de dados. Atualmente, o conceito de mediadores evoluiu, segundo atesta o próprio WIEDERHOLD (1997), para o conceito de mediadores inteligentes, cujo principal objetivo é a aquisição de conhecimento sobre determinado domínio de aplicação e a posterior manipulação das informações encontradas. O uso de mediadores inteligentes é amplo, sendo principalmente um tópico de pesquisa na busca de informações na Internet (WIEDERHOLD, 1999). Seu aspecto inovador consiste no particionamento das informações por domínio e no uso de ontologias para a busca de informações armazenadas nos mais diferentes formatos. Esta forma de busca, segundo estudos (BERNAKI, 1997) (WIEDERHOLD, 1997), é muito mais eficiente do que mecanismos como os utilizados pelos sistemas de busca na Internet atuais, tais como o Alta Vista, Lycos, entre outros. Sendo assim, a principal função do mediador é prover informação integrada, sem a necessidade de integrar as fontes de dados. Para isso, as principais tarefas que devem ser realizadas por um mediador são (BAYARDO, 1997):

- acessar e disponibilizar dados relevantes de fontes de dados heterogêneas;
- abstrair e transformar os dados disponibilizados para uma representação e semântica comum;
- integrar os dados de acordo com a busca realizada;
- reduzir os dados integrados, através de abstração, de forma a aumentar a densidade das informações no resultado a ser transmitido.

Para isso, a arquitetura de um mediador é basicamente constituída de três camadas (Figura 3.2):

a) as aplicações que estão buscando informações relevantes;

b) os módulos de mediação;

c) e as fontes de informação, como base de dados, bases de conhecimento, sistemas de arquivos, servidores www, entre outros.

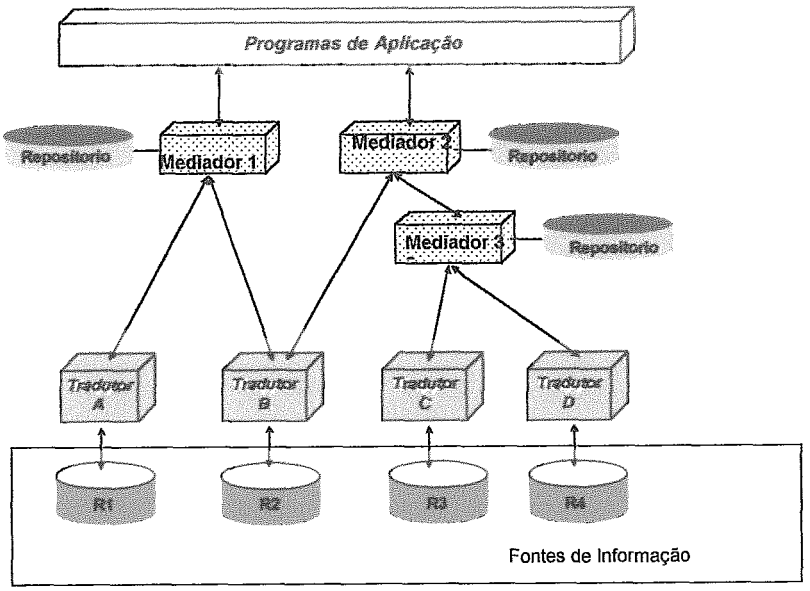

Figura 3.2 - Estrutura de mediação (adaptado de (PIRES, 1997))

A Figura 3.2 apresenta a arquitetura básica de uma estrutura de mediação. As aplicações acessam as fontes de dados através dos mediadores. Os tradutores são módulos de software que controlam e facilitam o acesso às fontes de dados. Os tradutores são responsáveis pela ligação do modelo agregado das fontes de dados (produtores) com o modelo nativo da base de dados. Se o modelo nativo for compatível com o modelo do mediador, o acesso pode ser direto, sem a necessidade de nenhum tradutor. Exemplo desta possibilidade é a utilização de um modelo de objetos compatível com o padrão ODMG para a mediação e o acesso a um SGBDOO compatível com ODMG. Entre estas camadas, existem duas interfaces principais:

a) a interface entre as aplicações e os módulos de mediação;

b) a interface entre as fontes de informação e os módulos de mediação.

Uma preocupação crescente na implementação dos mediadores atuais é a necessidade de padronização das estruturas utilizadas, o que facilita sua posterior extensão. Assim, a especificação de mediadores passa pela utilização de componentes padronizados

na especificação de suas estruturas e pela especificação de uma arquitetura aberta, que possa ser customizada. O uso de padrões como CORBA, para o acesso a dados distribuídos, e um modelo de objetos compatível com o padrão, como é o caso do modelo ODMG 97 (www.odmg.org), facilitam a especificação das estruturas dos mediadores. Outra característica importante na arquitetura de um mediador é a sua escalabilidade, ou seja, a capacidade que o mediador possui de agregar novas fontes de informação, sem que isso prejudique o funcionamento da estrutura como um todo. A adição de novas fontes de informação pode requerer apenas que novos módulos de mediação sejam agregados à estrutura.

Um conceito importante na estrutura dos mediadores é o de modelos que permitirão a troca de dados entre as aplicações (consumidores) e as fontes de dados (produtores) (WIEDERHOLD, 1997). Os modelos de dados dos produtores são os modelos das fontes de dados que serão agregadas ao sistema. Como exemplo dos modelos de dados dos produtores temos: modelos relacionais, modelos OO, modelos textuais e multimídia (ex. documentos HTML), modelos baseados em regras, entre outros. No entanto, para que a estrutura de mediação possa utilizar os dados armazenados nestas fontes de dados heterogêneas, devemos gerar um modelo unificado dos produtores, que permita a consulta às bases de dados. Wiederhold propõe um modelo agregado dos produtores na forma de um hipergrafo, que, segundo o autor, abrangeria todos os outros modelos em relação à representação semântica dos mesmos. Assim, este modelo, denominado modelo do consumidor, deve ser baseado em alguma modelagem padronizada que permita o fácil entendimento por parte do usuário e permita também a representação, de forma mais abstrata possível, de todo o conhecimento relevante ao consumidor. Neste contexto, padrões de modelagem como o do ODMG 97 podem ser utilizados, levando em consideração as adaptações necessárias. Nesta linha, temos a proposta do projeto DIOM (LIU *et al.*, 1996), que se baseia no modelo ODMG 93. Outras abordagens já utilizam o padrão XML para a modelagem do consumidor (XHUMARI *et al.* , 1999).

No entanto, a medida que novas fontes de informação vão sendo agregadas à estrutura de mediação, a quantidade de informação a ser modelada aumenta, gerando inconsistências, ambigüidades e conflitos na informação representada. Uma das formas para se resolver este problema é particionar os modelos de consumidores e produtores a

serem utilizados e a estrutura de mediação por domínios de aplicação. Como principais vantagens deste particionamento por domínios, temos:

- melhor definição, sem ambigüidades, dos modelos a serem utilizados por cada um dos domínios;
- divisão de responsabilidades na manutenção dos modelos;
- diminuição na quantidade de informações a serem analisadas a cada requisição de um usuário, uma vez que as informações a serem analisadas são apenas aquelas específicas àquele domínio.

A estes modelos, particionados por domínio, alguns autores dão o nome de ontologias sobre um dado domínio (WIEDERHOLD, 1997). Outra questão importante, em relação ao particionamento dos mediadores por domínios, é que muitas aplicações podem necessitar de informações de mais de um domínio de aplicação. Neste caso, é necessário que os modelos dos diferentes domínios sejam integrados. Esta integração de modelos não é uma tarefa fácil de ser realizada, existindo apenas algumas propostas para esta integração, como as de (BENAKI, 1997), (MENA, 1998) e a de WIEDERHOLD, através da especificação do que é denominado álgebra DKB (WIEDERHOLD, 1994), através da qual algumas operações básicas como interseção, união, diferença e mapeamento permitem a integração de modelos (ontologias) diferentes, partindo do pressuposto de que nenhum termo significa a mesma coisa que outro, a não ser que existam "regras de semelhança" que digam explicitamente isso.

3.7.3.2 Uso de Ontologias para a Recuperação de Informações

O uso da tecnologia de mediadores pressupõe a adoção de um modelo integrador que permita a recuperação das informações das bases componentes. Atualmente, existe um grande interesse no desenvolvimento de ontologias para a integração de bases de dados (MENA, 1998) (WIEDERHOLD, 1999) (BAYARDO, 1997). A idéia é que o sistema de mediação seja acessado através das ontologias. Estas ontologias são descrições abstratas de conceitos em um dado domínio de aplicação. Assim, fica muito mais fácil para o usuário a identificação de conceitos já conhecidos no domínio de aplicação do que lidar com as especificidades dos repositórios de dados. O sistema tem a responsabilidade de mapear a heterogeneidade e distribuição dos dados nos repositórios para os conceitos ontológicos.

Neste contexto, uma ontologia pode ser definida como um vocabulário de termos e a especificação do relacionamento entre eles (WIEDERHOLD, 1995). Para cada termo, uma definição deve ser criada, através de uma descrição informal, agregando inclusive alguns exemplos e também uma especificação formal dos relacionamentos entre os termos, formando assim uma rede semântica. Estas ontologias devem ser criadas por especialistas no domínio de aplicação, de modo que reflita a conceituação utilizada daquele domínio. Segundo MENA (1998), uma ontologia são metadados semanticamente ricos, permitindo a descrição das informações de maneira mais precisa e facilitando a formulação de consulta aos dados. Assim, cada termo em uma ontologia utilizada no contexto de recuperação de informação, deve ter associada uma informação de mapeamento que relacione o termo com os dados nos repositórios. Estes mapeamentos é que guiam a recuperação de informações relacionadas ao termo. Não é nosso objetivo apresentar uma descrição do conceito de ontologia em um contexto mais amplo. Nosso interesse se resume a sua utilização no contexto de recuperação de informação. Neste sentido, a descrição acima para este contexto é suficiente. Para o leitor interessado em uma descrição mais detalhada, é indicado consultar (GUARINO, 1998).

3.7.3.3 Mecanismos de Interoperabilidade

O uso de mediadores deve permitir a participação de repositórios implementados fisicamente por hardware e software com características distintas dos demais, ligados por canais de comunicação e protocolos distintos, além de possuírem diferentes controles operacionais. Esta funcionalidade traz à tona alguns problemas. Sistemas operacionais e máquinas diferentes podem utilizar diferentes formas de representação para os dados (por exemplo, diferentes representações para números de ponto flutuante). Mecanismos distintos de comunicação necessitam de *gateways* que podem ser de difícil implementação, devido às diferentes funcionalidades envolvidas. Diferentes fabricantes de repositórios podem utilizar linguagens de manipulação e definição distintas, mesmo adotando o mesmo modelo teórico de dados (PIRES *et al.*, 1997).

Atualmente, os padrões para sistemas abertos e distribuídos visam solucionar estes problemas, possibilitando a interoperabilidade entre sistemas. Vários grupos, governamentais e não governamentais, têm trabalhado nesta direção. Como exemplo podemos citar o trabalho *Object Management Group* (OMG), através da especificação do

modelo de objetos CORBA (www.omg.org), o modelo de interoperabilidade para componentes Java, o RMI (*Remote Invocation Method*) (www.java.sun.com), e o modelo de objetos COM, da Microsoft (www.microsoft.com). Cada uma das abordagens possuem prós e contras, sendo que a especificação CORBA, é a que abrange mais aspectos relacionados a como deve ser idealmente um modelo de objetos.

O OMG é um consórcio internacional da indústria de software com dois objetivos principais (PIRES *ET AL.*, 1997):

1. promover a utilização do paradigma de orientação a objetos na engenharia de software em geral e
2. desenvolver modelos e padronizar interfaces para o desenvolvimento e uso de aplicações distribuídas de larga escala utilizando a metodologia de orientação a objetos.

De acordo com (www.omg.org) CORBA é o núcleo do mecanismo de comunicação que permite a operação entre objetos. CORBA é a especificação de uma arquitetura e interface que permite que aplicações façam solicitações a objetos, de uma forma transparente e independente, indiferente à linguagem, sistema operacional ou considerações de localização. Sendo assim, o CORBA pode ser visto como um paradigma de comunicação, baseado na tecnologia de orientação a objetos para ambientes de computação distribuídos, sendo que atualmente existem várias implementações disponíveis no mercado, como, por exemplo, VisiBroker (www.inprise.com) e Orbix.

CORBA utiliza um modelo orientado a objetos e define um módulo intermediário entre clientes e servidores, o *Object Request Broker* (ORB). Neste modelo clientes enviam requisições para o ORB solicitando algum serviço para ser executado por qualquer servidor que possa responder a suas requisições. Somente o ORB necessita conhecer a localização dos clientes e servidores na rede, com isso a localização do servidor é transparente para o cliente e por sua vez, a do cliente para o servidor. Uma biblioteca de funções chamada de *Basic Object Adapter* (BOA), é utilizada pelo programa do servidor para localizar e inicializar as implementações e invocar o método apropriado para responder à requisição do cliente.

Modelos de interoperabilidade como CORBA são interessantes e bem aceitos pela comunidade acadêmica em banco de dados como um mecanismo de interoperabilidade a

ser utilizado para a ligação de bases distribuídas e heterogêneas. No entanto, os pesquisadores em agentes argumentam que a interoperabilidade disponibilizada por mecanismos como CORBA somente se preocupam com o nível sintático e, na tecnologia de agentes uma comunicação mais efetiva é necessária a nível semântico. O uso de linguagens de comunicação entre agentes (ADL) é um caminho apontado pelos pesquisadores. O próprio OMG reconhece isso e já propõe alguma evolução neste sentido, acoplada a tecnologia CORBA (www.omg.org, 2000). O uso de ontologias também é um caminho promissor apontado por diversos pesquisadores (BAYARDO, 1997) (HUHNS, SINGH, 1997).

Desta forma uma possível evolução destes mecanismos de interoperabilidade é no sentido de acoplar mais semântica, de uma forma ou de outra, às suas definições. Um agente de informação que seja composto de um agente de filtragem e agente de recuperação como descrito, utilizando perfis do usuário, ontologias e acesso a dados distribuídos e heterogêneos parece ser o caminho para a especificação de um agente de informação completo. No entanto, ainda há a necessidade de uma maior padronização na área, com o uso disseminado de ontologias e o uso de representações como XML e/ou RDF (FININ, 1998), uso de mecanismos de comunicação entre agente de filtragem e recuperação semanticamente mais rico e recuperação eficiente de dados multimídia. Além disso, é necessária uma maior experiência neste tipo de agente, uma vez que ainda existem poucas aplicações efetivamente testadas. Neste contexto, é importante ressaltar a necessidade de melhoria de desempenho das aplicações desenvolvidas.

3.7.3.4 Sistemas Existentes

Apresentamos a seguir alguns sistemas existentes que utilizam a tecnologia de mediação para a recuperação de dados distribuídos e heterogêneos. Em uma abordagem mais genérica do uso de mediadores, temos os sistemas HERMES, TSIMMIS, HEROS e a arquitetura baseada em mediadores desenvolvida em (PEREIRA, 1999). SUBRAHMANIAN *et al.* (1999) desenvolveram o sistema HERMES *(Heterogeneous Reasoning and Mediator Syste*m), que utiliza mediadores como ponto de partida para a integração de informações localizadas em fontes heterogêneas. Uma limitação deste modelo é, no caso de fontes estruturadas, a implementação para bancos de dados específicos, como DBASE, INGRES e PARADOX. Além de ser uma solução limitada a

um conjunto restrito de bancos de dados, exige do usuário conhecimento específico sobre o domínio (banco de dados) onde está sendo realizada a consulta, o que pode ser bastante indesejável. O sistemas HEROS (UCHÔA, MELO, 1999) também se baseia em mediadores, sendo desenvolvido como um framework para a integração de banco de dados heterogêneos. O interessante nesta abordagem é que este framework pode ser adaptado a sistemas diferentes. O projeto, chamado *The Stanford-IBM Manager of Multiple Information Sources* - TSIMMIS (CHAWATHE et al., 1994), é voltado para a integração de informações de fontes heterogêneas, incluindo dados estruturados e não estruturados. Uma limitação desta ferramenta é que ela assume que o usuário do sistema é capaz de entender tanto da linguagem SQL quanto o necessário para formular uma consulta nesta linguagem. Em (PEREIRA, 1999) foi também desenvolvida uma arquitetura de mediadores, utilizando um produto da IBM (IBM, 2000), o *data joiner*. Esta arquitetura foi aplicada em um caso real de integração de dados de fabricantes de automóveis.

Em um contexto mais específico de uso de mediadores e ontologias, temos os projetos SIMS, InfoSleuth e OBSERVER. O projeto SIMS (*Services and Information Management for decision Systems*) (ARENS, *et al.*, 1993) é formado por um conjunto de agentes de recuperação de informação (mediadores). Cada agente contém um modelo do seu domínio de conhecimento. Este modelo é a base para a interação com um agente específico. Existem também os modelos das fontes de informação que descrevem o conteúdo das fontes de dados e como se dá o mapeamento deste modelo para o modelo do agente. Para expressar os modelos é utilizada uma linguagem de descrição baseada em lógica. O sistema InfoSleuth (BAYARDO, 1999) é baseado no uso de uma infra-estrutura baseada em agentes para a busca de informações via Internet. Os agentes se comunicam uns com os outros através de uma ontologia, usando também ontologias específicas para a captura de informações. Uma característica interessante do InfoSleuth é exatamente o uso de ontologias e de utilizar uma linguagem semanticamente rica (KQML) para a comunicação entre agentes. No entanto, não existe nenhum tipo de integração entre as múltiplas ontologias. Assim, se uma busca é especificada utilizando uma dada ontologia, não existe a possibilidade de realizar esta busca utilizando termos de outras ontologias, como é feito no nosso trabalho, não considerando assim o compartilhamento de vocabulário entre diferentes ontologias. Novamente, nada é especificado em relação à filtragem de

informação. O sistema OBSERVER é baseado em uma estrutura de agentes de recuperação de informações na Internet . Para isso, utiliza uma abordagem baseada no uso de ontologias que descrevem domínios de aplicação. Assim, uma consulta a ser realizada é baseada em termos de uma dada ontologia. No entanto, quando uma consulta não obtém respostas satisfatórias, o sistema pode utilizar uma outra ontologia, ligada a outro conjunto de fontes de informação para realizar a consulta. Desta forma, a chance de se obter respostas satisfatórias é bem maior. No entanto, não é utilizado nenhum tipo de mecanismo que filtre a informação de acordo com as preferências do usuário que está realizando a busca. Além disso, o mecanismo de formulação das consultas, apesar de ser baseado nos termos ontológicos, necessita que o usuário saiba um pouco a respeito de formulação de consultas utilizando operadores lógicos.

3.8 Modelo do Usuário

A modelagem do usuário é uma técnica bastante utilizada tanto na especificação de agentes de interface quanto na especificação de agentes de informação, particularmente na filtragem de informação. Devido a sua importância, descreveremos nesta seção, as principais abordagens utilizadas atualmente para a modelagem do usuário.

A modelagem do usuário pode ser descrita como sendo o processo de criação de um perfil do usuário, baseado em seus interesses e hábitos, e na utilização deste perfil para melhorar a comunicação homem/máquina. Em um sistema de filtragem de informação este modelo do usuário auxilia na disponibilização de apenas as informações que são realmente relevantes ao modelo, antecipando inclusive informações que o usuário sequer supunha que eram necessárias ou relevantes a sua busca. Em um agente de interface, o modelo do usuário auxilia na adaptação da interface de acordo com as preferências do usuário. Podemos definir, ainda, a modelagem do usuário como o processo de adquirir conhecimento sobre o interesse do usuário, ou de um grupo destes, e representar este conhecimento de forma que ele possa ser utilizado posteriormente (POHL, 1997) (FRAGOUDIS, 1997) (LINO, 1999).

Um sistema de modelagem do usuário pode ser dividido em quatro módulos principais: aquisição da informação do usuário, representação da informação, inferência ou raciocínio baseado na informação disponível e decisão. Um modelo do usuário pode ser especificado de diversas maneiras (DAVIES, 1998), dentre elas :

- Através de entrevistas diretamente com o usuário;
- Por engenheiros do conhecimento, usando estereótipos dos usuários (coleções de interesses que são compartilhados por usuários que pertencem a um grupo específico);
- Através de técnicas de Aprendizado de máquina, como inferência, indução, etc., onde o modelador tenta identificar padrões no comportamento do usuário;
- Utilizando perfis construídos através de exemplos, onde o usuário provê exemplos de seu comportamento e o software de modelagem armazena estas informações;
- Utilizando perfis baseados em regras, onde o usuário especifica suas próprias regras no perfil. Estas regras controlam o comportamento do modelo de acordo com a ocorrência de certos eventos.

Segundo POHL (1997), em relação às técnicas empregadas na modelagem, podemos dividir os sistemas em duas abordagens, as que utilizam técnicas de representação do conhecimento e as que utilizam técnicas de aprendizado de máquina. As técnicas de representação do conhecimento utilizam uma maneira formal de representar o conhecimento de modo a manter informações a respeito do usuário em uma base de conhecimento. Técnicas de inferência são utilizadas para estender o modelo e acessar os dados armazenados na base. Uma crítica ressaltada pelos pesquisadores é que o processo de aquisição não é incremental, i.e., não leva em conta as mudanças de atitude do usuário ao longo do tempo. Pohl ressalta que esta deficiência pode levar a previsões não totalmente precisas. Para resolver este problema, um sistema de manutenção da base de conhecimento deve ser especificado.

Abordagens que utilizam técnicas de aprendizado de máquina surgiram mais recentemente. Este tipo de sistema é baseado em históricos de utilização feitos por usuários para tomar decisões. Assim, geralmente em um sistema deste tipo, temos um conjunto de dados de treinamento (*training set*) que servem de suporte para a tomada de decisão. Estes dados de treinamento vão evoluindo ao longo do tempo e são chamados de "resultados do aprendizado". Assim, ao invés de se ter uma base de conhecimento, os resultados do aprendizado servem de fonte de informação a respeito do usuário. O comportamento do usuário é observado e capturado para a formação dos dados do treinamento. A aquisição é feita pelo próprio módulo ,através da aplicação de algoritmos de aprendizado sobre os

dados de treinamento. A representação é implícita e os dados armazenados são específicos para serem utilizados pelos algoritmos de aprendizado (árvores de decisão, redes neurais, probabilidades, etc.). Um problema com abordagens puras de aprendizado de máquina é que se o sistema não possui dados de treinamento, não é possível a tomada de decisão.

De acordo com as características acima, POHL (1997) classifica como ideal o sistema que possa observar o comportamento do usuário, não forçando a aplicação a formar fatos a respeito do usuário baseada em seus dados (modelo mental). A aquisição deve ser incremental, levando em consideração históricos de utilização. A representação do conhecimento deve ser explícita de forma a ser reutilizada, sendo que os fatos a respeito do usuário não podem somente ser comportamentais (técnicas de aprendizado de máquina) ou mentais (base de regras). Uma abordagem que mescle as duas técnicas é mais adequada. Além disso, o sistema deve tomar decisões por si próprio. Podemos incluir ainda nesta caracterização o sistema utilizar uma abordagem que leve em consideração o comportamento de um grupo de usuários.

3.9 Conclusões

Este capítulo apresentou aspectos conceituais relacionados a mecanismos de busca e recuperação de componentes. A tecnologia de agentes é a base conceitual destas tecnologias. Devido à relativa novidade da área de agentes, principalmente no contexto da engenharia de software, consideramos importante uma descrição geral da tecnologia.

Analisando os conceitos envolvidos com sistemas baseados em agentes (SBAs), podemos observar que existem diversas técnicas a serem utilizadas para a sua especificação. As técnicas de aprendizado de máquina parecem ser as mais promissoras, e são as mais utilizadas nos sistemas existentes. No entanto, estes sistemas existentes raramente utilizam somente aprendizado de máquina. Na verdade, as abordagens utilizadas são híbridas, onde abordagens baseadas em base de regras se mesclam com técnicas de aprendizado de máquina. Além disso, a utilização de técnicas para modelagem do usuário é também uma constante neste tipo de sistema. Assim, supomos adequada uma abordagem que siga estes caminhos, ou seja, utilize técnicas de modelagem do usuário, conjuntamente com uma base de regras e técnicas mais ativas para captura do comportamento do usuário, como são as técnicas de aprendizado de máquina. No capítulo 5, apresentamos uma abordagem que segue esta perspectiva.

4 Odyssey-DE

4.1 Introdução

De maneira geral, um processo de desenvolvimento, para ser eficaz e conduzir à construção de produtos de boa qualidade, deve ser definido considerando-se seu propósito, a tecnologia a ser adotada na construção do produto, a organização que irá utilizar o processo, o grupo de desenvolvimento e os futuros usuários do produto (ROCHA *et al.*, 1996).

Desta forma, um processo de engenharia de domínio (ED) deve levar em conta estas características, devendo ser definido respeitando-se:

- o propósito do projeto (e.g. geração de aplicações no domínio e/ou maior entendimento dos conceitos relacionados ao domínio e/ou apoiar o desenvolvimento baseado em componentes (DBC));
- a tecnologia adotada na construção do produto (i.e., uso do paradigma OO, técnicas de IA, técnicas para sistemas distribuídos, etc.);
- a organização (i.e., quais são os objetivos específicos da empresa com a implantação de um processo de ED);
- o grupo de desenvolvimento (i.e., que tipo de técnicas de desenvolvimento de software são mais familiares ao grupo, quais são os profissionais disponíveis) e os futuros usuários do produto (que tipo de produtos a ED deve disponibilizar para satisfazer o usuário final: uma taxonomia do domínio, uma biblioteca de componentes reutilizáveis, diretivas para a construção de aplicações no domínio, etc.).

Na literatura especializada, diversas abordagens para a sistematização do processo de ED foram propostas, conforme apresentamos no capítulo 2. Por outro lado a necessidade de desenvolvimento de aplicações cada vez mais complexas e distribuídas está fazendo com que abordagens para o desenvolvimento de aplicações a partir de "partes" interoperáveis esteja ganhando força. Neste contexto, existe uma demanda crescente por processos que sistematizem a criação destas "partes" ou componentes interoperáveis, de forma adequada. Para que estes componentes sejam criados para serem reutilizados no desenvolvimento de aplicações, estes devem ser especificados de acordo com os direcionamentos dos processos de ED, cuja ênfase em reutilização é característica básica.

Analisando estas abordagens para a ED e as necessidades do desenvolvimento de aplicações atuais, concluímos que nenhuma das abordagens é totalmente adequada às necessidades de desenvolvimento de componentes, para suprir a demanda de desenvolvimento de novas aplicações. O desenvolvimento de aplicações, atualmente, requer o suporte de componentes reutilizáveis em todas as etapas do desenvolvimento, desde as fases de análise até a fase de implementação. Além disso, novos modelos arquiteturais, distribuição e necessidade facilitada de busca por componentes reutilizáveis existentes, devem ser levados em consideração. Com isso, consideramos necessária a especificação de um processo de ED que tentasse minimizar as deficiências encontradas nos processos descritos na literatura e absorvesse suas vantagens.

Apresentamos neste capítulo um processo de ED com ênfase em DBC, denominado Odyssey-DE. O Odyssey-DE enfoca, de maneira sistemática, o desenvolvimento de componentes com ênfase na sua reutilização. Assim, a abordagem do Odyssey-DE une os aspectos de reutilização e entendimento do domínio, que são providos pelos processos de ED atualmente disponíveis, e o detalhamento do desenvolvimento de componentes, provido pelos processos de DBC, atendendo aos requisitos listados no capítulo 2.

Outro ponto importante é que, devido à complexidade das aplicações atuais e consequentemente dos processos de desenvolvimento para apoiá-las, o suporte de ferramental automatizado é indispensável. Neste sentido, o Odyssey-DE conta com o suporte da infra-estrutura Odyssey, que apoia, além do desenvolvimento de componentes reutilizáveis e do desenvolvimento de aplicações baseadas nestes componentes, o armazenamento, busca e recuperação destes componentes reutilizáveis. Os aspectos da infra-estrutura Odyssey relacionados ao suporte ao Odyssey-DE serão descritos neste capítulo e no capítulo 5 serão detalhados os aspectos de armazenamento, busca e recuperação de componentes.

A descrição da utilização do Odyssey-DE em um domínio específico é apresentada no capítulo 6, sendo este domínio o domínio de processamento legislativo[1].

Ao longo do presente capítulo, apresentamos alguns *templates* de modelos utilizados no processo. Estes *templates* serão apresentados utilizando-se a infra-estrutura Odyssey (WERNER *et al.*, 2000) como ferramental.

---

[1] O domínio de Processamento Legislativo engloba todas as etapas relacionadas à tramitação de projetos no contexto de casas legislativas, sejam estas municipais, estaduais ou federais.

4.2 Desenvolvimento baseado em Reutilização

Um processo de desenvolvimento baseado em reutilização é caracterizado por duas etapas distintas: o desenvolvimento de artefatos reutilizáveis, denominado genericamente de processo de **desenvolvimento para reutilização**, e o processo de desenvolvimento de aplicações baseadas nestes artefatos, denominado **desenvolvimento com reutilização**. Apresentamos a seguir o **Odyssey-DE**, que segue uma abordagem de **desenvolvimento para reutilização** e o **Odyssey-AE**, que trata do **desenvolvimento com reutilização**.

4.2.1 Odyssey-DE – Processo de Engenharia de Domínio do Odyssey

Segundo PRESSMAN (1997), todos os projetos de desenvolvimento de software possuem atividades gerenciais e específicas de desenvolvimento. As atividades gerenciais, tais como planejamento de entregas dos produtos, alocação de recursos, monitoramento do processo e análise de riscos, entre outros, e as atividades de desenvolvimento, que criam o software a partir de seus requisitos. O Odyssey-DE é composto de cinco etapas: **planejamento, análise de risco, análise do domínio, projeto do domínio e implementação do domínio**. Trata-se, portanto, de um processo completo, onde tanto atividades gerenciais quanto de desenvolvimento são contempladas. No entanto, no contexto desta tese, somente as atividades de desenvolvimento são descritas em detalhes. Detalhes específicos a respeito de atividades gerenciais fazem parte de outro trabalho de pesquisa (BARROS, 2000).

Na etapa de planejamento, há uma subfase, denominada "estudo da viabilidade do domínio", que é uma atividade essencialmente relacionada com a decisão sobre a realização efetiva da ED naquele domínio. Devido a sua importância para as atividades de desenvolvimento, esta é especialmente descrita em detalhe nesta tese.

A etapa de **planejamento do domínio** tem como objetivo prover um conjunto de critérios que permitam ao gerente do domínio fazer estimativas razoáveis em termos de recursos a serem utilizados, custos e cronograma. Estas estimativas são realizadas no início do processo e atualizadas regularmente. Dentre as atividades que fazem parte do planejamento do domínio, temos: estudo da viabilidade do domínio, estimativa dos recursos requeridos para a realização da engenharia do domínio, incluindo recursos humanos (especialistas, engenheiros do domínio, desenvolvedores de aplicações no domínio, programadores, etc.), recursos de software (fontes de informação do domínio, componentes reutilizáveis já existentes no domínio, etc), estimativa dos custos do

projeto (BARROS, 2000). No estudo de viabilidade do domínio, são definidos alguns critérios e pontuação para estes critérios, sendo discutida a real necessidade de se desenvolver componentes reutilizáveis no domínio.

A **análise de risco** trata as ameaças ao planejamento realizado na fase anterior, ou seja, como tentar minimizar problemas como atrasos no cronograma, aumento de custos, entre outros. Para isso, deve-se analisar prioritariamente dois tipos de riscos: risco técnico, que está relacionado ao software a ser desenvolvido e risco do negócio, que lida com a viabilidade dos componentes reutilizáveis a serem desenvolvidos. Dentre as atividades realizadas nesta etapa, temos: identificação dos riscos (identificação de potenciais ameaças ao plano do projeto), projeção (estimativa da probabilidade de ocorrência do risco e as conseqüências associadas ao risco) dos riscos, monitoração e gerenciamento dos riscos (uma vez identificados e estimados como monitorar e gerenciar os riscos) (BARROS, 2000).

A **análise do domínio** consiste na definição dos principais conceitos do domínio, ressaltando as similaridades e diferenças entre estes conceitos em um nível de abstração que seja de fácil entendimento para os diversos usuários do domínio. Para facilitar a seleção de atividades e conceitos do domínio relevantes a uma dada aplicação, os produtos desta fase do Odyssey-DE são particionados de acordo com as principais características (serviços) do domínio. Desta forma, todos os modelos utilizados são baseados nesta divisão. É também nesta etapa que é criado o modelo de características do domínio, que é um modelo que captura as principais abstrações do domínio. Nesta etapa, são detalhadas ainda as interfaces externas dos componentes em termos funcionais.

Na etapa de **projeto do domínio**, os conceitos e características considerados importantes no domínio são projetados, adicionando-se considerações arquiteturais e de projeto aos componentes conceituais identificados na etapa anterior. São também detalhadas as interfaces externas dos componentes em termos arquiteturais. Novamente, os modelos são divididos baseados nas principais características do domínio.

Finalmente, na etapa de **implementação do domínio**, os componentes especificados na fase de projeto são codificados, definindo-se precisamente suas interfaces para comunicação com outros componentes. Nesta etapa é analisado se existem componentes legados que disponibilizam alguma funcionalidade requerida pelo domínio. Neste caso, o componente legado é agregado ao domínio através da definição

de uma interface que permita que este componente seja conectado aos demais componentes do domínio de forma transparente.

A disponibilização de componentes reutilizáveis no contexto de um dado domínio deve ser feita em um processo incremental e evolutivo, ou seja, a medida que novas informações vão sendo agregadas ao domínio, os componentes reutilizáveis vão sendo evoluídos com base nas novas informações do domínio adquiridas. Para isso, o Odyssey-DE necessita de um modelo de ciclo de vida que atenda a esta necessidade.

Um modelo de ciclo de vida descreve as etapas do processo de desenvolvimento e as atividades a serem realizadas em cada etapa (PRESSMAN, 1997). Analisando os modelos de ciclo de vida descritos na literatura, concluímos que a proposta do modelo em espiral é a mais adequada para os propósitos do Odyssey-DE. Além disso, todas as etapas do processo são realizadas de maneira interativa e incremental, uma vez que os componentes reutilizáveis disponibilizados em todas as etapas do processo são melhorados durante todo o ciclo de vida. O desenvolvimento dos componentes reutilizáveis passa desde a fase de análise até a fase de testes, completando um ciclo inteiro do desenvolvimento de software. Um diagrama simplificado com o modelo de ciclo de vida do Odyssey-DE em espiral, é apresentado na Figura 4.1. Este modelo pode ser visto de acordo com a classificação de PRESSMAN (1997) como *spiral model adapted for the entire life cycle*.

Devemos ressaltar que, a partir de um esforço inicial de identificação dos principais conceitos do domínio, o processo de desenvolvimento de componentes reutilizáveis, nos mais diversos níveis de abstração, podem ocorrer em paralelo, através de mini-iterações (espirais internas, Figura 4.1). Nesse sentido, o processo do Odyssey-DE se assemelha ao modelo de ciclo de vida concorrente descrito em (PRESSMAN, 1997), ou seja, como o principal objetivo é o desenvolvimento de componentes reutilizáveis, o desenvolvimento de um determinado componente do domínio pode estar na fase de implementação e o desenvolvimento de outro componente pode estar ainda na fase de validação de casos de uso, tudo isto ocorrendo em paralelo. Além disso, quando o esforço inicial de identificação dos principais componentes reutilizáveis no domínio já tiver sido realizado, pode ser que ocorra a necessidade de inserir novas e/ou modificar informações do domínio. Esta possibilidade está presente na Figura 4.1 através da caixa *2 - Aquisição de novas informações*, que insere informações na segunda iteração do ciclo. A partir desta inserção de novas informações, as atividades das etapas de análise, projeto e implementação do ciclo se modificam, sendo realizadas as

atividades identificadas pelo número 2 na Figura 4.1. Assim, como podemos ver na Figura 4.1, a atividade se transforma em uma reavaliação do impacto daquela nova informação nos modelos já criados. Através da aquisição de novas informações relativas ao domínio, na Figura 4.1, esta nova interação inicia-se com a aquisição de novas informações (quadro em destaque).

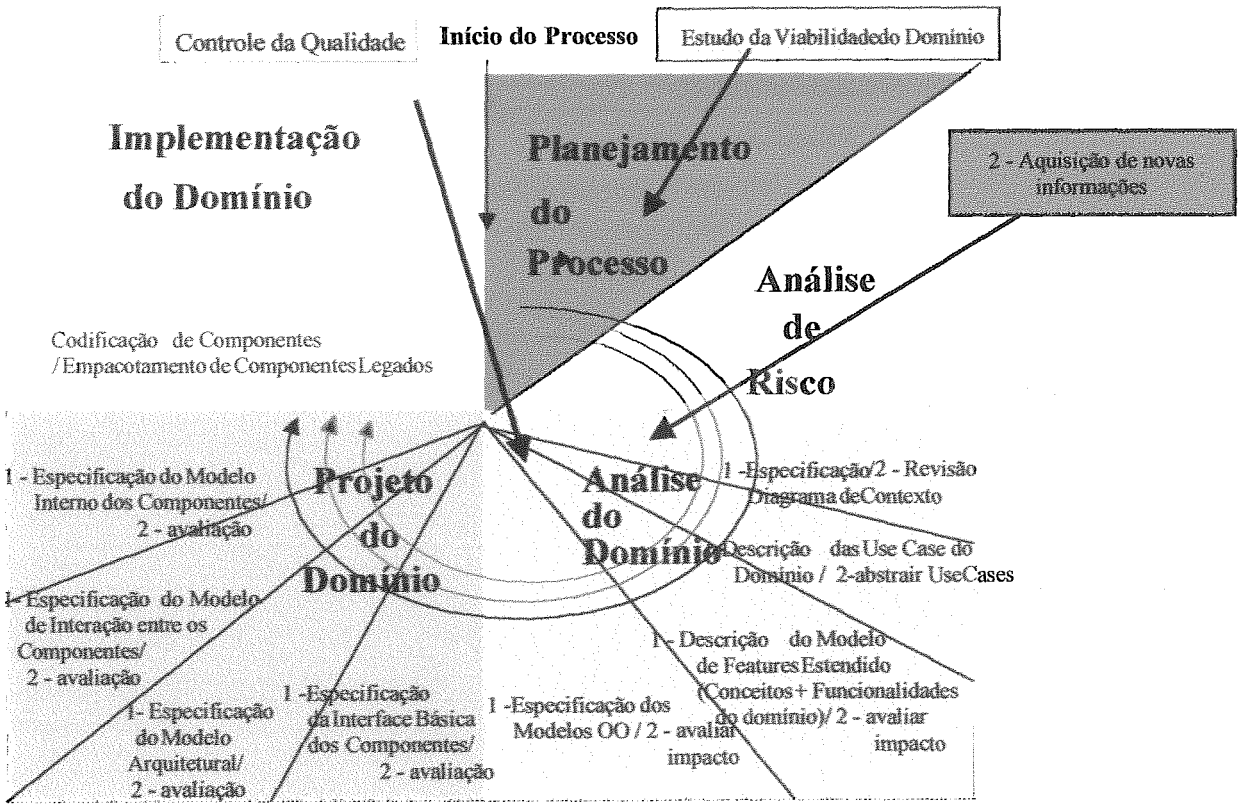

Figura 4.1 – Ciclo de vida do Odyssey-DE

A avaliação da qualidade destes componentes é de suma importância para o processo, uma vez que estes componentes serão utilizados na construção de aplicações no domínio. Assim, é premissa básica para o sucesso da implantação da reutilização, que os componentes reutilizáveis sejam de qualidade. Uma das formas para garantir a qualidade dos componentes, é a constante avaliação dos produtos de cada fase pelos usuários do domínio, ou seja, pelos especialistas e desenvolvedores de aplicações no domínio. Esta atividade de avaliar qualidade é considerada no contexto do Odyssey-DE como atividade de suporte ao processo como um todo, durante todo o ciclo de vida.

4.2.2 Odyssey-AE – Processo de Engenharia de Aplicações do Odyssey

A Engenharia de Aplicações (EA) se dedica ao estudo das melhores técnicas, processos e métodos para a produção de aplicações, no contexto do desenvolvimento

atividades identificadas pelo número 2 na Figura 4.1. Assim, como podemos ver na Figura 4.1, a atividade se transforma em uma reavaliação do impacto daquela nova informação nos modelos já criados. Através da aquisição de novas informações relativas ao domínio, na Figura 4.1, esta nova interação inicia-se com a aquisição de novas informações (quadro em destaque).

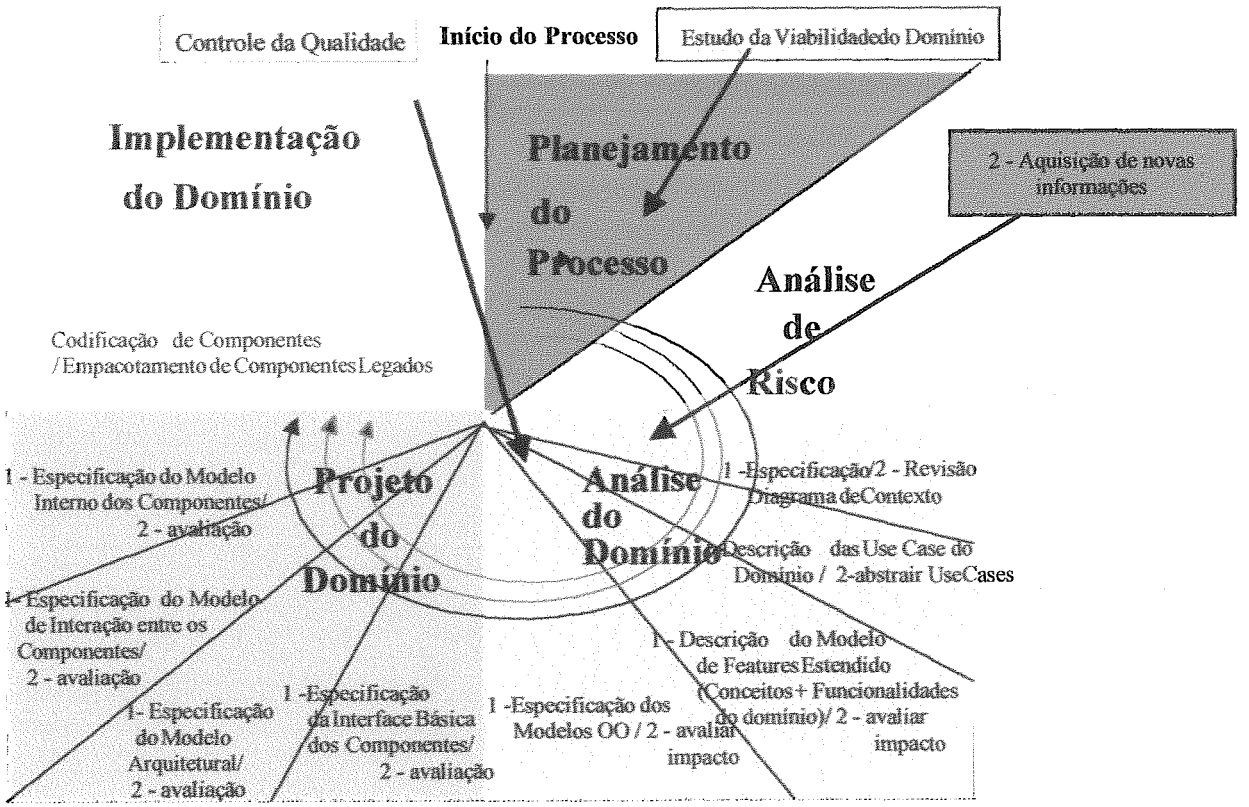

Figura 4.1 – Ciclo de vida do Odyssey-DE

A avaliação da qualidade destes componentes é de suma importância para o processo, uma vez que estes componentes serão utilizados na construção de aplicações no domínio. Assim, é premissa básica para o sucesso da implantação da reutilização, que os componentes reutilizáveis sejam de qualidade. Uma das formas para garantir a qualidade dos componentes, é a constante avaliação dos produtos de cada fase pelos usuários do domínio, ou seja, pelos especialistas e desenvolvedores de aplicações no domínio. Esta atividade de avaliar qualidade é considerada no contexto do Odyssey-DE como atividade de suporte ao processo como um todo, durante todo o ciclo de vida.

4.2.2 Odyssey-AE – Processo de Engenharia de Aplicações do Odyssey

A Engenharia de Aplicações (EA) se dedica ao estudo das melhores técnicas, processos e métodos para a produção de aplicações, no contexto do desenvolvimento

baseado em reutilização (GRISS *et ali.*, 1998). Assim sendo, a EA atua de forma paralela à ED. Na EA, os componentes construídos na ED são utilizados para o desenvolvimento de um produto de software.

O processo genérico de EA no contexto da infra-estrutura Odyssey, é denominado Odyssey-AE e tem como um de seus principais objetivos a completa integração com o Odyssey-DE. As atividades do Odyssey-AE devem servir como referência na definição de processos de desenvolvimento de aplicações no contexto do Odyssey.

O processo genérico de EA do Odyssey, está baseado nas seguintes fases (MILLER, 2000): Planejamento e Análise de Risco, Análise (que determina os requisitos da aplicação), Projeto (soluções arquiteturais) e Implementação (componentes implementados). Estas fases são dispostas num ciclo de vida em espiral (Figura 4.2).

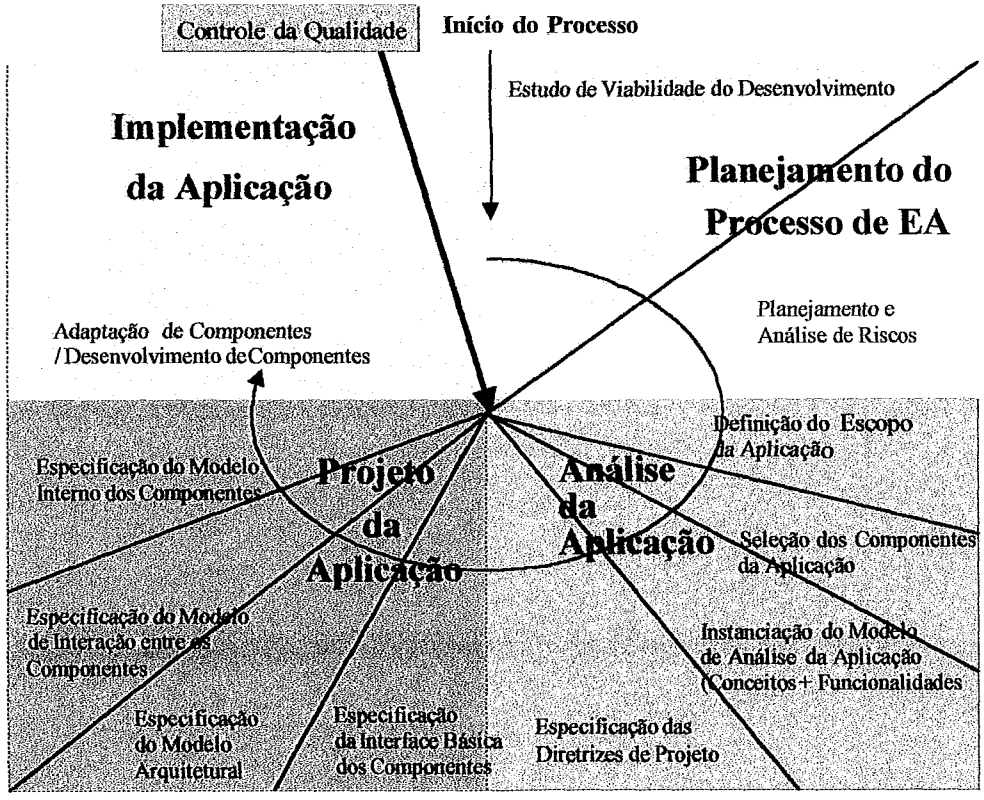

Figura 4.2 – Ciclo de vida do Odyssey-AE

O Odyssey-AE começa com uma fase de planejamento e análise de risco, que consiste simplificadamente na avaliação do domínio com o intuito de verificar a viabilidade do desenvolvimento da aplicação desejada, utilizando os componentes existentes. Esta avaliação deve levar em conta as funcionalidades já disponibilizadas no

domínio, a afinidade da família de sistemas com o produto que será construído e os riscos para o processo de desenvolvimento identificados na ED.

Após a fase de planejamento, o processo é composto das tradicionais fases de análise, projeto e implementação. Durante a EA, escolhe-se os componentes necessários à aplicação num alto nível de abstração. Estes componentes se encontram conectados semanticamente aos de outros modelos, abrangendo todas as fases do desenvolvimento, garantindo assim sua ligação com os componentes implementados ou semi-desenvolvidos do domínio.

No Odyssey-AE, a visão funcional é o primeiro nível de contato do reutilizador com o modelo do domínio, e é baseado nesta informação que o Engenheiro da Aplicação inicia sua exploração sobre os demais aspectos modelados, visando a produção da sua aplicação. Deste modo, no processo de EA, o desenvolvedor escolhe as funcionalidades que ele julga necessárias a sua aplicação. As fases posteriores do desenvolvimento são regidas por estas escolhas, na medida em que elas restringem o escopo de informações do domínio que farão parte da aplicação.

Na fase de Projeto, os conceitos provenientes da fase de análise são detalhados, com base em componentes arquiteturais do domínio e nas informações fornecidas pelo reutilizador. É nesta fase que se define a arquitetura da aplicação através da instanciação de estilos arquiteturais do domínio, disponibilizadas previamente através do processo de ED (XAVIER, 2000). O processo de adaptação dos componentes da aplicação à arquitetura instanciada e a adição de outros componentes tecnológicos (como os que tratam do relacionamento com bancos de dados, por exemplo), devem ser executados pelo Engenheiro da Aplicação neste momento.

Finalmente, na fase de implementação, o desenvolvedor se preocupa com características ligadas à codificação da aplicação. É nesta fase que se tratam características específicas ligadas ao sistema operacional, interface com o usuário, armazenamento de dados, etc. O intuito é estabelecer, de fato, o código executável da aplicação.

4.3 Conceitos básicos

Alguns conceitos inerentes aos processos Odyssey-DE e Odyssey-AE, em particular, e ao projeto Odyssey, de maneira geral, merecem destaque. Dentre estes conceitos, podemos destacar o conceito de modelo de características estendido, o

conceito de rastreabilidade e a preocupação com a especificação de componentes reutilizáveis desde as fases iniciais do processo.

O projeto Odyssey utiliza um modelo em alto nível de abstração para capturar os principais conceitos e funcionalidades do domínio. Este modelo, chamado de modelo de características, foi especificado inicialmente no contexto do método FODA (KANG *et al.*, 1990), e desde então é considerado um modelo adequado para capturar as similaridades e diferenças entre os conceitos do domínio, além de permitir a modelagem dos serviços no contexto de um dado domínio. O modelo é apresentado de forma hierárquica, com o objetivo de facilitar o reutilizador na identificação das características do domínio que são passíveis de reutilização. Também podem ser representados no modelo as características que são essenciais ou opcionais dentro de um domínio. Atualmente, o modelo de características é um modelo conhecido no contexto da ED e utilizado em diversos processos FeatuRSEB (GRISS *et al.*, 1998), FORM (KANG *et al.*, 1998), ODM (SIMOS, 1996).

No projeto Odyssey, esta definição clássica de modelo de características foi estendida, adicionando características como uma taxonomia mais abrangente, capacidade de extensão do modelo, através da criação de novas visões e submodelos, e uso de recursos visuais. Deste modo, é facilitada a finalidade de transformar o modelo de características numa representação padronizada dos diversos componentes do domínio, e sua integração (através da rastreabilidade) (MILLER, 2000). O modelo de Características do Odyssey-DE apresenta uma notação que é uma evolução da notação utilizada nos modelos de características descritos no FODA (KANG *et al.*, 1990) e FODACom (GRISS *et al.*, 1998). Esta notação foi proposta por (MILLER, 2000) e tem como principal objetivo obter uma melhor apresentação visual dos conceitos embutidos no modelo de características. Além disso, nestes trabalhos, o modelo de características não possui o conceito de múltiplas visões e múltiplos níveis, como é proposto em (MILLER, 2000).

No contexto desta tese somente são consideradas características funcionais e conceituais. No entanto, aspectos não funcionais também são contemplados no modelo de características estendido proposto em (MILLER, 2000), sendo que as características não funcionais são detalhadas em (XAVIER, 2000). As características não funcionais foram definidas buscando o conjunto mínimo de requisitos de qualidade estabelecido na norma ISO/IEC 9126, tendo em vista um conjunto de padrões arquiteturais específico (BUSCHMANN *et al.*, 1996) para um domínio/aplicação. Algumas destas

características são: Interoperabilidade, Segurança de acesso, Maturidade, Tolerância a falhas, Modificabilidade, Testabilidade e Portabilidade, sendo que as mesmas podem ainda ser agregadas em grupos.

Uma característica interessante do modelo de características estendido é a descrição mais abrangente de cada característica, através de um *template* estruturado[2] para descrição da mesma. Este *template* estruturado é denominado *padrão de domínio*, sendo similar aos padrões de documentação para o *framework* HotDraw (JOHNSON, 1992) e aos padrões para documentação de componentes reutilizáveis tal como proposto em (SILVA e WERNER, 1996). A importância do padrão de domínio reside na sua própria estruturação, apresentando o detalhamento da característica, bem como suas ligações com os componentes de outros modelos, facilitando sua compreensão pelo Engenheiro da Aplicação e evitando descrições discrepantes ou incompletas sobre componentes, feitas por diferentes especialistas do domínio.

Inicialmente, foi definido um conjunto de diferentes tipos de características, organizados em visões relativas às fases do desenvolvimento (MILLER, 2000). Estes tipos podem ser estendidos pelo reutilizador, se necessário, criando-se novas características adequadas ao domínio em questão ou a sua realidade específica. Os seguintes tipos de características foram especificadas: funcionais/conceituais (representando os principais conceitos e funcionalidades do domínio); de projeto e implementacionais.

As características funcionais/conceituais são subdivididas ainda em: essenciais, organizacionais, entidade, externas, não definidas e adicionais. Maiores detalhes sobre estes tipos está fora do escopo desta tese. O leitor interessando em um maior detalhamento em relação a este aspecto, deve consultar (MILLER, 2000).

Os relacionamentos entre as características também são importantes neste modelo. Estes relacionamentos não são apenas hierárquicos. Podemos visualizar o modelo de características estendido como um grafo que pode se expandir em várias direções e não somente na forma de uma árvore. Os relacionamentos entre as características também possuem uma nomenclatura e um significado semântico próprio. É interessante notar que, nos outros métodos que utilizam o conceito de características, os relacionamentos são considerados características auxiliares, não tendo no modelo relevância similar a das próprias características. O enfoque do Odyssey valoriza a

[2] Este *template* será descrito em detalhes na seção 4.4.

semântica deste tipo de característica do modelo, dando a ela uma maior capacidade de representação, através das características adicionais propostas (novos tipos de características e relacionamentos).

Assim, a ênfase do modelo de características estendido é exatamente ser uma taxonomia detalhada do domínio, com o objetivo de se ter um maior entendimento do domínio como um todo, facilitando assim o desenvolvimento com reutilização de aplicações no domínio. Para isso, o modelo de características estendido possui duas visões, que são essenciais para o completo entendimento do domínio por parte dos usuários, são elas: **Modelo de Funcionalidades do Domínio** que apresenta, em um nível abstrato, o relacionamento entre as principais funcionalidades do domínio; e **Modelo de Conceitos do Domínio**, que representa os conceitos do domínio e o relacionamento entre eles. Neste sentido, o modelo de características se assemelha ao conceito de modelo ontológico, proposto por GUARINO (1998).

A divisão do modelo de características estendido, em visão conceitual e funcional, permite ao engenheiro da aplicação um maior entendimento dos conceitos do domínio e a seleção de quais funcionalidades/componentes são realmente necessárias no desenvolvimento de uma dada aplicação. Além disso, no Odyssey, estes conceitos do domínio são representados pelo modelo de conceitos do domínio, no nível conceitual, no modelo arquitetural dos componentes, a nível arquitetural e a nível implementacional pelos componentes binários, disponíveis para serem utilizados no domínio. No modelo de características original do FODA e usado pelo FODACom, todas estes características são representadas em um só modelo de características, o que torna o entendimento dos conceitos difícil.

Esta divisão facilita o entendimento, uma vez que se o objetivo for o de entender alguma característica essencialmente de projeto, o desenvolvedor pode consultar somente o modelo relacionado a este nível. Como o propósito do Odyssey é prioritariamente facilitar o entendimento do usuário, devemos evitar que características não necessárias para o usuário em um dado momento sejam apresentadas. Assim, considerações arquiteturais e implementacionais não devem ser feitas no nível conceitual e por isso são deixadas para quando forem realmente necessárias, utilizando assim a idéia de múltiplos níveis.

Analisando as semelhanças e diferenças entre o modelo de características e o modelo ontológico como descrito por GUARINO (1998), podemos considerar que o modelo de características seria um modelo que implementa em "parte" os conceitos e

restrições utilizadas nos modelos ontológicos. O modelo de características é uma descrição dos termos e o relacionamento entre eles, formando uma rede semântica. Esta conceituação de modelo de características é exatamente a conceituação clássica utilizada para a descrição de uma ontologia. No entanto, não possui o formalismo de um modelo ontológico. Não são utilizados axiomas formais para se descrever os termos e relacionamentos do domínio. Apesar do modelo de características estendido do Odyssey-DE modelar restrições entre características, não é utilizado nenhum formalismo nesta modelagem, o que dificulta sua verificação. No entanto, em relação ao propósito final, que é o entendimento do domínio e seus termos, ambas as abordagens são similares.

Analisando o modelo de características sob uma ótica de recuperação de informação, podemos dizer que este modelo é bastante semelhante ao conceito de ontologia utilizada pela comunidade de banco de dados (WIEDERHOLD, 1994) (BAYARDO, 1997), no sentido de ser um modelo que descreve um determinado domínio em um nível conceitual alto e onde o mapeamento entre as informações a serem recuperadas e os termos do modelo ontológico são especificados. Desta forma, o modelo de características atende a este propósito, tendo como objetivo a modelagem de um domínio em alto nível de abstração, agregando aos termos informações que permitam o seu entendimento (padrão de domínio) e o mapeamento destas para outras informações do domínio (rastreabilidade entre modelos).

Outro conceito importante é o de "rastreabilidade" entre os modelos. A "rastreabilidade" é definida como o relacionamento entre os diversos modelos descritos no Odyssey, de forma a facilitar a identificação de componentes do domínio em todos os níveis de abstração. Assim, as ligações de "rastro" são especificadas de forma a se ter diversas visões de um componente, desde a funcionalidade abstrata que o representa, descrita no Modelo de Funcionalidades do Domínio, passando pelos conceitos relacionados, descritos no Modelo de Conceitos do Domínio, até um detalhamento maior do componente, através de modelos de classes, diagrama de seqüência, etc.

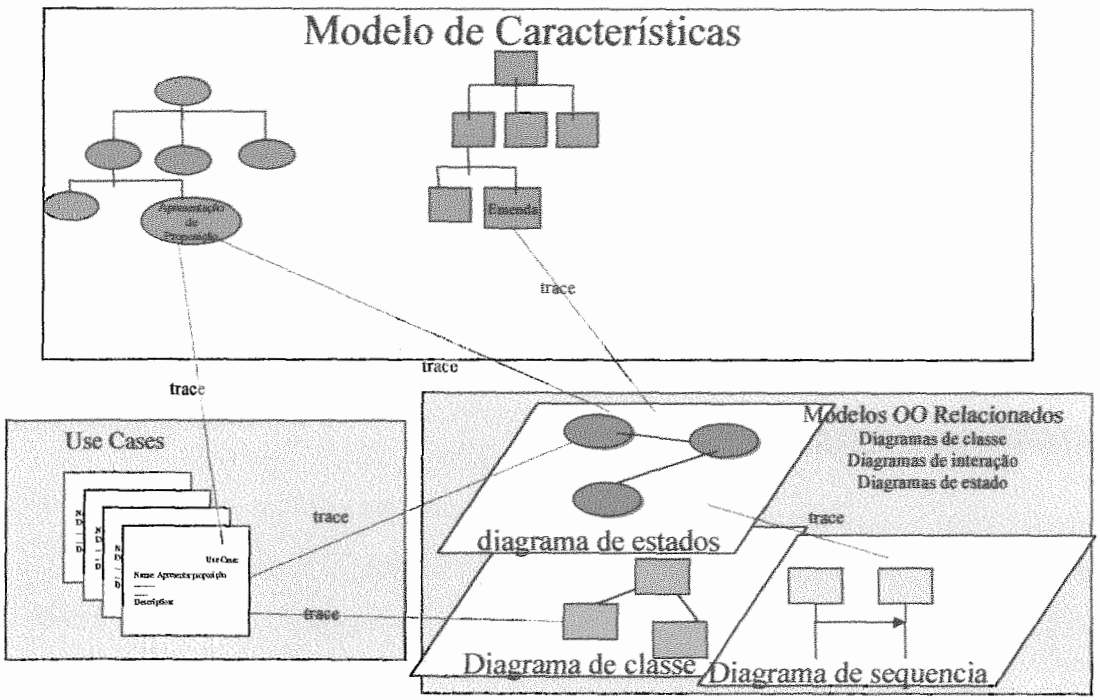

Figura 4.3 – "Rastreabilidade" entre os modelos

Com este conceito de rastro, o entendimento do componente é facilitado, uma vez que se consegue navegar pelos diversos modelos (Figura 4.3), que detalham visões diferentes sobre o componente, possibilitando o entendimento do domínio como um todo e facilitando, por conseqüência, a reutilização em níveis mais baixos de abstração. Com isso, é inerente ao processo, a preocupação com a especificação de componentes desde as fases iniciais. Um componente é representado no modelo de características em seu mais alto nível de abstração e detalhado dos diversos modelos relacionados. Portanto, se o usuário estiver examinando um determinado modelo, este modelo possuirá ligações com os outros modelos que descrevem o mesmo componente, em um mesmo nível de abstração ou entre diferentes níveis.

4.4 Etapas do processo Odyssey-DE

A seguir descrevemos em detalhe, algumas das fases do processo Odyssey-DE, definidas no contexto desta tese.

4.4.1 Estudo de Viabilidade do Domínio

A identificação e seleção de um domínio específico deve ser a primeira etapa do processo de ED, sendo crítico para o sucesso de todo o processo (SIMOS, 1996), (QSORT, 1995). O objetivo principal da seleção de um domínio adequado é a minimização do risco de se realizar a ED, através da identificação de um domínio que

tem grande possibilidade de ser analisado com sucesso, dado o tempo e recursos disponíveis.

A seleção de um domínio adequado depende prioritariamente da efetividade da equipe que irá participar do processo de ED em definir, usar e aplicar critérios para seleção e avaliação do domínio. Sendo assim, a definição de alguns critérios e a pontuação destes critérios é uma forma de se realizar esta seleção. No entanto, como atestado em (ARC, 1996), o verdadeiro sucesso da ED só pode ser medido posteriormente, em termos da utilidade dos produtos da ED para seus usuários. Mesmo assim, a adoção de alguns critérios de seleção pode ajudar na seleção de um domínio que tenha maior chance de sucesso.

Baseados nos trabalhos descritos em (ARC, 1996) e (SIMOS, 1996) em relação a estudos de viabilidade de realização da ED em domínios específicos, descrevemos a etapa que denominamos "estudo de viabilidade do domínio", cujo principal objetivo é a identificação de um domínio, entre os domínios candidatos, que seja o mais adequado para a reutilização.

Esta etapa é composta das seguintes fases:

- Seleção dos domínios candidatos ao processo de reutilização;
- Seleção de critérios definidos para a avaliação dos domínios candidatos;
- Pontuação e totalização dos critérios utilizados na seleção e seleção do domínio mais adequado à reutilização;
- Identificação inicial dos relacionamentos entre os domínios candidatos e o domínio selecionado.

A fase de seleção dos domínios candidatos ao processo de reutilização é realizada através de um *brainstorm* entre especialistas das diversas áreas de atuação estratégicas para a empresa, engenheiros de domínio e desenvolvedores de aplicações estrategicamente selecionados. Nesta reunião são selecionados domínios os quais os participantes consideram adequados para sofrerem um processo de ED para a disponibilização de componentes reutilizáveis. O resultado desta fase é a seleção de 2 a 4 domínios candidatos ao processo de reutilização[3].

[3] Se existir somente um domínio, este domínio será avaliado pelos critérios de seleção e somente se este receber uma pontuação considerada aceitável é que o mesmo será passível de sofrer um processo de Engenharia de Domínio.

Na fase de seleção de critérios para a avaliação de domínios são selecionados critérios gerais de reutilização e critérios específicos para a empresa. Com base nos critérios identificados em (ARC, 1996), identificamos três áreas principais de critérios: técnicos, organizacionais e sociais. Basicamente adicionamos os critérios técnicos 3 e 7, o critério organizacional 3 e o critério social 4, apresentados nas tabelas 4.1, 4.2 e 4.3. Além destes critérios apresentados pelo Odyssey-DE, podem ser agregados novos critérios de acordo com a organização que está utilizando o processo. Sendo assim, de acordo com os domínios levantados e com os participantes do *brainstorming* inicial, novos critérios podem ser definidos. O processo para a definição de novos critérios é feito da seguinte maneira: os critérios básicos definidos pelo Odyssey-DE são lidos e discutidos para se ter um entendimento bem detalhado dos mesmos. A seguir, os participantes, em um prazo a ser estipulado, podem propor novos critérios, justificando claramente a sua proposição. As propostas de novos critérios são analisadas até se chegar a um consenso de quais devem ser agregados aos critérios básicos.

Uma vez definidos quais são os critérios a serem analisados, inicia-se a fase de pontuação e totalização dos critérios de seleção. Nesta fase são definidos pesos para cada um dos critérios. Esta totalização de critérios pode ser feita seguindo diversas abordagens, variando de formais, baseadas em modelos matemáticos, a informais. Em (ARC, 1996), apresenta-se como uma possível proposta o seguinte procedimento: cada participante na seleção dos domínios define individualmente pesos para cada critério baseando-se na importância do critério, onde 1 é a mais baixa pontuação e 10 a mais alta. É realizada então uma reunião para se formular um peso mediano para cada critério. Pesos definidos para critérios utilizados em processos de seleção anteriores podem ser utilizados para auxiliar nesta definição. Se um critério tiver uma grande variação de pesos, este critério deve ser discutido em detalhe. Cada participante deve justificar o peso escolhido. No capítulo 6 desta tese, apresentamos o estudo de viabilidade realizado para o domínio legislativo.

Técnicos

Critério	*Descrição*
1. Número de sistemas existentes	Determina se existem sistemas suficientes no domínio para serem analisados para a criação dos modelos do domínio. Além disso, este critério mede também a maturidade do domínio
2. Futuros sistemas a serem desenvolvidos	Determina se existem necessidades futuras de desenvolvimento de sistemas no domínio

3. Melhoramentos nos sistemas existentes	Determina se existem sistemas no domínio que devem ser melhorados
4. Recursos técnicos disponíveis para a ED	Determina se existem recursos técnicos disponíveis para o projeto. Ex. linguagem de programação OO, protocolo CORBA, componentes legados, etc.
5. Estabilidade do domínio	Se a variação de um sistema para outro no domínio for grande, pode não ser possível produzir componentes reutilizáveis ou mantê-los corretos. Este critério mede até que ponto os sistemas no domínio são parecidos.
6. Acessibilidade das informações	Este critério mede a disponibilidade de informações técnicas sobre o domínio, ou seja, se os sistemas já desenvolvidos no domínio foram bem documentados.
7. Metodologias utilizadas no desenvolvimento de sistemas no domínio	O uso de uma metodologia consistente faz com que seja fácil comparar os sistemas existentes e documentação para se identificar as similaridades e diferenças entre os sistemas existentes.

Tabela 4.1 – Critérios Técnicos para escolha do Domínio

Organizacionais

Critério	*Descrição*
1. Potencial para reutilização	Este critério mede o quão útil os produtos da ED podem ser se novos sistemas no domínio forem desenvolvidos ou se sistemas já existentes forem melhorados. Como o principal objetivo da ED é a reutilização, este critério, apesar de muito geral, é importante
2. Conhecimento em reutilização na empresa e experiência prática na aplicação da reutilização	Um alto nível de experiência em reutilização torna fácil a explicação de conceitos e produtos da ED. No entanto, o treinamento em reutilização pode auxiliar no processo, se o conhecimento em reutilização for mínimo.
3. Compromisso a nível gerencial	Este é um dos critérios mais críticos para assegurar o sucesso da ED. Havendo o comprometimento dos gerentes com o projeto, recursos e tempo serão adequadamente alocados.
4. Conhecimento do Domínio	É importante que se tenha pessoas que entendam do domínio como um todo e de algumas áreas em particular. Os especialistas do domínio são uma fonte de informação e validação importante.
5. Escopo	Os domínios variam muito em tamanho. Em um domínio muito abrangente é difícil se fazer uma análise adequada. A literatura demonstra que estórias de sucesso em ED ocorrem em domínios pequenos e bem delimitados.

Tabela 4.2 – Critérios Organizacionais para escolha do Domínio

Sociais

Critério	*Descrição*
1. Necessidade de se ter informações do domínio para o treinamento de novatos	O treinamento do grupo pode ser uma característica necessária no domínio
2. Participação dos especialistas no domínio	Para capturar as informações dos especialistas no domínio e assegurar que estes especialistas vão reutilizar os produtos da ED em suas novas aplicações, a participação adequada dos especialistas do domínio é crítica.
3. Consistência dos especialistas do domínio	Os especialistas do domínio podem perceber o domínio de diferentes maneiras. Se novos especialistas forem consultados durante a ED, os produtos da mesma podem mudar significativamente. Isto pode causar atrasos e invalidação dos resultados prévios.
4. Comprometimento dos desenvolvedores de aplicações no domínio com as técnicas utilizadas pelo processo de ED	O grau de comprometimento e conhecimento dos desenvolvedores com as técnicas utilizadas pelo Odyssey. Ex. paradigma OO, CORBA, Java,DBC.

Tabela 4.3 – Critérios Sociais para escolha do Domínio

Além da identificação do domínio para a realização da ED, também é feita a identificação dos relacionamentos do domínio com as outras áreas da empresa. Assim, é feito um esboço de um diagrama de contexto do domínio, cuja representação é similar aos diagramas de contexto de um sistema da análise estruturada, onde são identificados seus limites lógicos, os atores e a interface entre o domínio e os outros domínios estratégicos para a empresa. Por exemplo, o domínio de processamento legislativo pode ter interação com o domínio de processos judiciários, o domínio do executivo municipal, entre outros. A Figura 4.4 apresenta um diagrama com as principais fases desta etapa.

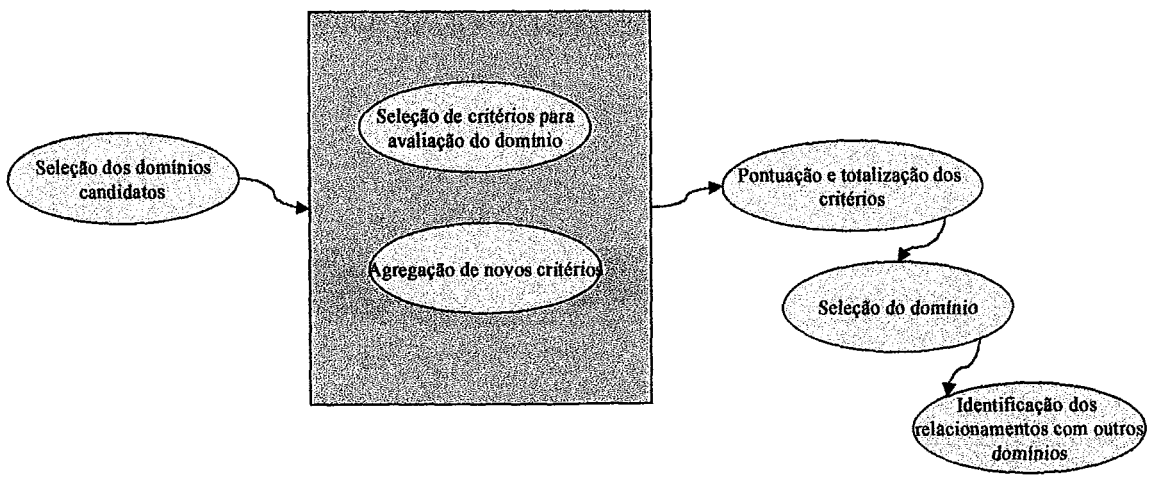

Figura 4.4 – Fases do estudo de viabilidade do domínio

Os principais produtos desta fase são:

- O domínio selecionado e os critérios e justificativas utilizadas para a seleção deste domínio;
- Um esboço do diagrama de contexto do domínio.

4.4.2 Análise do domínio

A etapa de análise do domínio do Odyssey-DE tem como objetivo a identificação dos principais conceitos e funcionalidades do domínio de aplicação, descrevendo também os limites do domínio e o seu relacionamento com os demais domínios identificados.

Conforme ressaltamos anteriormente, um dos objetivos originais da proposta do Odyssey-DE é tentar mesclar, as pesquisas em Engenharia de Domínio (ED), pesquisas em Desenvolvimento Baseado em Componentes (DBC) e, em menor escala, as pesquisas em Sistemas Baseados em Conhecimento (SBC) (WERNECK, 1995).

No Odyssey-DE, a principal ênfase dada à elaboração dos modelos da análise do domínio é na facilitação do entendimento por parte do engenheiro da aplicação dos principais conceitos e funcionalidades. As funcionalidades são importantes porque é a partir destas que serão especificados os componentes reutilizáveis. Esta abordagem é reforçada por autores como (GRISS *et al.*, 1998) e (JACOBSON *et al.*, 1997), que atestam que, no DBC, o mais importante é a identificação das funcionalidades de uma dada aplicação, pois é a partir destas que serão especificados os componentes reutilizáveis.

Os conceitos são igualmente importantes, uma vez que através destes os usuários têm um maior entendimento do domínio como um todo, além de facilitar o entendimento dos tipos[4] que interagem internamente em um componente reutilizável. As características não funcionais, conforme descrevemos anteriormente, apesar de não serem utilizadas explicitamente no contexto desta tese, também estão contempladas no modelo de características estendido e são detalhadas em (XAVIER, 2000).

Para atingir a estes objetivos, a análise de domínio é composta das seguintes fases:

- Descrever o domínio no contexto da organização;
- Elicitar o conhecimento do domínio, que é dividida em duas sub-fases:
  - Adquirir conhecimento mais específico do domínio;

[4] Utilizamos a noção de tipo, como é proposto pelo modelo OMG e ODMG, como uma referência a um objeto ou classe de um modelo OO. O tipo é o objeto em um nível de abstração alto.

- Representar o conhecimento do domínio.

A fase descrever o domínio no contexto da organização tem como objetivo a especificação do diagrama de contexto. O diagrama de contexto situa o domínio em relação ao seu escopo, limites, relacionamentos com outros domínios de aplicação e principais atores envolvidos. Este diagrama tem como base as informações coletadas na etapa de seleção. O diagrama de contexto é um diagrama que evolui de acordo com as mudanças no domínio. Seu objetivo é dar um panorama geral do domínio no contexto da organização. A medida que novas informações vão sendo adquiridas, o diagrama de contexto pode mudar, refletindo novos domínios com os quais o domínio em questão possui interação, novos atores, etc.

A sub-fase adquirir conhecimento específico do domínio é uma das mais importantes do processo. É através desta fase que as informações são capturadas e posteriormente representadas. As principais fontes de conhecimento são: os especialistas e/ou desenvolvedores de aplicações no domínio; documentação existente ou de aplicações já desenvolvidas no domínio.

A forma escolhida pelo Odyssey-DE para a elicitação do conhecimento é a descrição de cenários (JACOBSON, 1992), que posteriormente são abstraídos para diagramas de casos de uso do domínio. A eficiência da utilização de casos de uso para capturar o conhecimento para o desenvolvimento de aplicações é descrita por diversos autores consagrados na área (JACOBSON, 1992), (FOWLER, 1997), (JACOBSON *et al.*, 1997), (COCKBURN, 1999) e atualmente existem técnicas de ED que também começam a utilizar os casos de uso para representar o conhecimento no domínio (GRISS *et al.*, 1998), (CHAN *et al.*, 1998), (COHEN, 1998).

Alguns diretivas para a descrição destes cenários, a fim de facilitar na instanciação dos outros modelos do domínio, são:

- Tentar explicitar claramente os atores envolvidos no cenário e manter o mesmo nome, se o ator for mencionado mais de uma vez;
- Todos os conceitos que os especialistas consideram importantes no domínio devem ser ressaltados;
- Todas as ações consideradas importantes também devem ser ressaltadas;
- Tentar capturar de onde foi extraído o cenário;
- Tentar particionar os cenários por funcionalidades do domínio; e

- Pode-se ter mais de um cenário para descrever a mesma funcionalidade.

Apesar de adotarmos a descrição de cenários como a abordagem principal para a captura do conhecimento do domínio, nada impede que utilizemos outras abordagens em conjunto a mesma. Outras abordagens de captura do conhecimento do domínio, como a descrita em (ZOPELARI, 1998), podem ser utilizadas para facilitar o detalhamento dos cenários.

A partir das descrições dos cenários, os especialistas do domínio vão realizar uma análise destes cenários e tentar abstrair conceitos e funcionalidades que os mesmos consideram relevantes. Assim, os cenários, que são descritos a partir das funcionalidades de aplicações existentes ou necessidades de aplicações futuras no domínio, são abstraídos pelos especialistas com o objetivo de se formar o modelo de abstrações (termos) (modelo de características) e os casos de uso do domínio.

Este processo é realizado para os vários cenários capturados pelo método. No final deste processo, temos um conjunto de cenários, alguns voltados para o domínio genérico, outros ainda muito voltados para aplicações específicas, muitos cenários semelhantes, entre outras inconsistências. Com isso, é necessária a realização de um processo de concatenação e abstração dos cenários para se gerar os casos de uso do domínio. Este processo é realizado pelos especialistas em conjunto com os engenheiros do domínio. Os atores, conceitos e ações apresentados nos cenários são abstraídos até ser gerado um conjunto de casos de uso, que seja consenso entre os especialistas e engenheiros do domínio. Dos especialistas vem o conhecimento do domínio em questão e o que é ou não é importante considerar em um modelo do domínio, e dos engenheiros do domínio vem o conhecimento de como abstrair os conceitos.

Com base no cenário descrito pelos especialistas (ou retirado da documentação de uma aplicação), é feita uma instanciação inicial de um *template* de caso de uso padrão proposto pelo Odyssey-DE. Com a instanciação do caso de uso, damos início a fase de representar o conhecimento do domínio. As Figuras 4.5a e 4.5b apresentam este *template* padrão, tendo sido adaptado do trabalho de COCKBURN (1999).

Devemos ressaltar que a ligação do caso de uso do domínio com os cenários que deram origem ao mesmo deve poder ser "rastreada". Com o objetivo de capturar os relacionamentos deste caso de uso com outros casos de uso do domínio, é também utilizada uma representação diagramática. Esta visão diagramática, que também faz parte da descrição do caso de uso, captura as possíveis variações do caso de uso e suas

extensões. A representação deste diagrama é similar a representação deste tipo de diagrama em UML.

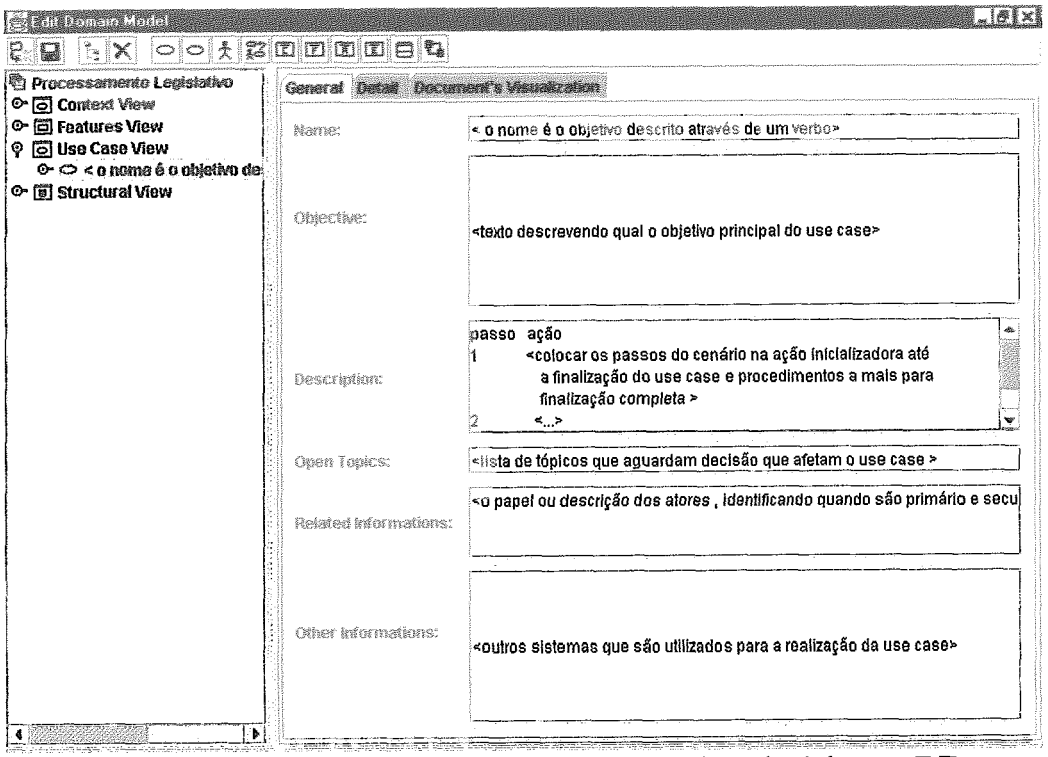

Figura 4.5 a – *Template* de caso de uso usado pelo Odyssey-DE

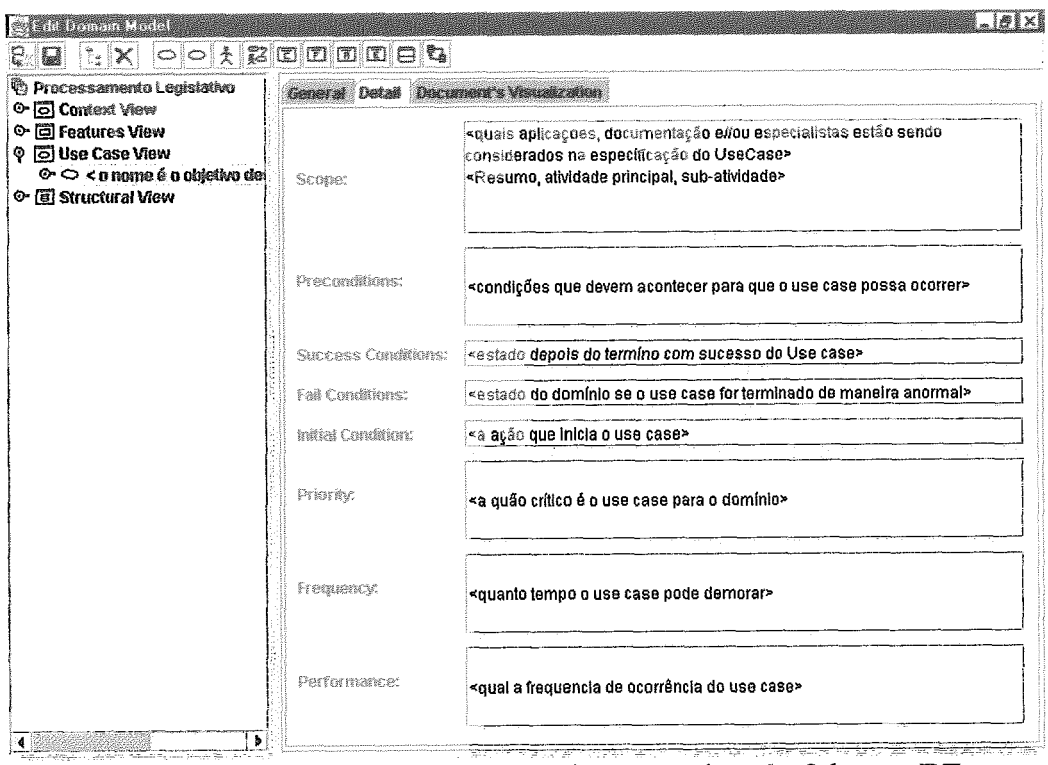

Figura 4.5 b – *Template* de caso de uso usado pelo Odyssey-DE

Com base nesta etapa de abstração dos cenários para a geração dos casos de uso do domínio, é derivado o modelo de características estendido. Assim, são descritas no modelo de características estendido, as funcionalidades capturadas como as mais importantes, bem como conceitos relacionados, formando as duas visões do modelo, i.e., visão **modelo de funcionalidades do domínio** e visão **modelo de conceitos do domínio**.

Na visão de **modelo de funcionalidades do domínio** são modeladas as funcionalidades identificadas nos casos de uso do domínio como mais importantes. São também especificados os relacionamentos entre estas funcionalidades abstratas. Estes relacionamentos são baseados em parte nas relações de "usar" e "estender" descritas nos diagramas de caso de uso, mas também em outros tipos de relações que possam surgir entre os casos de uso, de acordo com o domínio a ser modelado. Ou seja, podem ocorrer novos tipos de relacionamentos, que só ocorrem em um dado contexto e por isso estes relacionamentos variam. Além disso, podem existir funcionalidades especificadas no modelo de características estendido que podem corresponder a um conjunto de casos de uso, ou seja, não existe obrigatoriamente um mapeamento de um para um entre uma característica funcional e um caso de uso do domínio. Neste modelo funcional, podem existir funções abstratas que não são detalhadas por casos de uso, mas sim descritas em um nível mais alto de abstração, através dos padrões de domínio.

Por outro lado, a visão de **modelo de conceitos do domínio** tenta explicar, com o auxílio da descrição detalhada de cada característica, qual o significado dos principais conceitos do domínio e seus relacionamentos, o que facilita o aprendizado do domínio como um todo, pois a análise de modelos de tipos (classes) dos componentes reutilizáveis do domínio não provê esta visão geral, ou seja, a análise das classes OO relacionadas a um ou mais casos de uso do domínio, através dos respectivos modelos de classes, não provê a visão geral do mesmo. Um exemplo de um modelo de características no domínio de processamento legislativo é apresentado no capítulo 6 desta tese.

Algumas diretivas para a representação de características baseadas em casos de uso, adaptadas de (GRISS *et al.*, 1998), são as seguintes:

- Identificar, inicialmente, características essenciais e opcionais:

  √ Criar uma lista com os casos de uso mais importantes, de acordo com a freqüência de ocorrência dos mesmos nos sistemas existentes (no Odyssey-DE, quantos

cenários deram origem àquele caso de uso). Estes casos de uso podem representar características mandatórias;

√ Para estas características essenciais, identifique as outras características através dos relacionamentos de associação. Identificar quais são as restrições e observações a serem feitas em relação aos relacionamentos, de forma que esta semântica possa ser utilizada no entendimento do domínio.

- Analisar quais características são opcionais e variações de uma dada característica;
- Analisar quais casos de uso são claramente especializações de outros e quais devem entrar no modelo de característica.

Com o modelo de características bem especificado, a "escolha" de quais funcionalidades do domínio podem ser selecionadas, através da visão modelo de funcionalidades, quando da construção de novas aplicações no domínio, fica facilitada. Além disso, o modelo de funcionalidades deixa clara a organização das similaridades e diferenças no domínio, fazendo com que o usuário não fique perdido e confuso na busca de funcionalidades do domínio. A seleção de uma determinada característica, ou um conjunto destas, remete o usuário ao detalhamento do (s) caso de uso*(s)* correspondente(s).

O outro modelo derivado dos casos de uso e de conceitos constantes da documentação do domínio é o modelo de conceitos do domínio. O modelo de conceitos do domínio, conforme ressaltado anteriormente, tenta descrever os principais conceitos do domínio e o relacionamento entre estes. Este modelo corresponde a uma das visões do modelo de características, onde todas as particularidades e restrições dos conceitos do domínio são representados.

O modelo de conceitos do domínio captura uma visão geral de todos os principais conceitos que aparecem nos modelos OO. Para se entender os diversos modelos OO derivados dos casos de uso do domínio e permitir a abstração dos conceitos embutidos nestes modelos, o modelo de conceitos do domínio é de grande valia.

No entanto, para o entendimento completo dos conceitos e funcionalidades do domínio, seus sinônimos, restrições, entre outros, descritos na visão de conceitos do domínio e funcionalidades abstratas descritas na visão de funcionalidades, o modelo de características por si só não é suficiente. Por exemplo, algumas vezes existe no próprio domínio uma série de termos que descrevem um mesmo conceito. Neste caso, se cada especialista utilizar uma terminologia para designar o mesmo conceito e isso não for

controlado, não é possível modelar o domínio de forma consensual e que possa ser reutilizado. Seria um verdadeiro caos para o entendimento do domínio se o modelo de tipos de cada componente utilizasse uma terminologia diferente. Assim, é necessário que exista uma forma de se relacionar estes termos e detalhar melhor conceitos que possam confundir o usuário do domínio no completo entendimento do domínio. Este completo entendimento é característica essencial para se saber como reutilizar estes termos no desenvolvimento de aplicações no domínio.

No Odyssey-DE, utilizamos para esta descrição detalhada dos termos do domínio, o que denominamos um padrão de domínio. A estruturação do padrão de domínio é apresentada nas Figuras 4.6a e 4.6b.

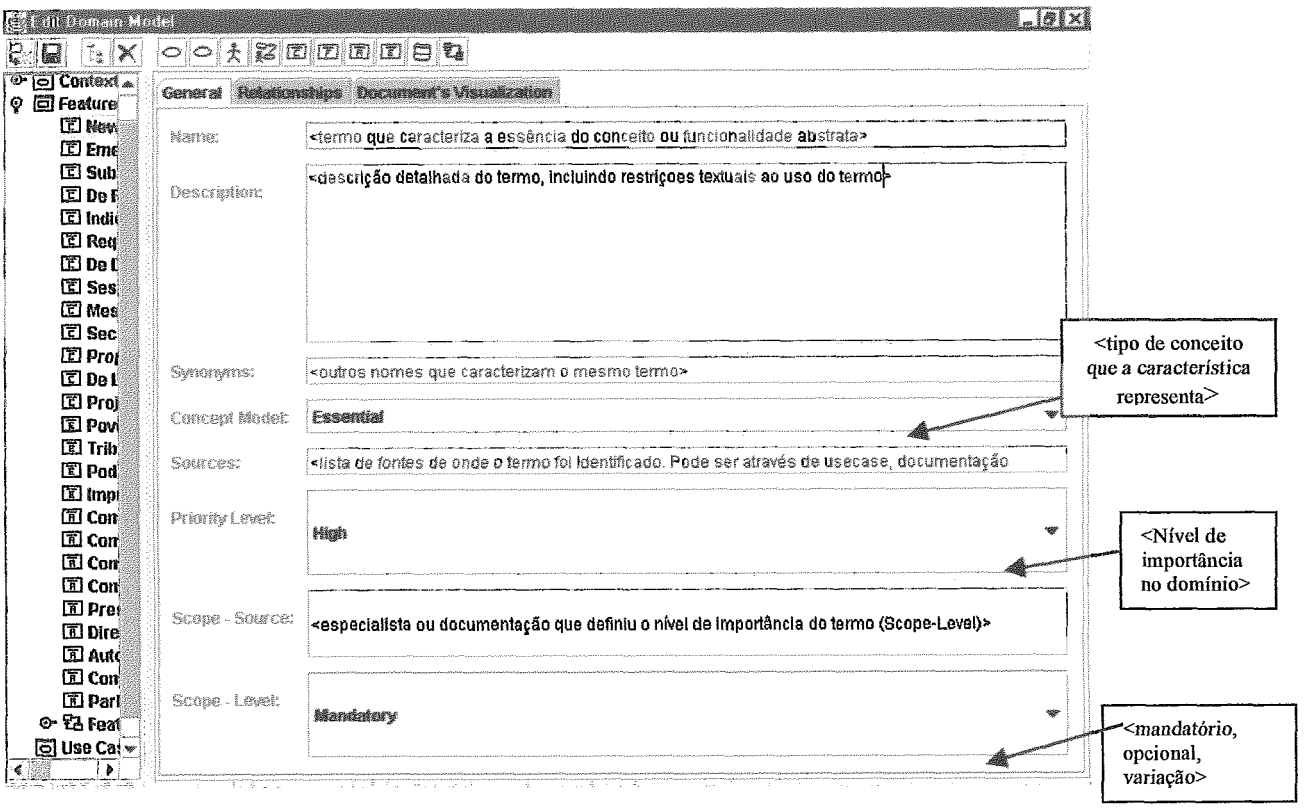

Figura 4.6 a - *Template* de um padrão de domínio

Como o principal objetivo do Odyssey-DE é o desenvolvimento de componentes reutilizáveis em um dado domínio, devemos nos preocupar com a especificação dos componentes reutilizáveis desde as etapas iniciais do processo. Na fase de análise de domínio, esta preocupação é ressaltada através da especificação dos casos de uso. Diversos autores (D'SOUZA *et al.*, 1998), (JACOBSON *et al.*, 1997), (GRISS *et al.*, 1998), (CHAN *et al.*, 1998), (BROWN *et al.*, 1997) ressaltam a correspondência dos

diagramas de casos de uso com componentes reutilizáveis do domínio. O Odyssey-DE segue a mesma linha dos trabalhos citados acima, ou seja, consideramos que existe uma correspondência quase que direta dos casos de uso do domínio e os componentes reutilizáveis em níveis mais baixos de abstração. A correspondência só não é direta por existirem componentes de suporte que não são diretamente ligados às funcionalidades do domínio e por existirem componentes em granularidade menores que a total funcionalidade dos casos de uso e que podem fazer parte de mais de um componente em maior granularidade.

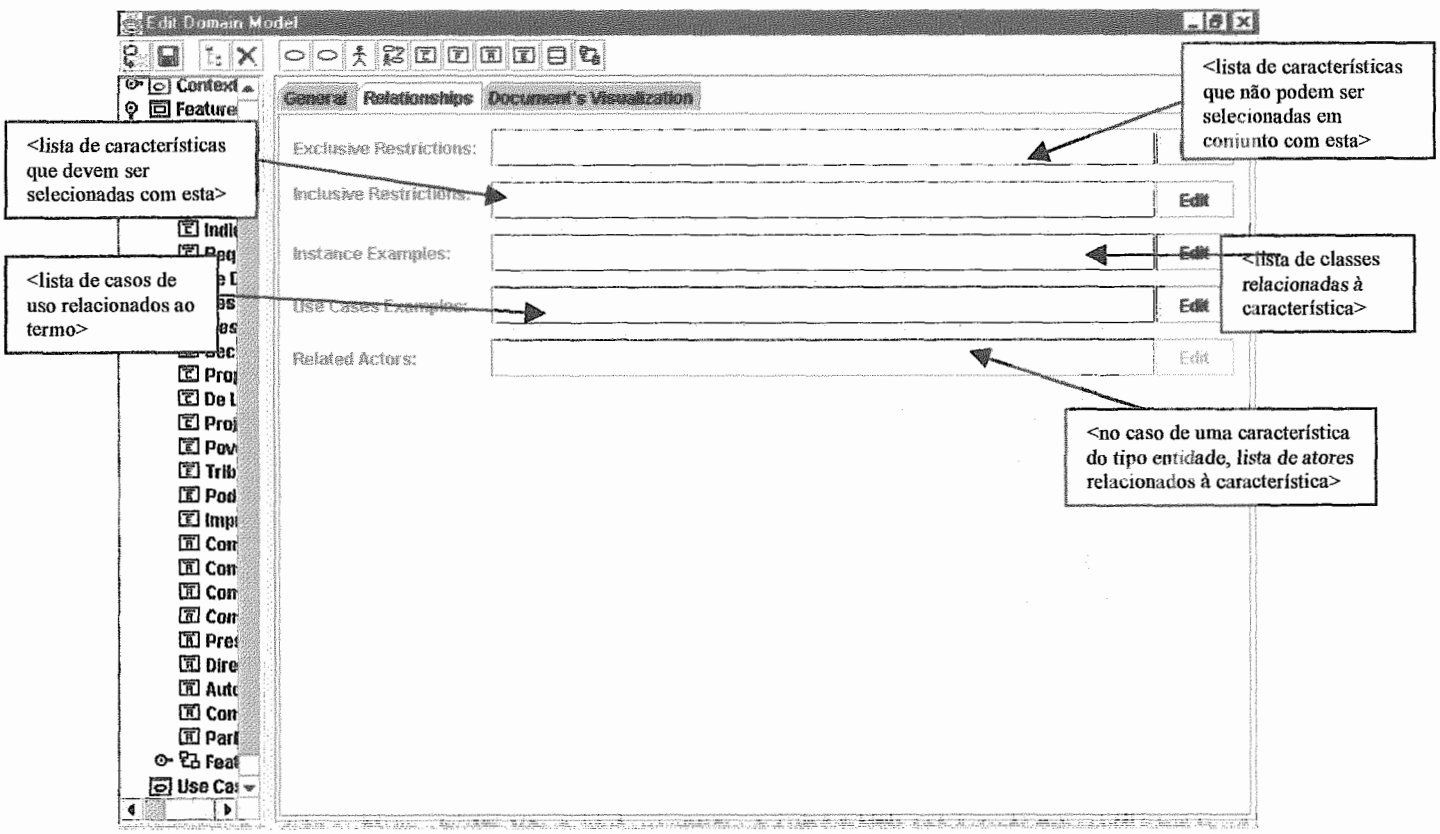

Figura 4.6 b – Continuação da *Template* de um padrão de domínio

Mesmo assim, um bom ponto de partida são os casos de uso para a derivação dos componentes do domínio. Para isto, a partir dos casos de uso do domínio, tenta-se derivar o que chamamos de modelo de colaboração[5] dos tipos que participam dos casos de uso. No entanto, para especificarmos este modelo de colaboração precisamos identificar primeiramente os tipos e ações do caso de uso. Assim, é montado um modelo inicial de colaboração, a partir da extração de tipos e ações identificados no caso de uso,

[5] Um modelo de colaboração é um modelo de interação entre os tipos participantes de um caso de uso

e com a ajuda do especialista do domínio e engenheiro do domínio é montado o modelo de colaboração dos tipos. O Odyssey-DE utiliza o diagrama de interação (notação UML) para a representação da colaboração entre os tipos em um caso de uso. Pode-se usar o diagrama seqüencial ou de colaboração, de acordo com a preferência dos engenheiros do domínio. O diagrama seqüencial é mais adequado quando as ações realizadas pelo caso de uso são seqüenciais. O diagrama de colaboração é mais adequado quando existem ações que podem ocorrer em paralelo no contexto do caso de uso.

O processo se dá da seguinte forma:

1. Cria-se modelos de colaboração para os casos de uso mais abstratos do domínio e representados por características mandatórias no modelo de características. É importante ressaltar que este também é especificado de maneira genérica, ou seja, sem ser representado por objetos de uma aplicação específica do domínio, como seria o caso do diagrama de seqüência original da notação UML;
2. Os tipos considerados nos modelos de colaboração criados são comparados entre si e com o modelo de conceitos do domínio para se tentar abstrair o máximo possível os tipos do domínio nos modelos de colaboração e construir diagramas UML para os casos de uso e seus tipos: diagramas de classe conceituais, diagramas de estado conceituais para cada tipo, levando-se em consideração os múltiplos casos de uso dos quais o tipo participa (mostra o papel do tipo nos diversos modelos de colaboração dos quais o tipo participa).
3. Analisar os casos de uso ligados aos casos de uso mandatórios através dos relacionamentos usar e estender e tentar criar modelos de colaboração, considerando-se que se irá reutilizar a estrutura criada pelos casos de uso mandatórios, ou seja, considerar que os casos de uso mandatórios são componentes reutilizáveis e identificar como podem ser reutilizados nestes novos casos de uso.
4. Definição de tipos denominados tipos de interface, que representam os serviços que os casos de uso podem disponibilizar para seus usuários[6]. É importante ressaltar a importância deste item, uma vez que através dele estaremos definindo o esboço da interface de funcionalidades do componente, representado neste nível abstrato pelo

[6] Apesar do conceito de interface de um componente ser considerado como um conceito mais ligado à fase de projeto, consideramos que no contexto do Odyssey-DE este já deve começar a ser delineado na fase de análise, uma vez que as funcionalidades do componente são capturadas nesta fase e é a partir destas que os serviços que o componente disponibiliza serão definidos.

caso de uso e seus modelos de colaboração, ou seja, modelo de classes, diagrama de estados, cenários, diagrama de sequência.

Ainda em relação aos modelos de colaboração dos casos de uso, vale a pena ressaltar que nesta etapa estamos preocupados com a identificação de que tipos devem interagir para a especificação do componente. A especificação da interface do caso de uso, especificando os serviços disponibilizados, segue o mesmo esquema, ou seja, não estamos preocupados nesta fase em detalhar precisamente os serviços mas sim identificá-los. O processo de criação dos modelos dos casos de uso baseados em casos de uso mais genéricos, e assim sucessivamente, termina quando houver um consenso entre os especialistas do domínio e engenheiros do domínio em que ponto terminar o processo.

É importante ressaltar que neste processo de especificação dos modelos dos casos de uso, podemos consultar a qualquer momento um conjunto de padrões de análise de suporte do domínio. Estes padrões ajudam no processo de abstração dos tipos do domínio, dando dicas de boas práticas de modelagem no escopo do domínio. A Figura 4.7 apresenta um *template* para o padrão de suporte à análise. Se boas práticas de modelagem do domínio forem sendo encontradas durante a especificação dos modelos de caso de uso, é especificado um padrão com base naquela modelagem e este padrão é agregado à base de padrões.

Com isso, o processo de análise de domínio e especificação dos modelos conceituais do domínio é finalizado, provendo para a fase de projeto do domínio uma série de modelos, particionados de acordo com os casos de uso do domínio. A Figura 4.8 apresenta uma visão geral de todas as atividades realizadas na etapa de análise de domínio.

Os principais modelos gerados na fase de análise de domínio são:

- Modelo de contexto do domínio;
- Descrição estruturada de casos de uso do domínio;
- Modelo de Características;
- Padrões do domínio;
- Modelos utilizando a representação UML dos casos de uso e tipos;
- Modelo de interfaces do componentes a nível abstrato (funcionalidades).

Nome: nome que descreva a essência do padrão
Sinônimos: Outros nomes para o padrão, se estes existirem
Fonte: Quem foi o autor ou a partir de onde este padrão foi descoberto
Objetivo: O que o padrão faz.
Tipo: Tipo do padrão de análise. As possibilidades são: Entidade abstrata, relacionamento, atividade
Aplicação: Tipo de problema de análise ao qual o padrão é aplicável
Ancestrais: Os padrões dos quais este padrão foi derivado (se aplicável)
Descendentes: Quais padrões são derivados destes.(se aplicável)
Padrões relacionados: Padrões similares a este (se aplicável)
Restrições: Quais são as restrições ao uso de padrões.
Exemplos de uso: Exemplos do padrão encontrados em sistemas reais
Colaborações: Diagrama mostrando os tipos participantes do padrão e suas responsabilidades.

Figura 4.7 – *Template* de padrão de análise

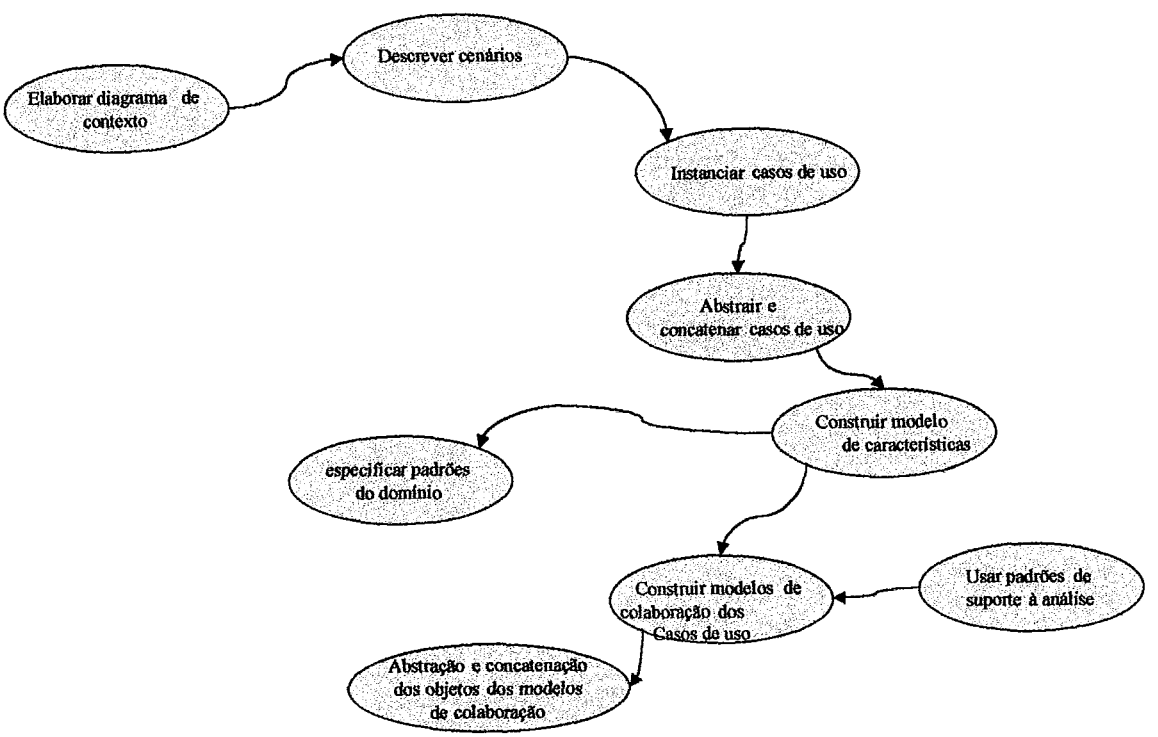

Figura 4.8 – Principais atividades da análise de domínio

4.4.3 Projeto do domínio

O projeto do domínio do Odyssey-DE é centralizado na definição arquitetural do domínio e no detalhamento dos modelos internos do componente, levando em consideração os componentes identificados na etapa de análise e a interação entre estes componentes para formar o projeto arquitetural do domínio.

Para isso, o projeto do domínio é dividido nas seguintes fases:

- Definição dos componentes e suas interfaces;
- Definição de um modelo geral de colaboração entre os componentes (modelo arquitetural de componentes), levando em consideração a utilização de componentes de suporte;
- Definição do projeto interno dos componentes.

A fase de definição dos componentes e suas interfaces inicia-se com uma análise dos modelos dos casos de uso conceituais e suas interfaces, com as definições dos serviços que os casos de uso vão disponibilizar para fora de seus limites, ou seja, deve-se centrar foco nas mensagens trocadas entre casos de uso. Para isso, a visão de funcionalidades do modelo de características deve ser consultada, com o objetivo de identificar os relacionamentos básicos entre as principais funcionalidades do domínio, mesmo que estas sejam em um nível de abstração alto.

Além da visão de funcionalidades do modelo de características, deve-se também analisar os relacionamentos entre casos de uso mais genéricos e casos de uso mais específicos. Esta abordagem é adequada no sentido de facilitar a especificação das interfaces requeridas[7] , além das interfaces providas, pelos componentes mais específicos, levando-se em consideração a necessidade de composição com os componentes mais genéricos. Um modelo com os serviços a serem providos e requeridos por cada componente (interfaces do componente) é gerado. Este modelo apresenta cada componente como um tipo que possui uma série de serviços a serem disponibilizados para os outros componentes e requer serviços de outros. É importante ressaltar que, nesta fase, estamos preocupados apenas com a definição de serviços que serão disponibilizados para o meio externo e não com serviços internos ao componente.

Em (JACOBSON *et al.*, 1997) é definida uma abordagem interessante para facilitar a definição das interfaces (serviços) entre os componentes: desenvolver um diagrama de interação entre os componentes (derivados dos casos de uso), determinando

assim quais tipos de mensagens eles trocam e em qual ordem. Desta forma, fica mais fácil visualizar que tipos de serviços estes componentes devem prover e que tipos de serviços os mesmos requerem. Um modelo estático destas interações é a própria visão de funcionalidades do modelo de características.

Com base no modelo de relacionamentos entre os componentes, é iniciada a fase de definição de um modelo arquitetural de colaboração entre componentes.

Na verdade, esta fase preocupa-se prioritariamente com a definição de uma arquitetura global do domínio, levando-se em consideração, além da forma de interação entre os componentes do domínio, questões arquiteturais como persistência, se o ambiente onde as aplicações do domínio irão rodar é distribuído, se haverá preocupações com paralelismo, entre outras. Assim, além da especificação das interfaces providas e requeridas pelos componentes a nível funcional, também deve-se especificar o que denominamos interfaces arquiteturais, ou seja, o conjunto de serviços necessários para a implementação de um dado estilo arquitetural.

O papel do engenheiro do domínio nesta fase é essencial. É ele quem vai especificar que considerações arquiteturais deverão ser levadas em conta no projeto arquitetural do domínio como um todo. Com base nesta descrição feita pelo engenheiro do domínio, uma base de padrões arquiteturais pode ser consultada para auxiliar nas decisões arquiteturais. BUSCHMANN (1996), em seu livro, apresenta algumas diretivas que podem auxiliar na seleção destes padrões. Devemos ressaltar que na verdade, conforme discutido no capítulo 2, a escolha de um determinado padrão arquitetural, ou melhor, estilo arquitetural é menos dependente do domínio em si e mais dependente da aplicação a ser desenvolvida. De acordo com as necessidades do ambiente onde a aplicação irá rodar, de questões não funcionais como desempenho e escalabilidade, o estilo arquitetural utilizado por uma dada aplicação no domínio pode mudar. Esta abordagem é assumida no Odyssey-DE, ou seja, o mais importante no domínio é a captura dos componentes do domínio e os relacionamentos dos mesmos em termos de funcionalidades, o que é especificado na etapa anterior. No entanto, existem alguns domínios onde o tipo de interação entre os componentes e as aplicações já desenvolvidas podem nos levar a privilegiar um dado estilo arquitetural como sendo o mais recomendado para aplicações naquele domínio. Mesmo assim, podem existir aplicações que não obedeçam necessariamente este dado estilo arquitetural.

---

[7] A definição de interfaces providas e requeridas foi apresentada no capítulo 2.

Estas questões arquiteturais também estão relacionadas com componentes de suporte, como um barramento para comunicação distribuída, banco de dados central, serviços de controle de transação, entre outros. Estes chamados componentes de suporte podem ser adquiridos de terceiros ou desenvolvidos a parte, podendo ser compartilhados por diversas aplicações de diversos domínios. No entanto, no desenvolvimento arquitetural de aplicações do domínio eles têm um papel importante.

Existem diversas empresas de software que atualmente comercializam estes componentes de suporte, e portanto, baseada na descrição da funcionalidade arquitetural que os mesmos devem prover, pode-se adquirir estes componentes de terceiros. Para auxiliar na escolha dos componentes de suporte disponíveis, um catálogo com os mesmos pode ser útil, utilizando *templates* de documentação tais como os propostos em (MURTA, 1999).

Na fase de definição do projeto interno dos componentes é definida em maior detalhe a estrutura interna de cada componente. Nesta fase, são definidos precisamente os tipos[8] que participarão dos componentes, definindo seus atributos e métodos de forma precisa dentro do componente e a interação entre os vários tipos. Todos os modelos de análise relacionados a um dado componente são refinados, tentando-se definir com mais detalhes a interação entre os tipos nos vários modelos. Assim, novos tipos podem ser agregados aos componentes e a forma de modelagem dos tipos também pode ser modificada para a produção de modelos mais robustos, flexíveis e extensíveis. Para isso pode ser utilizada uma base de padrões de projeto que é "consultada" para melhorar a forma de modelar os tipos internos a um componente.

A próxima atividade é o refinamento do modelo interno dos tipos participantes dos componentes. Para cada tipo, com base nos papéis desempenhados pelos mesmos nos diversos componentes, montar a estrutura da classe que corresponderá ao tipo e o diagrama de estados. Para cada classe representada, tem-se um conjunto de interfaces ligadas aos papéis desempenhados em cada componente. Também nesta atividade, os padrões de projeto podem auxiliar. É importante ressaltar que um mesmo tipo pode participar de diversos componentes, desempenhando em cada um deles um papel específico. O modelo de interface dos componentes da etapa de análise é refinado e a partir destes podem ser geradas diversas interfaces providas e requeridas de acordo com a nova modelagem interna dos componentes. Neste caso, um componente a nível

[8] Nesta fase os tipos são transformados em classes com múltiplas interfaces

conceitual pode dar origem a mais de um componente a nível de projeto ou um componente a nível conceitual pode ser fragmentado em vários componentes a nível de projeto. Desta forma, o modelo de colaboração de componentes arquiteturais é detalhado, refletindo os tipos internos ao componente que participam da interface daquele componente, mostrando assim as conexões precisas das interfaces dos componentes.

Na verdade, este modelo é um modelo em duas camadas, onde temos duas visões:

- Interação entre componentes;
- Interação interna dos tipos que compõem o componente.

As principais atividades que compõem as fases do projeto do domínio são apresentadas na Figura 4.9:

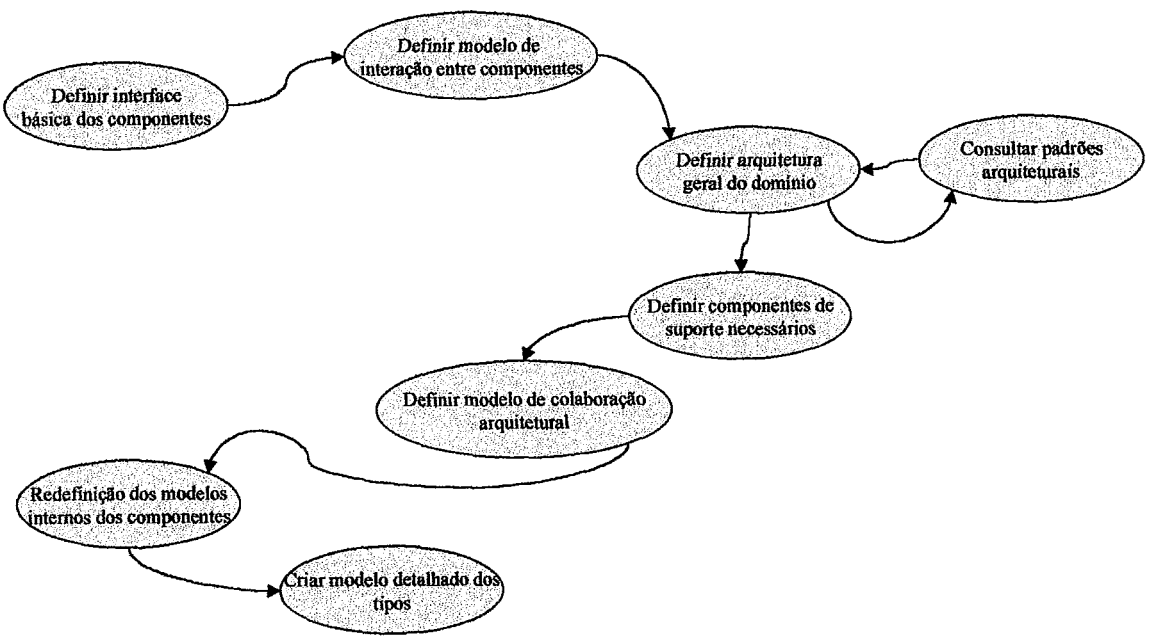

Figura 4.9 – Principais atividades do projeto do domínio

Os principais modelos gerados são:

- Modelo de serviços (interfaces) dos componentes;
- Modelo arquitetural de colaboração entre componentes do domínio;
- Modelo de classes e diagrama de estado de cada tipo participante de componentes do domínio;
- Modelo arquitetural em dois níveis.

4.4.4 Implementação do domínio

Esta etapa diz respeito à implementação de componentes que atendam às especificações arquiteturais definidas na etapa anterior. Para isso, conforme descrito anteriormente, temos duas estratégias que podem ser seguidas de forma concomitante:

1. Codificação de componentes em uma linguagem de programação OO: A codificação de componentes pode ser feita utilizando uma linguagem de programação OO, como JAVA. Esta codificação pode ser um processo semi-automático, uma vez que todo o interfaceamento entre classes, definição de atributos e chamada de métodos foi especificado. Na codificação dos componentes, temos três atividades precisas: codificação das classes do domínio, codificação do relacionamento das classes para compor um componente e empacotamento final do componente. Na codificação das classes do domínio, pode ser utilizada uma ferramenta automática onde são codificados os atributos, assinaturas e corpo de métodos e a definição das interfaces.
2. Utilização de componentes legados: A utilização de componentes legados passa primeiramente pela checagem se o componente legado atende adequadamente às necessidades do componente a nível abstrato, ou seja, se atende às necessidades funcionais de um dado componente abstrato, o qual o mesmo quer representar a nível implementacional. No entanto, deve ser definido e implementado um invólucro que faça o interfaceamento deste componente com os demais componentes com os quais o mesmo mantém relacionamento.

Feita a codificação dos componentes do domínio, estes componentes são agregados à base de componentes reutilizáveis, conjuntamente com os componentes de suporte adquiridos de terceiros. A próxima atividade é o teste dos componentes codificados e da interação entre eles mesmos e com os componentes de suporte. O teste da funcionalidade interna dos componentes pode ser feito com base nos casos de uso que deram origem ao mesmo, utilizando técnicas como as descritas em (VIEIRA, 1998), (HERBERT *et al.*, 1999).

O teste da interação entre os componentes pode ser feito com base no diagrama de interação arquitetural, nível de interação entre componentes, e observação do comportamento do modelo.

Uma vez que os componentes tenham sido adequadamente codificados e testados, pode ser gerada uma documentação, baseada no *template* apresentado em

(MURTA, 1999), para cada um dos componentes reutilizáveis do domínio. Na Figura 4.10, apresentamos as principais atividades desta etapa.

Os principais resultados são:

- Conjunto de componentes reutilizáveis do domínio;
- Conjunto de componentes de suporte;
- Documentação dos componentes reutilizáveis.

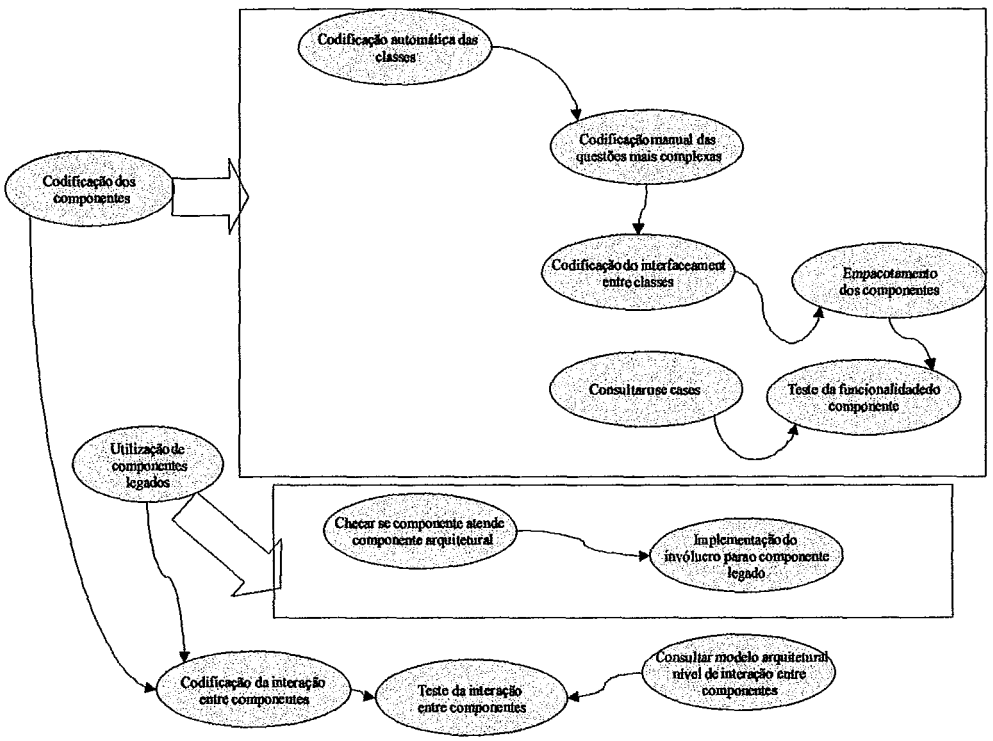

Figura 4.10- Principais atividades da implementação do domínio.

4.5 *Odyssey: Infra-estrutura de suporte a Reutilização*

De acordo com ARANGO (1988), KRUEGER (1992), COHEN (1994) e JACOBSON (1999), um ponto importante para o sucesso na adoção de um processo de reutilização de software é o suporte de ferramentas automatizadas. Desta forma, o suporte provido pela infra-estrutura de reutilização Odyssey, disponibiliza a automatização necessária para que as etapas do Odyssey-DE sejam realizadas de maneira eficiente, auxiliando o engenheiro do domínio na especificação dos componentes do domínio e sua posterior reutilização. Assim, a infra-estrutura Odyssey, além de prover suporte ao processo de engenharia de domínio Odyssey-DE, também provê suporte ao processo de engenharia das aplicações, denominado Odyssey-EA (MILLER, 2000). Sem o auxílio da infra-estrutura Odyssey, a execução das atividades

do Odyssey-DE, e do Odyssey-AE, implicaria em grande quantidade de trabalho manual, seja na criação e manutenção dos relacionamentos entre os componentes especificados nas diversas etapas do processo, seja no auxílio ao desenvolvimento de novas aplicações no domínio.

Assim, a infra-estrutura Odyssey é uma ferramenta para suportar a reutilização no desenvolvimento de software, provendo suporte para o desenvolvimento **para** reuso, através do suporte automatizado às etapas do Odyssey-DE e ao armazenamento de seus produtos, e desenvolvimento **com** reuso, com o suporte ao Odyssey-AE e a busca e recuperação dos componentes reutilizáveis. Devido ao tratamento especial dado no Odyssey para a busca, recuperação e armazenamento de componentes reutilizáveis, esse aspecto da Odyssey será tratado em detalhes no capítulo 5.

4.5.1 Características Arquiteturais

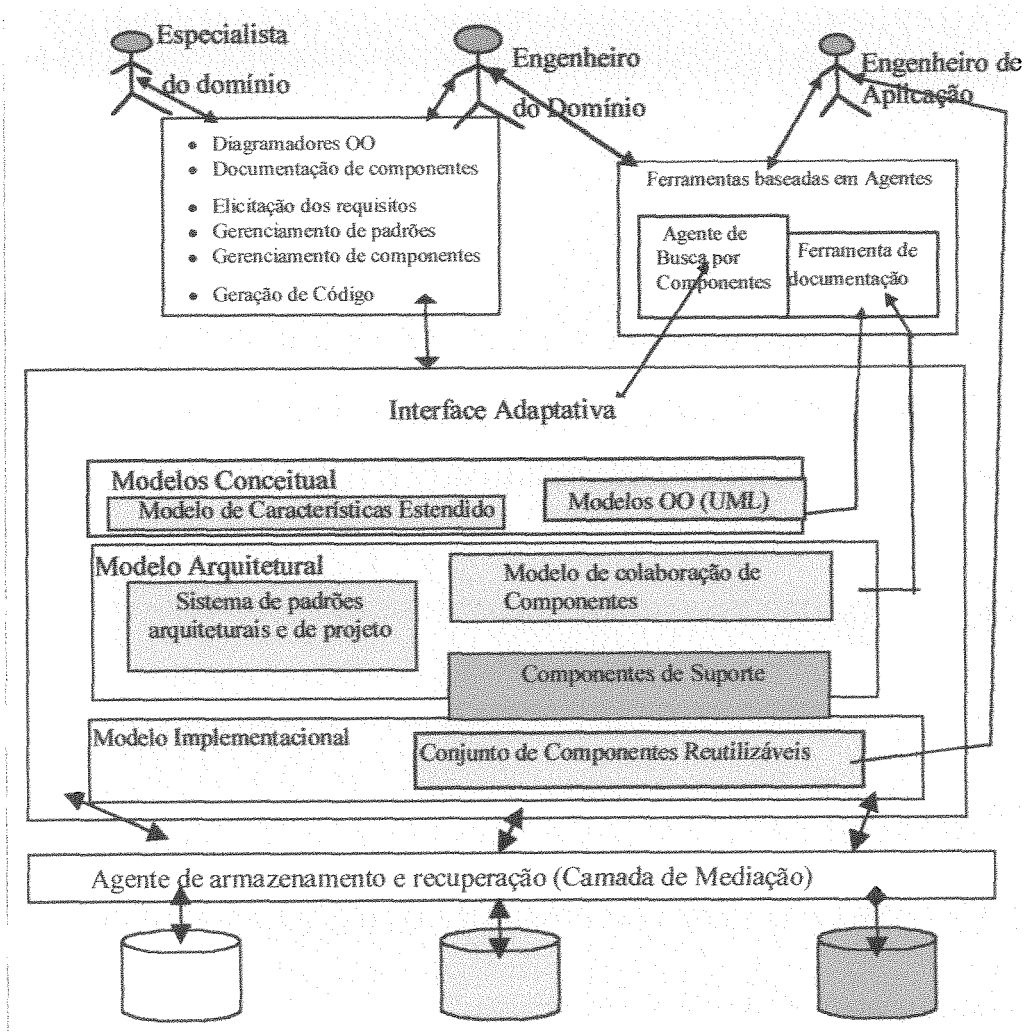

Figura 4.11 - Visão diagramática dos principais elementos da infra-estrutura Odyssey

A infra-estrutura Odyssey pode ser vista como um arcabouço onde modelos conceituais, arquiteturas de software, e modelos implementacionais são especificados para domínios de aplicação previamente selecionados, permitindo a reutilização destes modelos na especificação de aplicações neste domínio. A Figura 4.11 apresenta uma visão global dos principais elementos da infra-estrutura. Os modelos de domínio são especificados e, posteriormente, evoluídos segundo as atividades do Odyssey-DE, estando estas atividades suportadas por um conjunto de ferramentas, dentre elas, ferramentas para elicitação de requisitos, ferramentas para armazenamento e recuperação dos modelos, ferramentas para criação de modelos, ferramenta para a criação de ligações entre modelos e ferramenta para busca por componentes reutilizáveis.

Os principais usuários da infra-estrutura são o engenheiro do domínio, o especialista do domínio e o engenheiro de aplicação responsável pelo desenvolvimento de aplicações no domínio. O engenheiro do domínio e o especialista utilizam o Odyssey principalmente para especificar e evoluir os conceitos do domínio. O engenheiro de aplicação utiliza esta infra-estrutura para obter conhecimento sobre o domínio da aplicação e reutilizar este conhecimento na especificação de sua aplicação.

4.5.2 Principais ferramentas da infra-estrutura Odyssey

A infra-estrutura Odyssey, para apoiar os processos de desenvolvimento para e com reuso, disponibiliza um conjunto de ferramentas que dão apoio e/ou automatizam atividades específicas destes processos, sejam elas de desenvolvimento, planejamento, acompanhamento gerencial ou controle de qualidade. Detalhamos algumas destas ferramentas nesta seção, particularmente aquelas relacionadas à automatização do Odyssey-DE. Um descrição mais detalhada sobre as mesmas pode ser encontrada em (WERNER *et al.*, 1999) (WERNER *et al.*, 2000), (MILLER, 2000), (MURTA, 1999), (TRANNIN *et al.*, 1999), (BRAGA *et al.*, 2000a).

Podemos citar como exemplos a ferramenta de elicitação de requisitos (TRANNIN *et al.*, 1999), documentação de componentes (MURTA, 1999), os diagramadores genéricos, o gerador de código executável (WERNER *et al.*, 2000), especificação de arquiteturas específicas de domínio (XAVIER, 2000), ferramenta para planejamento e análise de risco (BARROS, 2000), sendo esta ferramenta primordial para a completude do processo de gerenciamento do Odyssey, em conjunto com a ferramenta de acompanhamento de processos (MURTA, 2000); e o navegador

inteligente (BRAGA *et al.*, 2000c), sendo este último também objeto de desenvolvimento desta tese e, portanto, detalhado no capítulo 5.

4.5.2.1 Ferramenta cooperativa para Aquisição do Conhecimento

A aquisição de conhecimento na infra-estrutura Odyssey, conforme já descrito, é baseada na criação de cenários e sua posterior instanciação em casos de uso do domínio. Assim, a ferramenta cooperativa para aquisição de conhecimento foi desenvolvida com o objetivo de apoiar a criação cooperativa de cenários e casos de uso. No caso de interações síncronas, o suporte oferecido seria o mesmo encontrado em editores de texto cooperativos, utilizando fontes com cores diferentes para cada autor, por exemplo. Um melhor aproveitamento pode ser conseguido, no entanto, através de interações assíncronas, devido à pouca disponibilidade de tempo do especialista.

Duas filosofias de controle de concorrência são utilizadas por esta ferramenta: uma livre (ou sem controle) e outra baseada em bloqueio vinculado. No caso da edição de cenários não é feito nenhum controle, deixando livre a criação dos mesmos. No entanto, na edição de casos de uso é adotada a política de bloqueio vinculado. Cada campo do caso de uso (Figura 4.5 a e b) pode ser bloqueado por um usuário e, a princípio, ocorre somente enquanto este estiver usando o sistema. Caso se queira manter o bloqueio entre sessões, visando o trabalho assíncrono, o usuário deve explicitar esta opção.

4.5.2.2 Editor de diagramas

O editor de diagramas tem como principal objetivo o suporte à criação dos modelos definidos pelo Odyssey-DE/EA (WERNER *et al.*, 1999). A notação utilizada, conforme visto neste capítulo é similar ao UML, com algumas construções adicionais para suportar o desenvolvimento de componentes reutilizáveis. Os principais modelos gerados são: modelo de características estendido (visão conceitual e funcional), modelos de caso de uso do domínio, modelos de classes, modelos dinâmicos (diagrama de sequência e estados) e demais modelos constantes da notação UML.

Todos os modelos especificados são ligados entre si através de ligações semânticas, as quais denominamos, no contexto do Odyssey, de rastro ("trace"). Assim, o entendimento do domínio como um todo e seus componentes em particular fica facilitado, além de guiar a escolha de quais modelos, em todas as fases do desenvolvimento, serão reutilizados no desenvolvimento de uma determinada aplicação.

A infra-estrutura Odyssey provê suporte automatizado para este tipo de rastreabilidade ("traceability").

4.5.2.3 Documentação dos Componentes do Domínio

Para dar apoio à criação e disponibilização de uma documentação padrão de componentes, o Odyssey apresenta um *framework* de documentação, denominado *FrameDoc* (MURTA, 1999). O *FrameDoc* permite a criação de *templates* de documentação, que são posteriormente preenchidos para cada tipo de componente (ex. classe, caso de uso, diagrama de estados, etc.). Os tipos de campos fornecidos para a construção do *template* são: texto, memo, *checkbox*, *radiobutton*, som, imagem, vídeo e HTML.

Algumas das características do *FrameDoc* são: exportação da documentação utilizando o padrão HTML; utilização de hiperligações no campo HTML para outros documentos ou páginas na Internet; criação de padrões de documentação, representados pelos *templates*; geração de documentação modular; utilização de multimídia na documentação; visualização da documentação a ser gerada; geração da documentação implícita ao componente (ex. diagramas UML).

4.5.2.4 Gerador de Código

A ferramenta de geração de código fonte do Odyssey foi definida e implementada por Pinheiro (WERNER *et al.*, 2000), de acordo com as características necessárias à implementação de componentes de domínio do Odyssey. Ela permite a geração do código fonte das classes, na fase de implementação dos componentes do domínio, baseada em suas definições de atributos e serviços. Até mesmo a geração do corpo dos serviços descritos no modelo estrutural pode ser executada através da descrição do seu conteúdo. A definição destes algoritmos internos dos métodos é feita numa meta-linguagem, semelhante ao Pascal.

A tarefa de geração é executada tomando como base a escolha da linguagem de desenvolvimento. Esta geração, realizada através de um processo de tradução, pode ser feita, atualmente, em três linguagens diferentes: Java, Delphi e C++.

4.5.3 Características implementacionais

A implementação da infra-estrutura Odyssey é totalmente voltada para a especificação de boas práticas de projeto e programação, privilegiando a flexibilidade de extensão e portabilidade. A linguagem Java (JAVAWEB, 2000) foi adotada, visando

facilitar a portabilidade da infra-estrutura, além de permitir a implementação de boas práticas de modelagem OO. Atualmente, a linguagem Java é considerada a linguagem mais próxima de uma linguagem voltada para o desenvolvimento de componentes.

A infra-estrutura é dividida em 10 pacotes principais, conforme apresenta a Figura 4.13:

- Ações, que possui classes que implementam todo o tipo de ação que pode ocorrer na infra-estrutura e em seus diagramadores;
- Componentes, cujas classes implementam padrões genéricos como janelas, painéis e elementos gráficos para serem utilizados na implementação de novas funcionalidades na Odyssey. Todas as funcionalidades até agora implementadas seguem os padrões especificados neste pacote;
- Diagrama, que descreve os elementos básicos léxicos a serem utilizadas nos diagramas implementados pela Odyssey. Classes com as funcionalidades básicas de todos os elementos que podem constar de um diagrama são encontradas neste pacote. Classes de pacotes com a implementação de diagramas específicos herdam das classes deste pacote;
- Ferramentas, que contêm pacotes com a especificação de todas a ferramentas disponibilizadas pela Odyssey, com exceção do editor de diagramas, que é especificado por pacotes específicos para cada diagrama;
- Goa, que é o pacote que tem a interface *socket* para conexão com o Sistema Gerenciador de Objetos (SGO) GOA++ (MAURO *et al.*, 1997). Toda a comunicação com o SGO e com a camada de mediação é intermediada por este pacote. Por fazer parte da estrutura utilizada para o armazenamento e recuperação dos componentes, este pacote será descrito em detalhes no capítulo 5;
- Ícones, que possui todos os ícones utilizados na Odyssey, desde ícones de botões, até ícones de janelas;
- Léxico, que contém subpacotes para cada tipo de diagrama encontrado na Odyssey. Cada um destes subpacotes contém elementos léxicos que permitem a criação, apresentação e modificação de representações para elementos constantes dos modelos da Odyssey;
- Modelo, que é o pacote principal da infra-estrutura Odyssey, pois contém todos os elementos semânticos que fazem parte dos modelos utilizados na especificação dos

processos Odyssey-DE e Odyssey-EA. Cada um de seus subpacotes contém elementos semânticos divididos por tipo de modelo utilizado;

- PainéisRepresentação, que contém subpacotes com painéis utilizados na representação dos elementos semânticos para edição;
- Representação, que indica como cada elemento léxico deve ser apresentado pela infra-estrutura Odyssey;
- Rule, que é o pacote responsável pela gerência da base de regras utilizado pela Odyssey, inclusive para acesso ao mecanismo de inferência utilizado por esta base de regras. Atualmente, a única ferramenta que faz uso desta base de regras é o Agente de Navegação, ferramenta esta descrita em detalhe no capítulo 5.

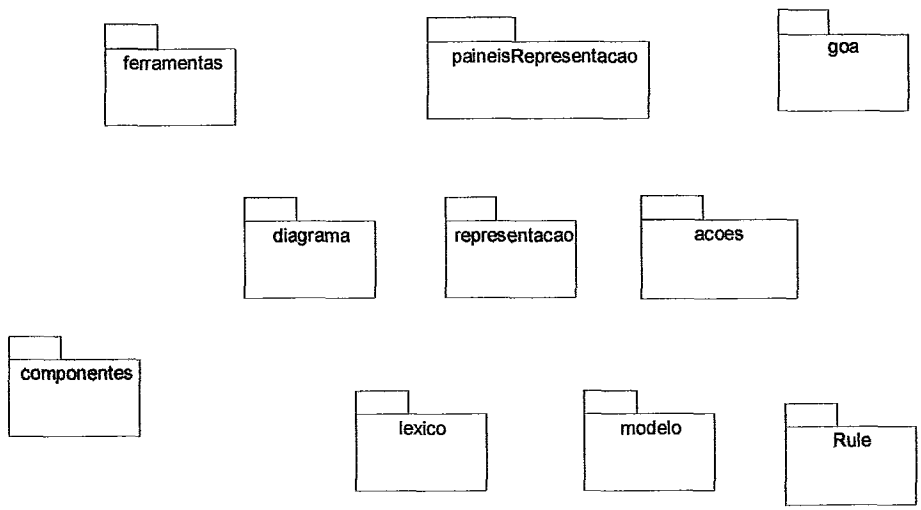

Figura 4.13 – Principais pacotes da infra-estrutura Odyssey

A implementação da infra-estrutura Odyssey de forma geral segue o padrão de projeto *Document-View*, que é uma variação do padrão de projeto *Model-View-Controller* (GAMMA *et al.*, 1994). No caso das ferramentas especificadas no pacote ferramentas, todas seguem o padrão *Model-View-Controller* puro. Assim, tornou-se um padrão no Odyssey a utilização destes padrões de projeto na implementação de novas ferramentas. Além destes dois padrões de projeto, são utilizados, quando necessários, outros padrões como *Singleton*, *Publisher-Subscriber*, entre outros. Não é objetivo desta tese descrever todos os pacotes implementados na infra-estrutura. Apresentamos a

seguir somente os principais pacotes, de forma genérica, a saber: o pacote Modelo, o pacote Diagrama e o pacote GOA, sendo este desenvolvido no contexto desta tese.

4.5.3.1 Pacote Modelo

Neste modelo genérico (Figura 4.14), a classe *ModeloAbstrato* representa uma abstração de alto nível de um modelo, ou elemento do domínio, definindo um protocolo padrão de atributos e serviços que deve ser seguido por todos os modelos e itens que compõem os elementos da infra-estrutura. São especializações desta classe os seguintes conceitos: *ItemModelo*, *ConjuntoModelos* e *Modelo*.

A classe *ItemModelo* representa os elementos dos diferentes modelos, podendo ser visto como folha da árvore de modelos de uma aplicação/domínio. Entretanto, por ser um modelo abstrato, o item também pode ter outros itens de modelo como subclasses. Uma aplicação ou domínio, no Odyssey, é representada por um conjunto de modelos, sendo *ConjuntoModelos* a classe que representa esta estrutura. A classe *Modelo* representa o segundo nível da árvore de Modelos de uma aplicação/domínio, sendo o primeiro nível composto pela aplicação/domínio (*ConjuntoModelos*) e os demais níveis compostos por itens de modelo (*ItemModelo*). O modelo funciona, assim, como um conjunto de itens de modelos.

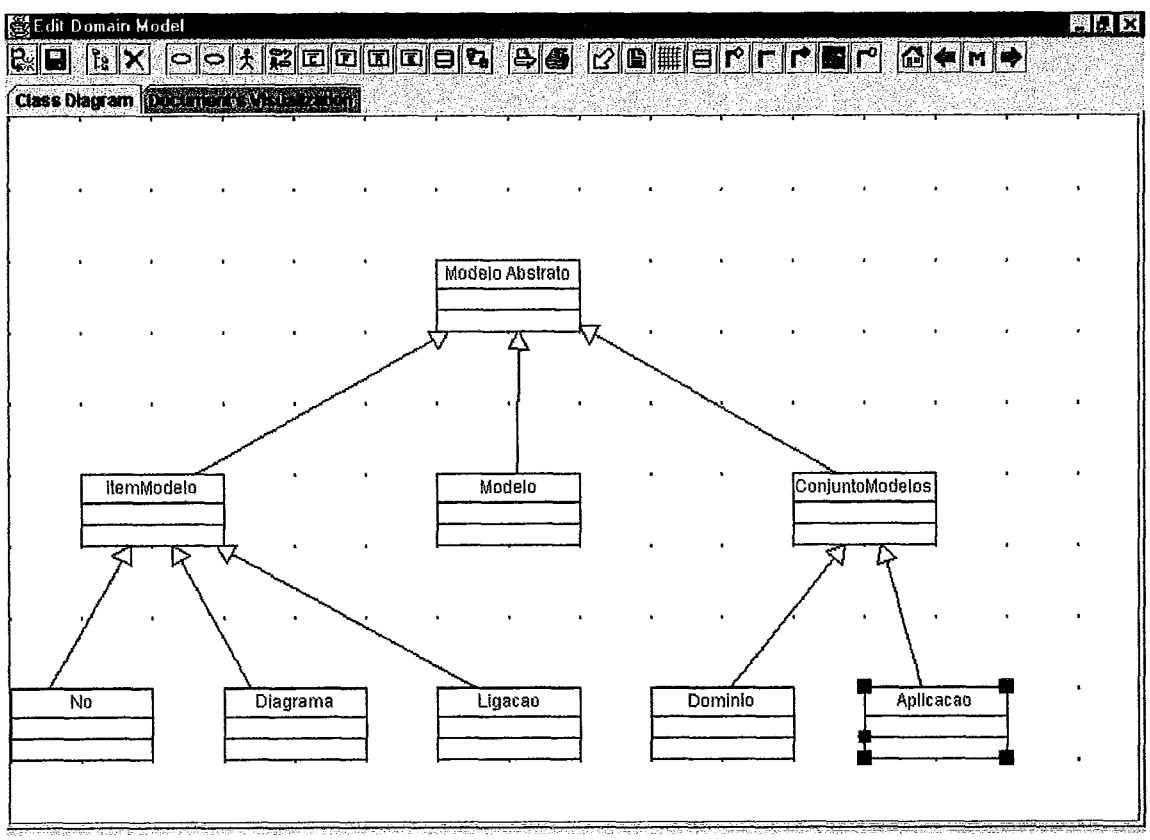

Figura 4.14 – Principais classes do pacote Modelo

A classe *Dominio* representa um domínio de aplicações. Um domínio pode estar relacionado com diversas aplicações. Cada aplicação desenvolvida dentro da infra-estrutura se encontra no contexto de um determinado Domínio. A classe *Aplicacao* representa, por sua vez, as aplicações do domínio. As classes *Diagrama*, *No* e *Ligacao* são as especializações dos itens de modelo que compõem um modelo. Assim, todos os modelos que representam o domínio na infra-estrutura especializam a classe *Modelo*, os diagramas, a classe *Diagrama*, os nós, a classe *No* e as ligações entre os nós, a classe *Ligacao*. As classes *Diagrama*, *No* e *Ligacao* estão em uma subdivisão do pacote modelo, chamado *Modelo.Generico*.

4.5.3.2 Pacote Diagrama

Neste modelo básico para todos os diagramas criados na infra-estrutura (Figura 4.15), a classe *ObjetoDiagramacao* representa uma abstração de alto nível de um item do diagrama, de forma genérica. A classe *NoPadrao* representa um nó de um grafo representável pela Odyssey e a classe *ArestaPadrao* representa uma ligação do grafo, que na verdade é um diagrama. Estas classes estão contidas em uma classes que representa um *DiagramaPadrao*, onde atributos e serviços necessários a todos os diagramas são especificados.

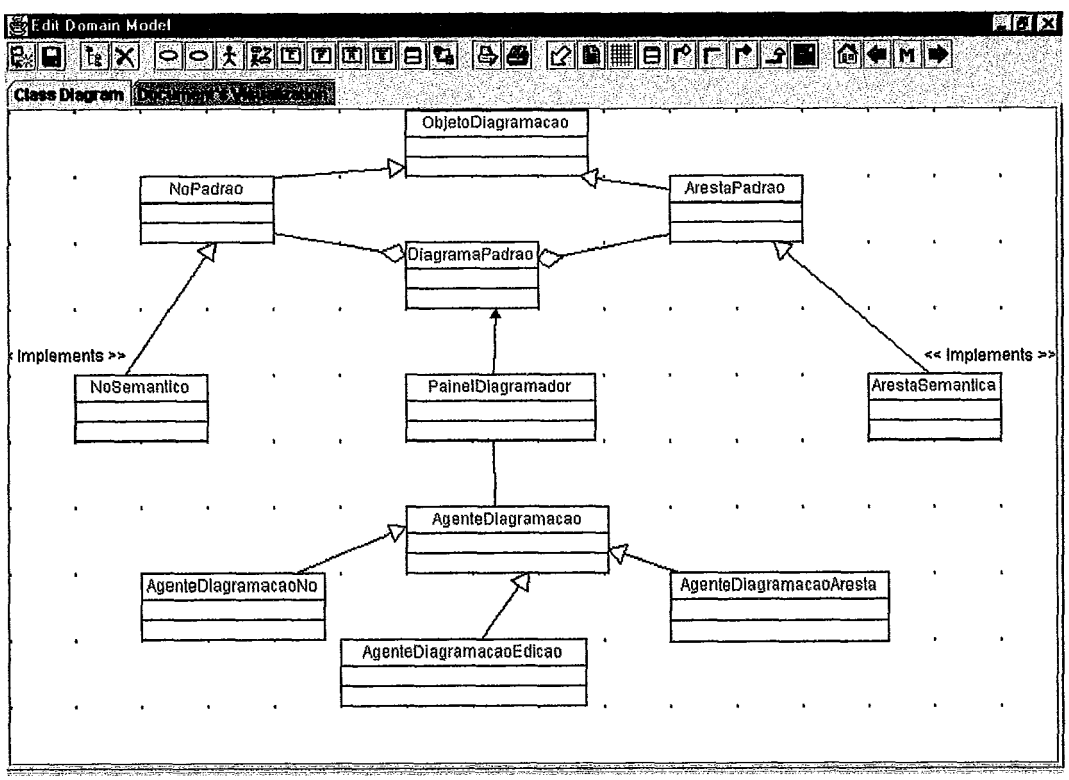

Figura 4.15 – Principais classes do pacote Diagrama

A classe PainelDiagramador representa um painel onde um diagrama pode ser criado. Para a criação dos elementos do diagrama, esta classe está ligada à classe AgenteDiagramacao, que é, de forma genérica, um agente para a criação de elementos no diagrama. São especializações desta classe as classes AgenteDiagramacaoNo, que é responsável pela criação de nós no diagrama, AgenteDiagramacaoAresta, que é responsável pela criação de aresta e AgenteDiagramacaoEdicao, que é responsável pela edição destes itens no diagrama. Todas estas classes são especializadas em diagramas específicos do Odyssey, permitindo a criação de diagramas para o modelo de classes, características, estados, sequencia, etc.

As classes *NoPadrao* e *ArestaPadrao* implementam ainda, respectivamente, as interfaces *NoSemantico* e *LigacaoSemantica*, que são a forma de ligação das mesmas com as classes do pacote modelo que representam os elementos semânticos do Odyssey.

4.6 Conclusões

Apresentamos neste capítulo o processo de engenharia de domínio Odyssey-DE, cujo objetivo maior é tentar mesclar uma abordagem de engenharia de domínio com o desenvolvimento de componentes. O processo considera a reutilização de componentes em todas as suas etapas, além de enfocar o processo de abstração e realização da abstração, através do mecanismo de rastro. Esta perspectiva é compatível com a abordagem considerada ideal por KRUEGER (1992). Desta forma, o Odyssey-DE apresenta este diferencial em relação aos demais processos de ED apresentados no capítulo 2.

O Odyssey-DE tenta unir o que há de melhor nas duas abordagens, capturando as abstrações do domínio e como estas abstrações estão relacionadas, através do modelo de características estendido, e também se preocupando em como desenvolver componentes e como estes componentes podem cooperar para a especificação de uma dada aplicação.

Outro ponto que pode ser considerado como diferencial do Odyssey-DE é o suporte de uma infra-estrutura que permite o apoio automatizado de suas etapas, o que é considerado por JACOBSON (1999) e outros (GRISS, 2000), (GOMMA, 2000) como primordial para qualquer processo de desenvolvimento atual.

Alguns pesquisadores da área (SAMETINGER, 1997), (SYZERPERSKI, 1998), (GRISS, 2000) ressaltam também a importância de um repositório para o armazenamento de componentes e técnicas de busca e recuperação dos mesmos. A infra-estrutura Odyssey provê este tipo de suporte, seja através do armazenamento de

componentes no SGO GOA++, e recuperação de componentes distribuídos através da estrutura de mediação, seja através da ferramenta de busca inteligente, que implementa o estado da arte em relação às técnicas de filtragem de informação. Estas abordagens são apresentadas em detalhes no capítulo 5.

Analisando os pontos levantados no capítulo 2, seção 2.5, podemos dizer que o Odyssey-DE atende a estes pontos da seguinte maneira:

- Especificar modelo de abstrações do domínio, que não só leve em consideração aspectos funcionais das aplicações do domínio mas também considere conceitos e outras características importantes: **o modelo de características estendido atende a este requisito.**
- Criar um formalismo que permita a verificação de consistência entre os modelos e também a consistência na seleção de abstrações que são conflitantes: **O Odyssey-DE atualmente não possui nenhum formalismo para esta checagem de consistência. Apenas é realizada uma checagem de inconsistências entre as características, durante o desenvolvimento de aplicações** (MILLER, 2000) **o que ainda é considerado insuficiente para as reais necessidades nesta área. Assim, neste requisito o Odyssey-DE ainda não atende satisfatoriamente. No entanto, um trabalho neste sentido está em andamento.**
- Definir um mecanismo de ligação entre os modelos de abstração e os modelos de especificação dos componentes: **O Odyssey-DE atende este requisito através do mecanismo de rastro.**
- Seguir uma abordagem essencialmente DBC, que possui o detalhamento necessário para tal, com definição de interfaces precisas, levando em consideração as restrições do estilo arquitetural adotado na colaboração entre os componentes: **O Odyssey-DE atende, parcialmente este requisito, uma vez que modelos de interface podem ser definidos, estilos arquiteturais são considerados, mas ainda existe a falta de ferramental automatizado para apoiar tais conceitos, uma vez que este tipo de consistência é muito difícil de ser realizada de maneira manual.**
- Automatizar o processo, uma vez que as ligações entre os modelos, checagem de consistência, entre outros, são atividades complexas de serem realizadas manualmente: **Através da infra-estrutura** Odyssey, **é proposta a construção de todo o ferramental necessário, embora esta ainda esteja em construção.**

É importante ressaltar que os aspectos relacionados ao armazenamento, busca e recuperação de componentes na infra-estrutura Odyssey foram todos especificados e implementados no contexto desta tese, seja através de pacotes específicos da infra-estrutura Odyssey, tais como os pacotes Goa, Agente (especificado no pacote Ferramentas) e Rule, que foram implementados em Java (JAVASUN, 2000), seja através da implementação da arquitetura de mediação, que foi implementada em C++ (INPRISE,2000). Ao todo foram especificadas cerca de 70 classes na implementação da arquitetura de mediação e cerca de 50 classes na implementação dos pacotes Java.

5 Odyssey-Search : Agente de Armazenamento, Busca e Recuperação de Informações do domínio

5.1 Introdução

Conforme ressalta ARANGO (1988), em sua tese de doutorado, parte importante da sistematização do processo de reutilização é o suporte ao armazenamento dos componentes reutilizáveis especificados através de um processo de engenharia de domínio, e a busca e recuperação destes e de outros componentes externos que possam vir a ser agregados à infra-estrutura de reutilização. SAMETINGER (1997) destaca que existem diversos tipos de repositórios para o armazenamento de componentes, sendo que repositórios relacionados a domínios específicos são mais efetivos. Discutindo um pouco mais o tema, Sametinger ressalta ainda a importância da utilização do que ele chama de "repositórios de referência", que seriam como mecanismos de acesso aos diversos repositórios relacionados a domínios específicos.

Analisando a necessidade de armazenamento de informações do domínio na infra-estrutura Odyssey, poderíamos pensar, em um primeiro momento, em implementar serviços de armazenamento e recuperação, partindo do zero, como um elemento a mais da própria infra-estrutura. No entanto, a implementação destes serviços básicos implicaria no desenvolvimento de um componente de gerência tão complexo quanto o próprio desenvolvimento da infra-estrutura. Analisando o estado da arte atual dos SGBDs, consideramos adequada a utilização de orientação a objetos em SGBDs (SGBDO), como os SGBDs orientados a objetos e os SGBDs Relacionais-Objeto para o suporte à gerência de dados da Odyssey, pois estes aliam um modelo de representação poderoso, utilizando o paradigma orientado a objetos, com a disponibilização de mecanismos de acesso e gerência de dados adequados a este tipo de modelo.

No entanto, podemos encontrar também informações de um dado domínio fora do contexto da infra-estrutura e que sejam passíveis de reutilização, ou seja, podemos ter informações do domínio legadas, que podem estar armazenadas em uma grande variedade de meios de armazenamento, utilizando os mais variados modelos de dados, mecanismos de acesso e plataformas operacionais. Além disso, muitas vezes estas informações estão distantes geograficamente, dificultando ainda mais a sua manipulação. Ou seja, as informações do domínio existentes podem estar organizadas de maneira heterogênea e distribuída.

Levando em consideração estas necessidades de armazenamento e acesso às informações por parte da infra-estrutura Odyssey, consideramos a utilização de uma camada que permitisse a integração de diferentes bases de informações, sejam estas distribuídas e heterogêneas, de forma que as informações acerca do domínio fossem apresentadas ao usuário de maneira consistente e em um formato amigável.

Assim, a heterogeneidade e distribuição das informações é também um fator preponderante. Considere que seja necessário o acesso a componentes tanto de um sistema de arquivos quanto de uma base de dados relacional. Suponhamos ainda que estes repositórios de informação estejam distantes geograficamente um do outro e do engenheiro de domínio/aplicação que irá utilizá-los. A tarefa de busca por estas informações heterogêneas e distribuídas seria complexa, uma vez que cada formato de dados e cada tipo de base requer uma abordagem diferente, além da necessidade do tratamento da distribuição destas informações.

Por outro lado, atualmente, uma das áreas de desenvolvimento de sistemas que vem recebendo muita atenção é a de técnicas para busca e filtragem de informação. Com o advento da Internet, estas técnicas se aperfeiçoaram e, aliadas a técnicas de hipermídia (através do uso do WWW), permitem a busca de informações de forma facilitada para o usuário.

No contexto do desenvolvimento baseado em componentes, ocorre, similarmente, a mesma necessidade pela busca de informações precisas e somente quando o usuário delas necessita. As informações acessadas ao longo de um processo de engenharia de domínio voltado para o desenvolvimento de componentes podem estar armazenadas em repositórios de dados dos mais variados tipos. Assim, a busca por estas informações nem sempre é trivial de ser realizada, dado o volume e diversidade das mesmas. Além disso, a forma de busca e disponibilização destas informações nem sempre é a mais adequada, do ponto de vista do usuário.

Podemos concluir, portanto, que a busca por informações do domínio possui problemas similares, em um escopo mais limitado, à busca de informações precisas na Internet. Poderíamos então considerar a utilização de mecanismos de busca disponíveis na Internet. No entanto, atualmente, estes mecanismos não provêem a funcionalidade adequada e necessária para a busca de componentes. Considere, por exemplo, um *cenário típico de recuperação de componentes*, onde o engenheiro de aplicação procura por componentes para a aplicação que está desenvolvendo. Se não existe nenhum mecanismo específico para esta busca, o espaço natural a ser utilizado é a Internet.

Considere ainda que o engenheiro de aplicação não tenha conhecimento de que componentes estão disponíveis para esta busca. Desta forma, as seguintes ações seriam necessárias para a localização dos componentes que satisfaçam às necessidades da aplicação em desenvolvimento:

1. Localizar componentes que estão armazenados em repositórios distribuídos. Esta busca pode ser feita por um mecanismo de busca qualquer da Internet, tal como AltaVista, que requer algumas palavras-chave como entrada e retorna uma lista de *sites* que podem conter a informação desejada. O sucesso desta tarefa irá depender do interesse dos administradores de repositórios de componentes em publicar suas informações e da precisão do engenheiro de aplicação em prover as palavras-chave corretas.
2. Determinar a usabilidade dos resultados da busca. Por conta da complexidade da análise da real utilidade de um componente, como discutido nos capítulos anteriores, onde se deve considerar o domínio do componente, funcionalidade e possibilidades de interconexão, a informação disponível no *site* provavelmente não será suficiente.

Desta forma, um mecanismo de busca na Internet simples não é capaz de lidar com a complexidade da busca por componentes reutilizáveis. Necessita-se, portanto, de um mecanismo que combine as seguintes características: i) distribuição e heterogeneidade – os componentes podem estar distribuídos e utilizar diferentes mecanismos de armazenamento; ii) riqueza semântica para a descrição dos componentes, com o objetivo de facilitar sua busca; iii) possibilidade de se agregar novas informações a respeito dos componentes e também agregar novos componentes a serem considerados pelo mecanismo de busca.

Pesquisando a literatura sobre o assunto (MOUKAS, 1997), (MLADENIC, 1998), (MAES, 2000), (SEACORD, 2000), verificamos que nenhum mecanismo de busca atual da Internet é capaz de agregar todas estas funcionalidades, principalmente em relação a uma descrição mais rica semanticamente dos componentes. Por outro lado, pesquisando a literatura sobre tecnologia de banco de dados distribuídos e heterogêneos, pudemos notar uma preocupação grande com a semântica dos metadados, principalmente através do uso de ontologias (MENA, 1998) para a busca por informações heterogêneas e distribuídas. No entanto, estas propostas, ressentem-se de um mecanismo mais "ativo" para a apresentação de informações para os usuários, como é o caso dos mecanismos

mais modernos de busca na Internet, que se valem do perfil do usuário para antecipar informações para o mesmo.

Assim, baseados nas pesquisas atualmente em curso em relação ao armazenamento de componentes reutilizáveis, recuperação de informação na Internet e nas pesquisas relacionadas a busca e recuperação de informações distribuídas e heterogêneas em banco de dados, propomos um mecanismo de armazenamento e busca de informações do domínio no contexto da infra-estrutura Odyssey. Para isso, utilizamos conceitos como agentes (WOODRIDGE, 2000), mediadores (WIEDERHOLD, 1998), perfis (modelos) de usuário (BENAKI, 1997) e hipermídia adaptativa (STAFF, 1997) na especificação de um multi-agente inteligente, denominado Odyssey-Search, para o armazenamento, busca e recuperação de informações do domínio. A Figura 5.1 apresenta uma visão simplificada dos elementos que fazem parte da arquitetura deste sistema.

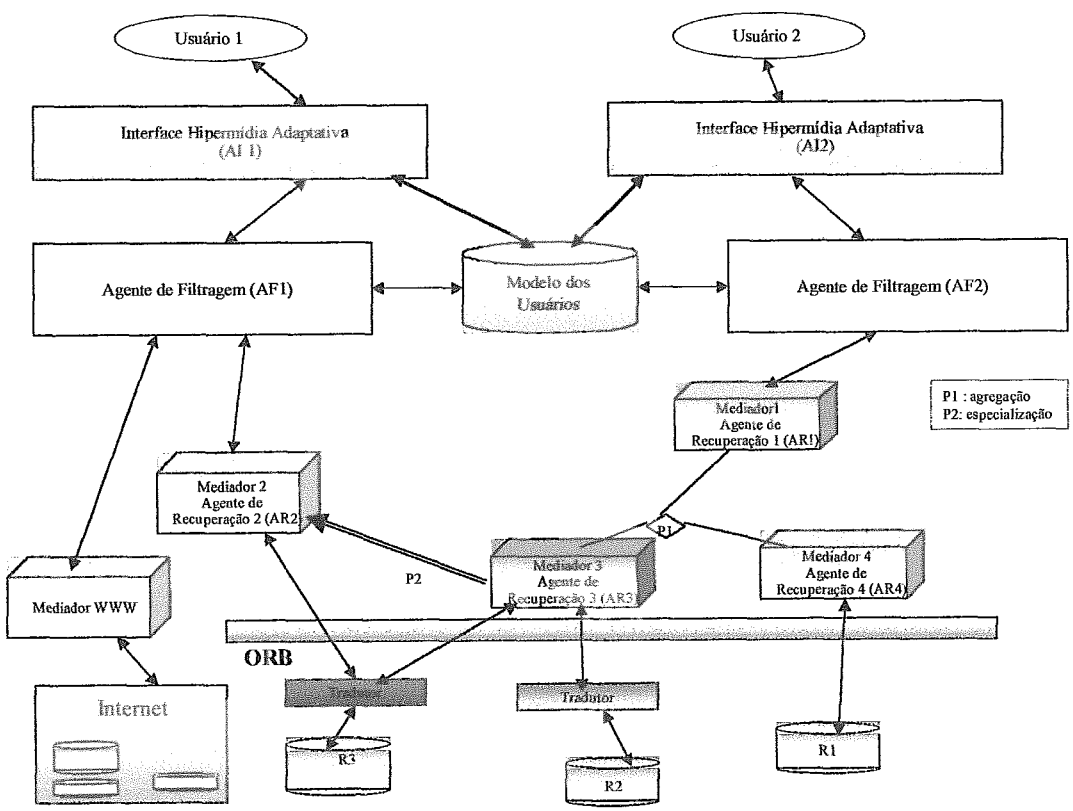

Figura 5.1 - Elementos básicos da Odyssey-Search

O Odyssey-Search é um sistema de múltiplos agentes, seguindo a classificação de WOODRIDGE (2000), que realizam: o armazenamento e a busca por informações de um dado domínio relevantes para a aplicação sendo especificada pelo engenheiro de software, e a busca por informações em domínios correlatos ao domínio em questão,

pelo engenheiro do domínio. Para isso, este sistema é dividido em três tipos de agentes: agente de interface (AI), agente de filtragem de informação (AF) e agente de armazenamento e recuperação de informação (AR). O AI é responsável pela adaptação da interface, de acordo com o perfil do usuário. O AF é responsável pela filtragem da informação, de acordo com as necessidade de informações por parte do usuário, e o AR é responsável pelo armazenamento e busca das informações nos repositórios de dados, lidando de forma transparente com questões como distribuição e heterogeneidade dos repositórios.

5.1.1 Características relacionadas à busca de componentes

YE e FISCHER (2000), através do estudo de aspectos cognitivos relacionados ao processo de reutilização, identificaram três modelos de reutilização: reutilização pela memória, onde o desenvolvedor lembra de um componente similar em outro sistema que ele já tenha desenvolvido e parte para reutilizá-lo; reutilização por recordação, que é quando o desenvolvedor se recorda vagamente de já ter visto um componente com aquela funcionalidade, mas não sabe exatamente qual componente; e a reutilização por antecipação, que é quando o usuário sequer sabe da existência do componente, mas acessa o repositório em busca do componente, sem saber efetivamente se o mesmo existe.

O primeiro modelo é o mais utilizado e é também chamado de reutilização *ad-hoc*. O segundo modelo, apesar de ser menos utilizado, indica que em alguns casos os desenvolvedores recorrem ao repositório em busca do componente. No entanto, se tiverem dificuldade de encontrá-lo, outras tentativas serão difíceis de ocorrer. Assim, para este segundo caso, o mecanismo de busca deve ser eficiente o bastante de forma a motivar o desenvolvedor. No terceiro modelo, o desenvolvedor raramente acessa o repositório para verificar se existe um dado componente. Geralmente nestes casos, os desenvolvedores argumentam que a chance de sucesso é incerta e por este motivo preferem desenvolver o componente a partir do zero.

YE e FISCHER (2000) discutem que na verdade o terceiro modelo deveria ser melhor trabalhado, uma vez que o volume de componentes não conhecidos pelos desenvolvedores é bem maior do que o dos componentes conhecidos. Assim, para que a reutilização destes componentes seja efetiva, necessitamos de mecanismos que facilitem a busca por componentes, incentivando a busca daqueles desconhecidos e sua localização, mesmo que estes estejam localizados remotamente. Além disso, é

importante a compreensão destes componentes, de forma que se possa entender perfeitamente sua função e como modificá-los.

O sistema multi-agente Odyssey-Search aborda os três tipos de modelos de reutilização, com ênfase maior para o terceiro, pois tenta antecipar as necessidades do usuário em relação aos componentes do domínio e de domínios relacionados, levando em consideração as necessidades da aplicação em desenvolvimento. Neste sentido, podemos classificar o Odyssey-Search como um repositório ativo, conforme descrito por YE e FISCHER (2000).

5.2 *Funcionalidades e Arquitetura básica da Odyssey-Search*

Analisando a literatura especializada em agentes (JANCA, GILBERT, 1997), (FLEMING, COHEN, 1999), (MAES, 2000) e as necessidades do engenheiro do domínio e do engenheiro de software na especificação de aplicações, chegamos à conclusão que um agente para a busca pelas informações do domínio deve apresentar as seguintes características:

- agir sempre de acordo com as necessidades do usuário o qual está auxiliando;
- ser personalizado, adquirindo informações a respeito do interesse de seu usuário e adaptar suas ações de acordo com estes interesses;
- ser persistente, ou executar continuamente, ou ainda salvar seu estado, de forma que possa guardar suas últimas ações para melhorar as próximas.

Para atender a estas características, a arquitetura da Odyssey-Search foi dividida em quatro módulos básicos:

- Módulo de modelagem do usuário, que especifica e disponibiliza o perfil do usuário para os agentes de interface e de filtragem;
- Agente de Interface (AI), que é baseado nos preceitos da hipermídia adaptativa, descrita no capítulo 3, onde a adaptação é realizada através de modificações na interface do usuário;
- Agente de Filtragem (AF) de informação, que, baseado nas necessidades iniciais do usuário, no modelo do usuário e na ontologia do domínio, filtra para o agente de interface apenas as informações mais adequadas para o usuário, decidindo também a respeito de como a informação será apresentada;
- Agente de Armazenamento e Recuperação (AR) de informação, que, possui duas funções específicas: armazenar componentes do domínio em repositórios

adequados, de acordo com o domínio ao qual o componente está relacionado e permitir a recuperação destes componentes de maneira adequada, utilizando para isso, o modelo de abstrações do domínio.

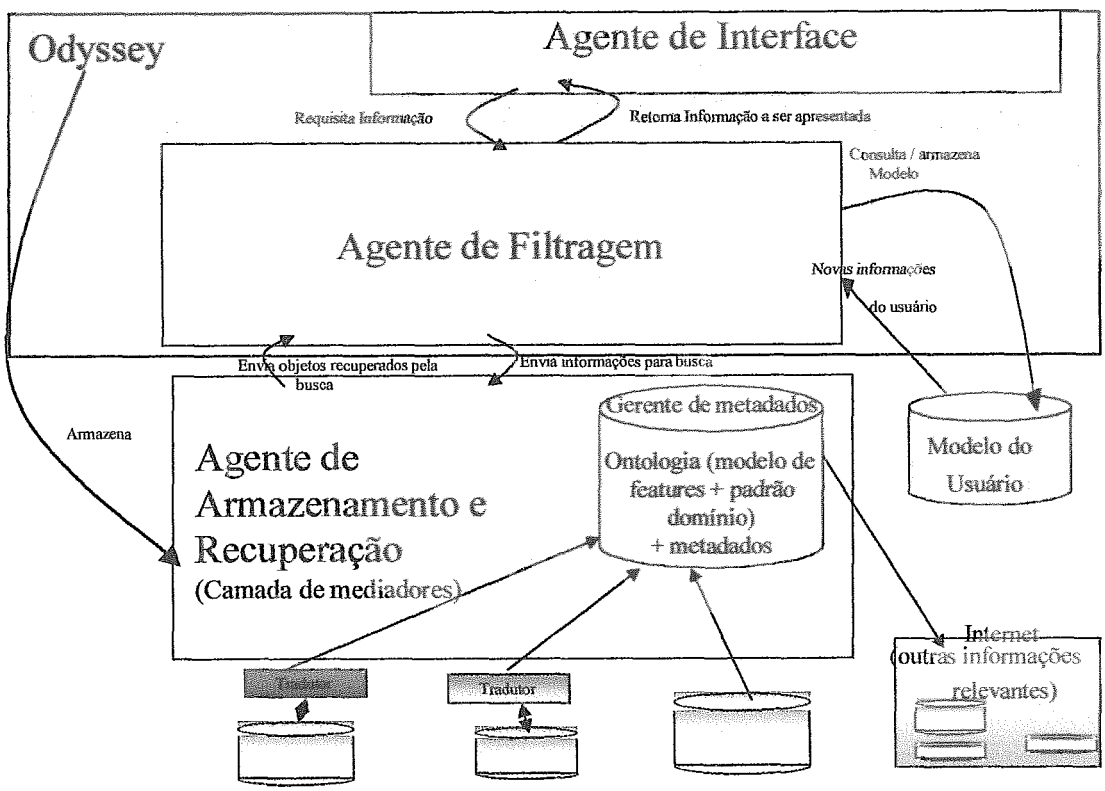

Figura 5.2 - Troca de informações entre os módulos da Odyssey-Search

A Figura 5.2 apresenta a troca de informações entre os módulos. As informações do domínio a serem disponibilizadas para os usuários (engenheiros de domínio/software) são apresentadas segundo a organização das informações na infra-estrutura Odyssey, descritos nos capítulos 4 e 6. Em relação aos diagramas, sejam estes diagramas de classes, características, interação, etc., são apresentados segundo a interface diagramática da Odyssey, que também possui características de hiperligações, podendo-se clicar em qualquer elemento para que informações adicionais sejam apresentadas, ou para que se navegue por outros níveis de informação (Figura 5.3).

5.2.1 Modelo do usuário

Para que as funcionalidades de adaptação e filtragem de informação possam ser especificadas, um item importante é a especificação de um modelo do usuário, onde informações a respeito de suas preferências são estabelecidas. Em um sistema de

adaptação e filtragem de informação do domínio, as técnicas de modelagem do usuário contribuem nos seguintes pontos:

- Criação de categorias de domínio: BENAKI (1997) ressalta a importância da identificação de categorias ou subsistemas do domínio. Esta categorização facilita a busca por informações do domínio. Por exemplo, um engenheiro de software que fosse desenvolver aplicações para controle do orçamento estaria interessado em informações como "verbas disponíveis", "proposta do orçamento", "ementas ao orçamento", etc., sendo estes pontos de partida adequados para se iniciar uma busca;
- Estereótipos: Estereótipos são grupos de usuários que compartilham os mesmos interesses de acordo com um conjunto de critérios. Na Odyssey-Search, estes estereótipos são conjuntos de funcionalidades as quais um dado grupo de usuários possui interesse;
- Perfil do usuário: o perfil do usuário deve guardar informações sobre o usuário em si, como nome, mas também informações a respeito dos estereótipos aos quais ele se encaixa, categorias de domínio, e outras informações sobre suas buscas passadas, atribuindo-se pesos às mesmas, de acordo com o número de visitas do usuário à informação, entre outros, denotando assim um maior ou menor interesse do usuário em relação a um dado item;
- Anotações a respeito do comportamento do usuário: Capacidade de aprender os interesses do usuário observando seu comportamento. Enquanto o usuário navega pelas informações relacionadas, o agente de filtragem faz anotações a respeito dos itens que despertaram maior interesse do usuário. O sistema pode modificar o perfil do usuário se ele identificar que as ligações visitadas pelo usuário não correspondem às preferências listadas em seu perfil. Este mecanismo, para produzir o resultado desejado, é acompanhado de uma interface que permite que o próprio usuário modifique parte do seu perfil a qualquer momento. É importante também que o sistema seja capaz de mudar o modelo do usuário, ou seja, mudar seu perfil, se as informações que ele requisitar e se interessar não corresponderem mais ao seu perfil. Novamente, esta mudança de perfil automática deve ser notificada ao usuário.

5.2.2 Agente de Interface

Analisando-se alguns trabalhos na área de Sistemas Baseados em Conhecimento (ANGELE *et al.*, 1996), (EUZEMAT, 1996), podemos notar que uma das formas mais utilizadas de interação do usuário com o conhecimento do domínio utilizado é a de

interfaces hipermídia. No entanto, somente a utilização da hipermídia não garante que o usuário irá realmente assimilar os conceitos do domínio e nem que a navegação que ele realizará pela hipermídia o levará a conceitos que de fato o interessam. Apesar das pesquisas em hipermídia apontarem para o uso de técnicas de *autoria* e do conceito de caminho para facilitar a navegação em um sistema hipermídia, esta ajuda não atende totalmente às necessidades de disponibilização de informações do domínio aos usuários, por serem estas técnicas estáticas, ou seja, não levam em consideração o caráter dinâmico da evolução do conhecimento armazenado e nem o conhecimento assimilado pelo usuário durante a navegação. Por isso, além da utilização de uma interface hipermídia, optou-se por prover uma forma de apresentar, de maneira dinâmica, as informações do domínio ao usuário, durante a navegação.

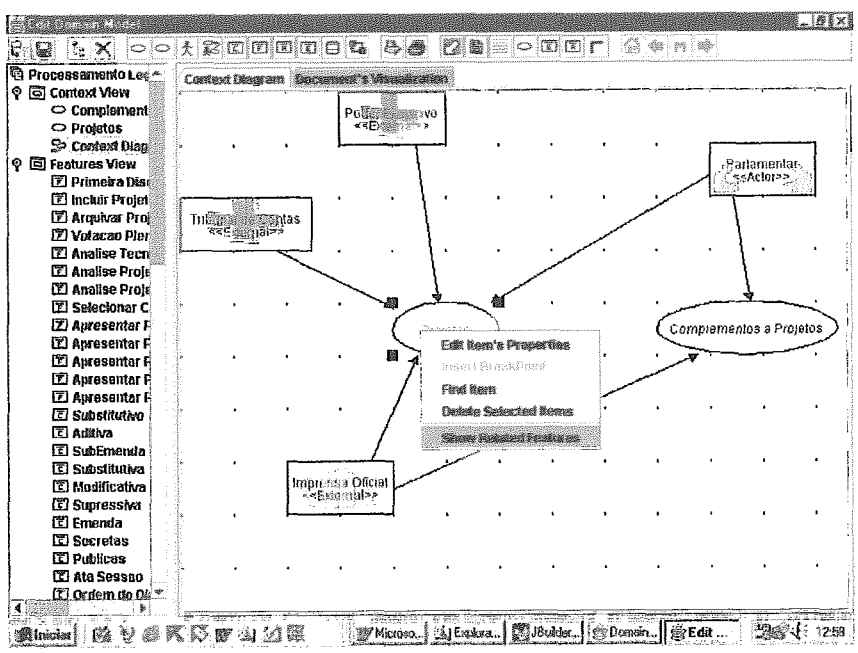

Figura 5.3 – Interface inicial para navegação com a Odyssey-Search

Desta forma, o agente de interface do Odyssey-Search, a partir das necessidades iniciais do usuário em relação à aplicação a ser desenvolvida e às escolhas e informações que ele disponibilize, é capaz de apresentar a informação ao usuário de acordo com suas necessidades. O agente de interface determina os requisitos do usuário e provê uma visão virtual das bases de informações do domínio. A forma de apresentação desta base de informações é alterada dinamicamente pelo agente, a fim de acomodar as necessidades e requisitos do usuário.

Sendo assim, a apresentação de todos os documentos do Odyssey é feita através da hipermídia adaptativa, onde a adaptação é acionada através de consultas ao perfil do usuário que está realizando a navegação. A adaptação é acionada principalmente em relação aos seguintes itens:

- o nível de aprendizado do usuário em relação aos termos do domínio. Este item é relacionado ao nível de conhecimento do usuário sobre o domínio. Este nível de conhecimento pode mudar de acordo com o aprendizado obtido através da navegação do usuário, determinando se a interface deve ser modificada para apresentar mais detalhes de cada componente visitado;
- o nível de abstração que o usuário deseja consultar (conceitual, arquitetural ou implementacional), onde o nível de detalhamento dos conceitos do domínio pode ser alterado, determinando se este detalhamento deve ou não ser apresentado ao usuário durante a navegação.

Além disso, o perfil do usuário também pode ser alterado a partir da observação do seu comportamento em relação à interface adaptativa, da seguinte forma:

- Se o usuário está apenas interessado no entendimento do domínio e não no desenvolvimento de aplicações, detalhes a respeito de componentes arquiteturais e implementacionais não devem ser apresentados. No entanto, é importante frisar que o agente não proíbe a navegação para este tipo de componente, apenas alerta para a não necessidade de fazê-lo (Figura 5.4);
- Se um usuário navega por documentos onde aparece muito um dado termo do domínio, mas o usuário ainda não apresentou nenhum interesse específico por aquele termo, o agente de interface percebe e sugere que o usuário consulte o termo (Figura 5.5);
- Considerar se o usuário é ou não conhecedor do domínio. Se não for, a forma de apresentação das informações é simples, apresentando detalhamento para termos técnicos e itens que possam ser de difícil entendimento para usuários não especialistas (Figura 5.6). Assim, o modelo do usuário deve capturar o conhecimento de que o usuário não é um especialista, armazenar esta informação e modificá-la à medida que o usuário for se tornando um especialista. Esta mudança é capturada através das sucessivas navegações do usuário e do número de abstrações do domínio consultadas.

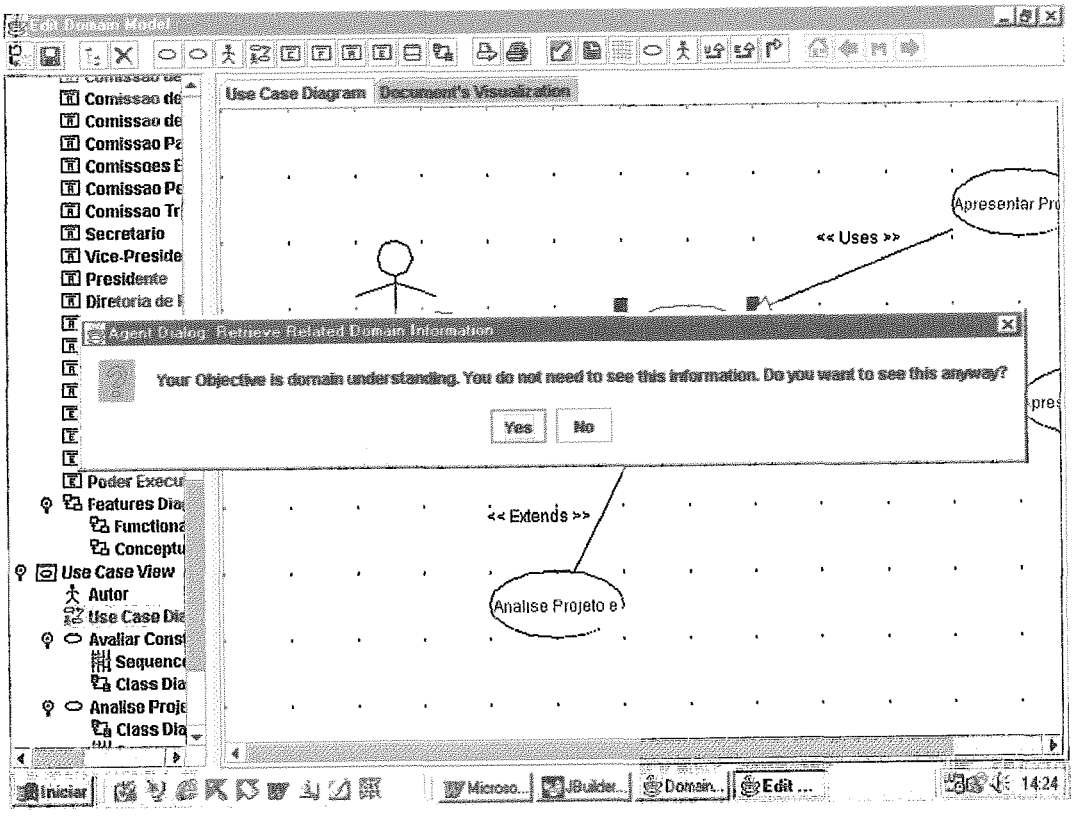

Figura 5.4 – Alerta para a não necessidade da informação

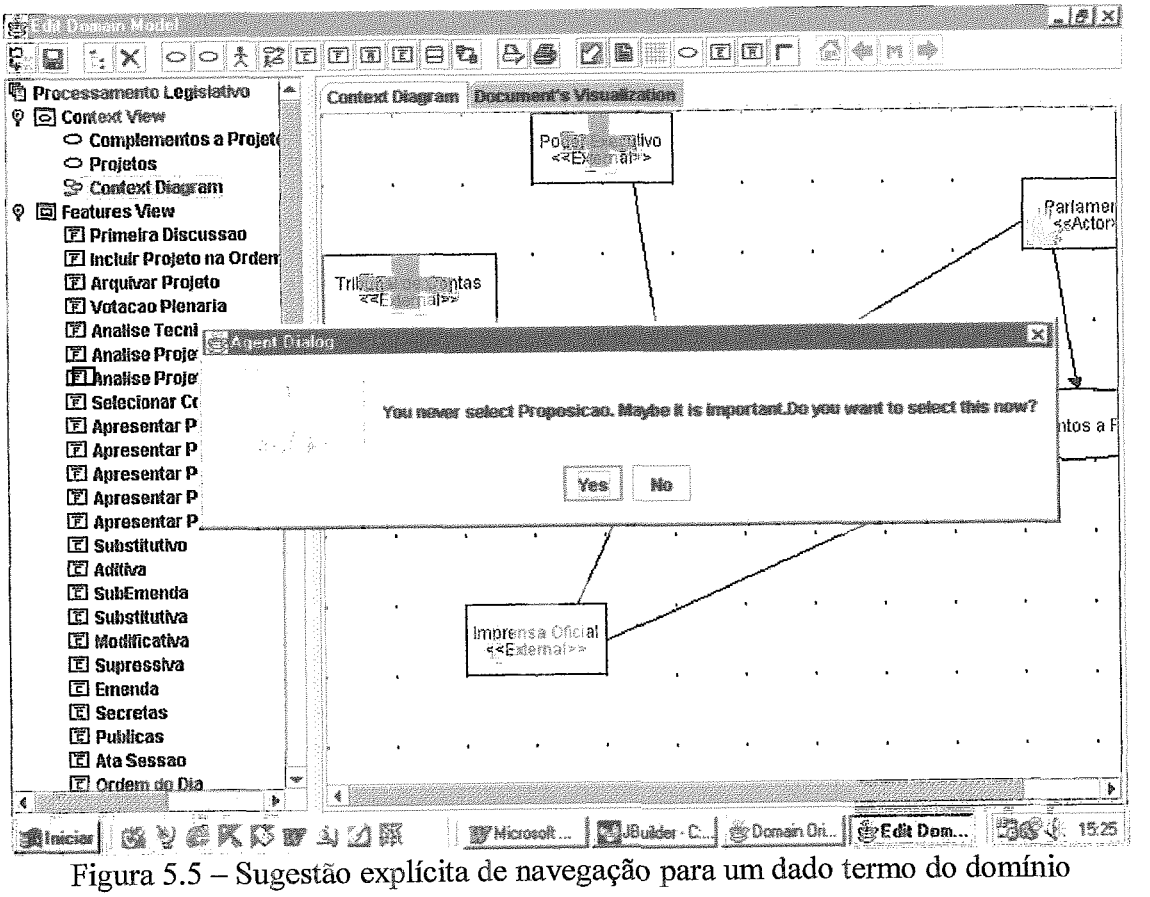

Figura 5.5 – Sugestão explícita de navegação para um dado termo do domínio

5.2.3 Agente de Filtragem de Informação

Levando-se em consideração as técnicas de filtragem de informação apresentadas no capítulo 3, no contexto da filtragem de informações do domínio, uma abordagem onde são mescladas funcionalidades de filtragem social e filtragem baseada em conteúdo é adequada. Em um primeiro momento, é utilizada a busca de informações baseadas no perfil do usuário, ou ainda, nas necessidades da aplicação a ser desenvolvida pelo engenheiro de aplicação (explicitadas na montagem de seu perfil) e, em um segundo momento, a busca baseada no conteúdo de uma fonte de informação e a partir daí derivar novas informações.

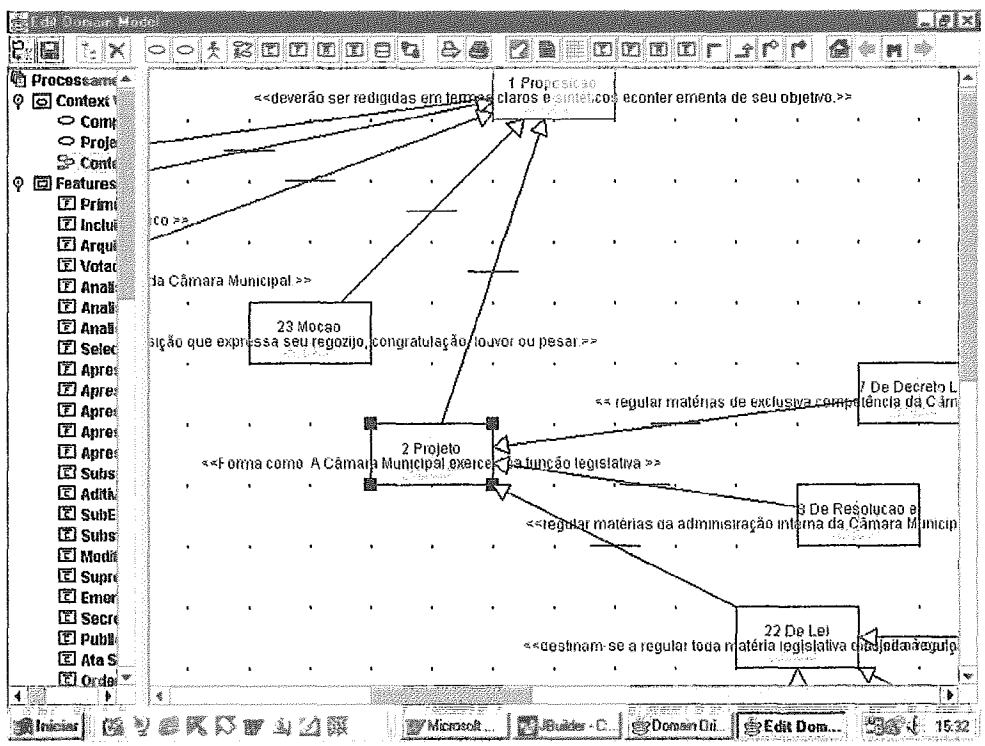

Figura 5.6 – Classificação das sugestões do agente de filtragem

Por isso, analisando as necessidades dos usuários do Odyssey, particularmente o engenheiro de aplicações, o agente de filtragem, em relação à modelagem do usuário, contempla as seguintes premissas:

- Na iteração do usuário com a ferramenta, captura quais são os tópicos de maior interesse para o usuário;
- A busca por informações do domínio leva em consideração que o usuário muitas vezes não sabe o que está procurando e as vezes utiliza termos não técnicos, ou não muito utilizados no domínio, para iniciar a busca. Assim, o agente faz com que esta

busca por informações do domínio seja bem flexível, utilizando um conjunto de sinônimos para os termos do domínio;

- O usuário é capaz de visualizar e monitorar o seu modelo, de modo que possa ganhar confiança em relação às sugestões dadas pelo agente;
- O agente captura os interesse do usuário observando seu comportamento, ou seja, quais são suas ações, que informações e caminhos de navegação são mais utilizados, a fim de sugerir novos caminhos de busca (Figura 5.6). Estas informações são sumarizadas de tempos em tempos com o objetivo de realizar uma limpeza neste histórico de navegações;
- Baseado no tipo de informação solicitada, pode-se consultar bases legadas do domínio na busca de informações a respeito de algum aspecto do domínio ou informações relativas ao desenvolvimento da aplicação;
- Se o usuário apresenta interesse por informações que possam estar relacionadas a outros domínios de aplicação, o agente de filtragem deve sugerir esta busca por informações em domínios relacionados (Figura 5.7). Este interesse por informações relacionadas a outros domínios é detectado através do tipo de abstração do domínio consultada. Se o usuário aceitar a sugestão, o agente de recuperação é acionado para recuperar as informações necessárias.

Desta forma, o perfil do usuário vai sendo modificado baseado no aprendizado do seu comportamento durante a navegação. Esta aprendizado é realizado através da observação das escolhas de caminhos de navegação e palavras-chave, i.e., a partir do número de ocorrências de palavras-chave na descrição dos itens do domínio e dos padrões de navegação seguidos pelo usuário ou por outros usuários com perfis similares.

Além do modelo do usuário, existem algumas funcionalidades que o agente de filtragem deve possuir para gerir de forma correta este modelo e atender às necessidades do usuário. Dentre os módulos que o agente de filtragem deve prover, é importante destacar:

- Módulo para visualização das premissas constantes do modelo: este módulo é responsável por manter a confiança do usuário em relação ao seu modelo, apresentando quais as palavras mais utilizadas em suas buscas, que tipo de

estereótipo são relacionados ao seu perfil e permitindo que o usuário monitore mudanças no seu perfil e as desfaça se assim desejar;

- Módulo de inferência de novas regras do domínio: O agente de filtragem deve contar com um conjunto de regras que defina como deve ser a filtragem de informações realizada pelo agente. Assim, devemos ter disponível um módulo responsável pela inferência, a partir dos documentos consultados pelo usuário, de novas regras que se adequam ao perfil do usuário e também pela disponibilização de regras para que o agente de filtragem possa enviar parâmetros de busca para o agente de recuperação.

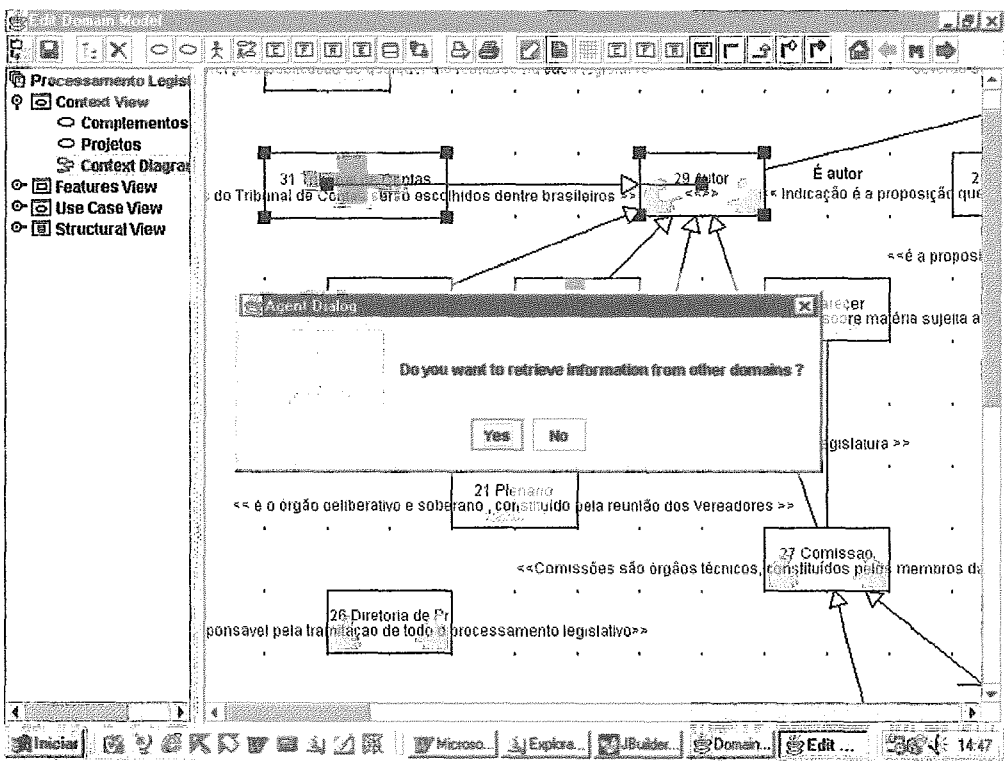

Figura 5.7 – Sugestão para busca por informações em outros domínios correlatos

5.2.4 Agente de Armazenamento e Recuperação de Informação

Pesquisas na área de desenvolvimento baseado em componentes argumentam que para que o processo de reutilização de componentes seja realmente efetivo, estes componentes devem estar ligados a um domínio de aplicação específico (JACOBSON *et al.*, 1997). Com isso, as possibilidades de reutilização aumentam substancialmente, uma vez que o componente será mais ligado a conceitos do domínio de aplicação, além da base de componentes a serem analisados ser bem menor, o que facilita a busca por

componentes específicos. Além disso, novos domínios de aplicação devem poder ser agregados à infra-estrutura. Sendo assim, a infra-estrutura Odyssey deve oferecer mecanismos que garantam a expansão da mesma, ou seja, mecanismos que permitam que informações acerca de novos domínios sejam agregadas, de forma facilitada.

Nossa proposta para o armazenamento dos componentes do domínio especificados através da infra-estrutura Odyssey foi a criação de uma camada de mediação inteligente, baseada no uso de ontologias do domínio, sendo que estas ontologias devem ser detalhadas por especialistas do domínio. O principal objetivo desta camada é permitir o armazenamento dos componentes reutilizáveis, divididos por domínios de aplicação, i. e., cada domínio possui uma ou mais bases de componentes, distribuídas ou não, que são acessadas através de um modelo integrador do domínio, ou seja, o modelo de características. Desta forma, a busca pelos componentes é facilitada, uma vez que a definição da ontologia está diretamente ligada a conceitos específicos do domínio. Com isso, os repositórios de componentes de suporte à infra-estrutura Odyssey, seguem a proposta de repositórios de componentes descrita em (SAMETINGER, 1997) e ressaltadas na introdução deste capítulo, ou seja, repositórios particionados por domínio.

Por outro lado, podem também existir aplicações que necessitem utilizar informações de mais de um domínio de aplicação. Neste caso, a estrutura de armazenamento e recuperação deve prover mecanismos que permitam a busca por componentes de diferentes domínios de aplicação. Outro ponto importante para que a reutilização dos componentes seja realmente efetiva, é que o engenheiro de domínio/aplicação que irá utilizar as informações do domínio, tenha as informações do domínio disponíveis em um formato que seja de fácil entendimento e que permita sua combinação de forma a possibilitar a especificação de aplicações no domínio.

Alguns fatores que podem afetar o acesso às informações do domínio são:

- as informações estão cada vez mais distribuídas (WIEDERHOLD, 1997) e utilizando os mais diversos meios de armazenamento como sistemas de arquivos, SGBDRs, SGBDOOs, textos em formato HTML, entre outros;
- é notório e reconhecido por pesquisadores consagrados na área que metadados, ou metadados semanticamente mais ricos como ontologias, que domínios específicos são melhores do que a especificação de metadados genéricos (WIEDERHOLD. 1995), uma vez que os termos do domínio são melhor definidos, sem ambigüidades e más interpretações;

- um sistema de armazenamento e recuperação de informações do domínio deve ser genérico o bastante para permitir que novos domínios de aplicação possam ser agregados ao sistema, ou seja, o sistema de recuperação deve oferecer mecanismos que permitam que informações sobre novos domínios de aplicação sejam disponibilizadas, atendendo ao formato padrão estabelecido pelo usuário destas informações. Sendo assim, o sistema de armazenamento e recuperação deve oferecer mecanismos que garantam a escalabilidade do sistema, ou seja, mecanismos que permitam que informações acerca de novos domínios sejam agregadas, de forma facilitada e outras informações concernentes ao desenvolvimento da aplicação possam ser consultadas, por exemplo, através da Internet.

Levando em consideração estas necessidades de armazenamento e acesso às informações por parte de um sistema de armazenamento e recuperação de informações do domínio e seus usuários, consideramos a utilização de um agente de armazenamento e recuperação. Este agente permite a integração de diferentes bases de informações, sejam estas distribuídas e heterogêneas, de forma que as informações acerca do domínio sejam apresentadas ao usuário de forma consistente e em um formato padronizado e de fácil entendimento. Este agente de armazenamento e recuperação é baseado na tecnologia de "mediadores inteligentes", conforme descrito no capítulo 3.

A escolha desta tecnologia se deve principalmente ao fato de que, no contexto da infra-estrutura Odyssey, de maneira geral, e da ferramenta Odyssey-Search, em particular, onde o acesso às informações do domínio é um requisito essencial, a utilização da tecnologia de mediadores vem permitir que este acesso seja realizado independente do formato da informação e da plataforma operacional onde está armazenada. Com isso, a estrutura de reutilização como um todo torna-se mais flexível, uma vez que as informações já existentes sobre um dado domínio, mesmo que especificadas utilizando outros formatos, podem ser agregadas de forma facilitada, sem a necessidade de conversão dos dados para um repositório de dados no padrão do Odyssey, já que a camada de mediação prevê a utilização de um tradutor para realizar a conversão dos dados para o formato adequado.

De maneira geral, um sistema de armazenamento e recuperação que seja baseado na estrutura de mediadores utiliza uma linguagem de consulta para a recuperação de informação. Em uma infra-estrutura de reutilização, como a infra-estrutura Odyssey, a especificação de consultas no formato SQL-*like* (*Structured Query Language* e derivados) não é a melhor forma para a busca por informação, pois muitas vezes o usuário de informações do domínio não sabe a priori que tipo de informação ele gostaria de consultar e nem como especificar uma consulta neste formato. Em uma linguagem de consulta convencional, para que se possa especificar consultas, é necessário se ter conhecimento do esquema da base de dados.

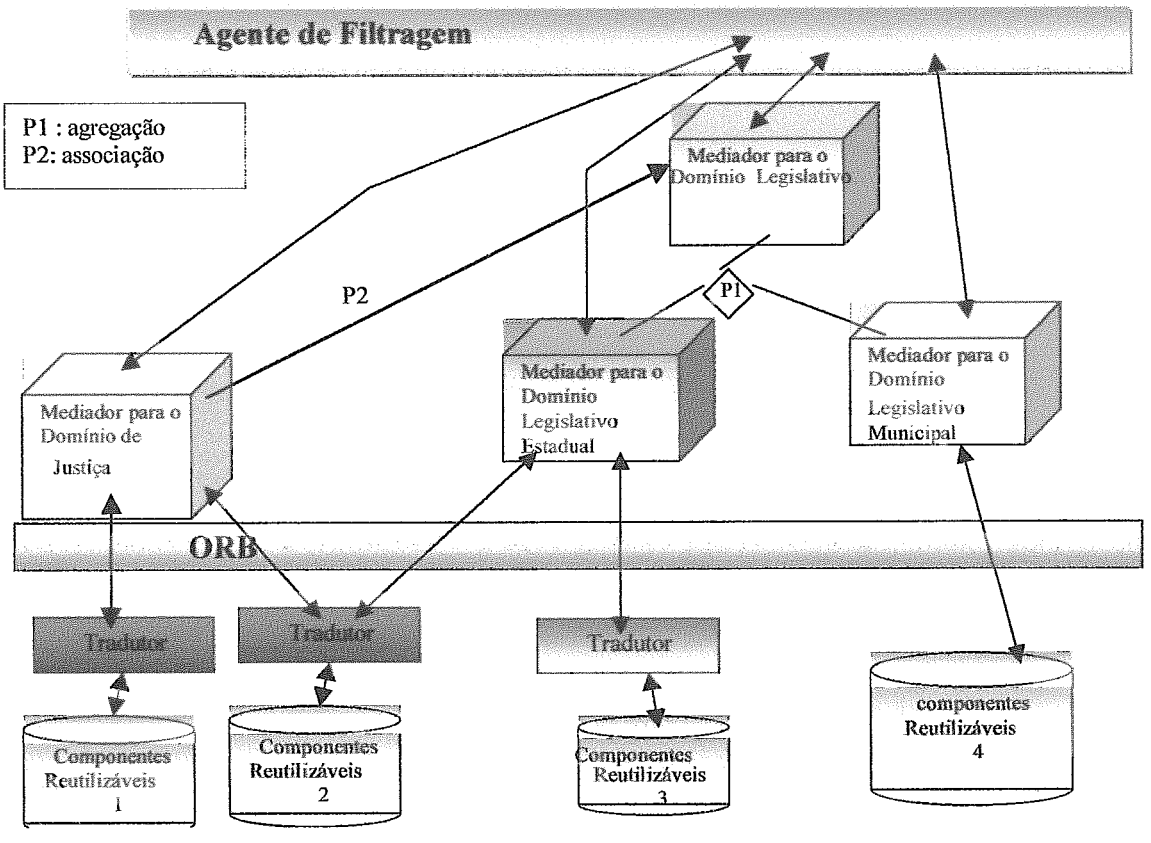

Figura 5.8 – Um exemplo de estrutura de mediação para o domínio legislativo

O agente de armazenamento e recuperação, para realizar consultas às bases de dados utiliza uma linguagem de consulta no formato SQL-*like*. No entanto, a chamada desta consulta é feita pelo agente de filtragem. O usuário não precisa saber a sintaxe do sistema de consulta, que no caso específico do agente de armazenamento e recuperação utiliza o módulo OQL do GOA++ (seção 5.3.5). O agente de filtragem é quem formula as consultas ao agente de armazenamento e recuperação. Por isso, é importante a

integração do agente de armazenamento e recuperação com o agente de filtragem para que as informações possam ser apresentadas ao usuário sem que este tenha que necessariamente conhecer linguagens de consulta, ou qualquer outro formato de recuperação de informações. A Odyssey-Search se encarrega de prover as informações de forma transparente em termos de localização e heterogeneidade, além de apresentar estas informações de maneira pró-ativa. Neste sentido, todos os três modelos de reutilização levantados por YE e FISCHER (2000) (seção 5.1.1) são contemplados pela Odyssey-Search. No entanto, é importante ressaltar que o agente de armazenamento e recuperação pode ser utilizado de maneira isolada da infra-estrutura Odyssey e da ferramenta Odyssey-Search, conforme descrevemos na seção 5.6.

A Figura 5.8 apresenta um exemplo de como seria esta estrutura de mediação para um domínio de aplicação específico, i.e., o domínio legislativo e seus subdomínios Legislativo Estadual e Legislativo Municipal. O domínio do Legislativo Estadual é agregado (p1) ao domínio do Legislativo Municipal, gerando um domínio mais genérico que é a combinação dos dois anteriores. Este domínio Legislativo geral pode ser usado em casos onde informações relativas aos dois domínios anteriores são necessárias. Cada mediador é conectado a uma ou mais fontes de dados que contêm componentes reutilizáveis relativos aos respectivos domínios. O mediador do domínio de Justiça pode ser acessado em casos onde o usuário necessitar de componentes relacionados a este domínio em particular.

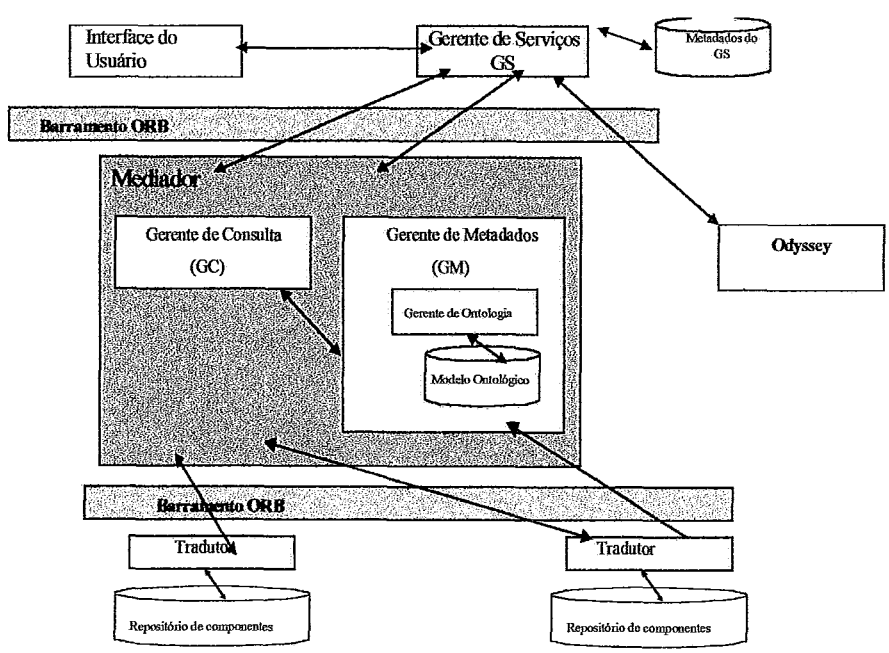

Figura 5.9 – Elementos de um mediador do Agente de Recuperação

A Figura 5.9 apresenta os elementos envolvidos na arquitetura do agente de recuperação, detalhando a estrutura interna de um mediador específico. Esta arquitetura é derivada da extensão da arquitetura HIMPAR (PIRES *et al.*, 1997), adicionando mais semântica e precisão através da utilização de ontologias especificadas para a recuperação de componentes reutilizáveis. Para que esta recuperação seja possível, no armazenamento dos componentes é também armazenado o relacionamento daquele componente com um dado termo ontológico do domínio. Estes relacionamentos, no contexto da infra-estrutura Odyssey como um todo, são os relacionamentos de "rastro", capturados através do Odyssey-DE. Como ressaltado no capítulo 4, cada termo ontológico é especificado como uma característica do domínio, que possui uma descrição detalhada, através do padrão do domínio relacionado. Cada característica é relacionada, através dos relacionamentos de "rastro", com componentes do domínio nos diversos níveis de abstração. Estes relacionamentos são armazenados, através da camada de mediação e, desta forma, é possível capturar os componentes dos respectivos domínios, levando em consideração informações semânticas dos mesmos, que são dadas pelos termos ontológicos.

Os principais componentes da arquitetura de mediação são:

- **Gerente de Serviços (GS):** Fornece serviços para a gerência e manutenção das diversas ontologias do domínio, conhecendo quais são os mediadores e tradutores disponíveis. Assim, o GS fornece um mapa dos recursos da camada de mediação, possuindo metadados a respeito da disponibilidade dos mediadores, tradutores e fontes de informações e também os metadados necessários para as ligações entre diferentes ontologias, que permitem o acesso a dados relacionados de múltiplos domínios. Para realizar os serviços de armazenamento e gerência dos metadados relativos a este componente, são utilizados alguns serviços do Sistema de Gerência de Objetos (SGO) GOA++ (MAURO *et al.*, 1997). O Esquema 5.1 apresenta uma visão geral dos metadados do Gerente de Serviços utilizando a notação ODL (*Object Definiton Language*) (ODMG, 2000). No entanto, a principal função do GS é o fornecimento de serviços para a integração das ontologias, quando for necessário, ou então redirecionar a busca de um dado domínio (utilizando um mediador) para outro domínio correlato (utilizando outro mediador). Esta integração de ontologias no contexto da Odyssey é realizada através do "casamento" das características constantes de cada modelo ontológico, através da identificação de sinônimos,

hiponônimos e hipernônimos. Através destes mecanismos, já bastante utilizados em outras abordagens para a integração de esquemas heterogêneos (MENA, 1998), podemos integrar ontologias de domínio similares e/ou direcionar as buscas em um dado domínio para domínios correlatos.

```
class Object_Himpar
( extent Himpares)
{
  attribute string Name;
}

class Mediator extends Object_Himpar
( extent Mediadores)
{
  relationship list<Container> AssociatedDataSources
   inverse DataSource::Medis;
  attribute string Description;
  attribute string KeyWords;
  relationship list<Mediador> Super
   inverse Mediador::*;
  relationship list<Mediador> Spec
   inverse Mediador::*;
  relationship list<Mediador> Assoc
   inverse Mediador::*;
  attribute string BaseName;
  relationship list<OntologyTerm> TermRel
   inverse OntologyTerm::MediadorRel;
  attribute String password;
}

class Wrapper extends Object_Himpar
( extent Wrappers)
{
  attribute string description;
  attribute string type;
  relationship list<DataSource> Repositories
   inverse DataSource::Trad;
}

class Component
( extent Components)
{
  attribute string type;
}
```

```
class DataSource extends Object_Himpar
( extent DataSources)
{
  attribute string owner;
  relationship list<Mapping> Structure
   inverse Mapping::Cont;
  attribute string AbstractionLevel;

 relationship Wrapper Trad
   inverse Wrapper::Repositories;
  relationship list<Mediator> Medis.
   inverse Mediator:: AssociatedDataSources;
  attribute String password;
}

class Mapping
{
  attribute string DataSourceName;
  attribute string map;
  relationship DataSource Cont
   inverse DataSource::Structure;
}

class OntologyTerm
( extent Terms)
{
  attribute string Name;
  relationship list<OntologyTerm> Synonym
   inverse OntologyTerm::*;
  relationship list< OntologyTerm > Hipernym
   inverse OntologyTerm::*;
  relationship list< OntologyTerm > Hiponym
   inverse OntologyTerm::*;
  relationship Mediator MediadorRel
   inverse Mediator: TermRel;
 relationship Mappingr Map
   inverse Mapping: *;
}
```

Esquema 5.1 – Metadados do Gerente de Serviços

- **Gerente de Metadados (GM)**: este componente é responsável pelo mapeamento entre o mediador e os repositórios de informação. Cada mediador tem um GM específico. Assim, para cada mediador, o GM armazena a descrição do modelo ontológico (modelo de características) e o mapeamento deste modelo ontológico para os componentes armazenados nos respectivos repositórios de dados, através do uso de tradutores. Este mapeamento é descrito na subseção 5.3.7. Este componente também utiliza os serviços do SGO GOA++ para o armazenamento e a gerência de seus metadados. É formado pelos seguintes subcomponentes:
  - Repositório do modelo ontológico: Conforme já ressaltado, o modelo ontológico é o modelo de características do domínio e os padrões relacionados, instanciados

com informações específicas do domínio, especificado de acordo com o padrão ODMG-2.0. É através deste modelo ontológico que o usuário realiza consultas às bases de dados. Todas as consultas às informações armazenadas a respeito daquele domínio de aplicação serão realizadas com base no modelo ontológico.

- Gerente de atualização dos modelos: Este módulo é responsável pelas atualizações a serem realizadas tanto no modelo ontológico quanto nos modelos de objetos das fontes. Qualquer alteração no esquema de dados das fontes deve ser comunicada pelos responsáveis pela fonte de dados para que o administrador da estrutura de mediação tome as devidas providências. Este processo é um processo manual uma vez que mudanças deste porte podem comprometer o processo de mediação como um todo se não forem bem monitoradas. Sendo assim, a participação de um engenheiro do domínio no processo permite a validação das mudanças.
- Gerente de consultas (GC): Componente responsável pelo processo de especificação de consultas às bases de dados que participam do processo de mediação. As consultas são especificadas de acordo com o modelo ontológico. O formato utilizado para as consultas é o formato OQL. Seus principais subcomponentes são:
  - Tradutor de consultas: módulo responsável por verificar e especificar corretamente as consultas do usuário em um formato adequado ao modelo ontológico, de acordo com os parâmetros de busca enviados pelo agente de filtragem.
  - Decompositor de consultas para as fontes de dados: É o módulo responsável por particionar as consultas de acordo com os modelos de objetos das diversas fontes de dados, ou seja, uma vez que a consulta é especificada segundo o modelo ontológico, o decompositor fará a verificação de quais fontes de dados irão participar da consulta e tratará de particionar adequadamente a consulta de acordo com os modelos de objetos das fontes de dados participantes.
  - Empacotador de consultas: uma vez que a consulta tenha sido particionada nos diversos modelos das fontes de dados, gerando diversas subconsultas e estas subconsultas sejam enviadas para os tradutores das fontes de dados, para que as consultas sejam realizadas na linguagem de consulta nata da fonte de dados, o resultado destas subconsultas também será enviado pelos

tradutores de forma particionada. Sendo assim, o empacotador de consultas tem como função principal "juntar" estas subconsultas, de acordo com o modelo ontológico, e disponibilizar o resultado para o usuário de forma unificada.

- **Tradutores**: os tradutores são os módulos responsáveis por fazer a ligação das fontes de dados com a estrutura de mediação. Estas fontes de dados podem ser qualquer estrutura de armazenamento, tais como: Bases de Conhecimento, SGBDRs, SGBDOOs, Sistemas de Arquivos, Documentos HTML, entre outros. Cada tradutor é responsável por permitir que a fonte de dados participe do processo de mediação. É de responsabilidade dos administradores dos repositórios de componentes prover o módulo de tradução para a estrutura do agente de armazenamento e recuperação. Alguns tradutores são mais sofisticados do que outros, pois algumas fontes de dados não possuem sequer uma estrutura para a especificação de consultas, sendo desta forma necessário que o tradutor crie esta estrutura de consulta sobre a fonte de dados (ex.: busca em documentos html). Outros tradutores já podem contar com uma estrutura mais organizada por parte da fonte de dados, como é o caso do SGO GOA++. Desta forma, a estrutura do tradutor é simplificada, servindo apenas de elo com a estrutura de mediação.

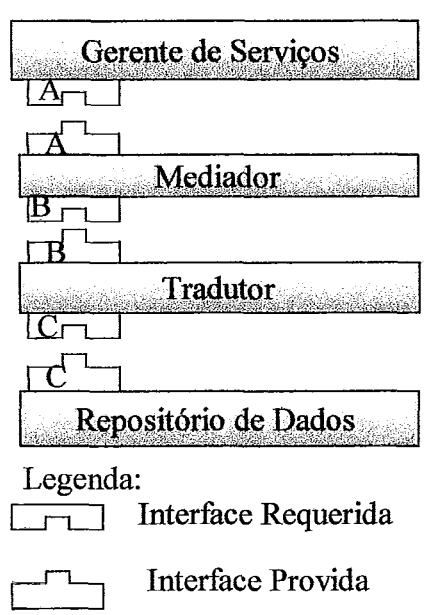

Figura 5.10 – Interfaceamento entre componentes do Agente de Recuperação

A estrutura utilizada para fazer a ligação do gerente de serviços, mediadores com os tradutores e respectivas fontes de dados, que estão distribuídas geograficamente, é a de

um ambiente ORB[1]. O Gerente de Serviços requisita informações para mediadores específicos. Cada mediador enviará as consultas aos tradutores empacotadas em uma interface IDL[2]. Os tradutores por sua vez serão também empacotados em interfaces IDL. Para que esta comunicação possa ser estabelecida, cada um dos componentes possui interfaces OMG IDL requeridas e providas. A Figura 5.10 apresenta esta configuração e os esquemas 5.2, 5.3 e 5.4 apresentam o detalhamento das respectivas interfaces em IDL.

```
Interface A
module Mediator
{

interface Access {
struct Feature {
                    string Nome;
                    string Descricao;
                    };
struct objeto {
                    string tipo;
                    string definicao;
                    };
struct OI {
                    long numpagina;
                    long indice;
                    };
struct obstruct { OI identificador;
            string tipo;
                    string fields[20];
                    string values[20]; };
typedef sequence<Feature> ListFeatures;
 typedef sequence<objeto> ListObjects;

// Funções para manipulação da base de dados
          string abre_base (in string nomebase);
          void  fecha_base (in string nomebase);
// Funções para manipulação da base de dados remotas
          string open_base (in string nomebase);
          string close_base (in string nomebase);
// Funções para manipulação de classes
          string get_class_description (in string nomeclasse);
     string get_schema( );

// Funções para a manipulação de objetos

                    objeto get_object (in string nome, in string
nomecolecao);
                    objeto get_object_oid (in OI ido);
                    obstruct get_objectstru_oid(in OI ido);
          obstruct    get_objectstru (in string nome, in string
nomecolecao);
          ListFeatures obtem_Names( in string nome_mediador, in
string nome_name);
          string getBindedObject(in string name);
          string getObjectStr(in string oid);
// Funções para manipulação da ontologia
                    ListFeatures    obtem_Ontologia    (in    string
nome_mediador);
// Funções para consulta a um mediador especifico
          ListObjects queryMediator(in string consulta);
          ListObjects queryMediatorbyBase(in string consulta); } };
```

Esquema 5.2 - Interface IDL de comunicação entre o GS e os GMs.

[1] ORB: *Object Request Broker*, barramento que realiza a comunicação entre os diversos objetos CORBA

[2] IDL: *Interface Definition Language*, linguagem de definição da interface dos objetos CORBA. Esta interface é que dirá que tipo de serviço o objeto poderá disponibilizar para outros objetos CORBA e como deve ser a sintaxe para a chamada dos serviços destes objetos. A IDL é uma linguagem puramente declarativa.

```
Interface B
module Tradutor{
interface Translator {
typedef long arg[512];
struct OI {          long numpagina;
                     long indice;          };
struct obstruct { OI identificador;
          string tipo;
                     string fields[20];
                     string values[20]; };
struct objeto {
                     string tipo;
                     string definicao;
                     };
struct Feature {
                     string Nome;
                     string Descricao;
                     };
struct base {
          string nome;
          string host;
          };
typedef sequence<OI> ListOIDs;
typedef sequence<base> ListBases;
typedef sequence<obstruct> ListObjectsStru;
typedef sequence<objeto> ListObjects;
typedef sequence<Feature> ListFeatures;
```

```
// Funções para manipulação da base de dados
                     string open_base (in string nomebase);
                     string close_base (in string nomebase);

// Funções para manipulação de classes
          string get_class_description (in string nomeclasse);
     string get_schema( );

// Funções para a manipulação de objetos
                     objeto get_object (in string nome, in string
nomecolecao);
                     objeto get_object_oid (in OI ido);
                     obstruct get_objectstru_oid(in OI ido);
          obstruct get_objectstru (in string nome, in string nomecolecao);
          ListObjects obtem_Names( in string nome_base, in string
nome_name);
          ListFeatures obtem_Ontologia (in string nome_base);
          string getBindedObject(in string name, in string nomebase);
          string getObjectStr(in string oid, in string nomebase);

//Funções para consulta a base de dados
                     ListOIDs query (in string consulta, in ListBases bases);
          ListObjects querybase(in string consulta, in base bases);
          ListObjects querybases(in string consulta, in ListBases bases);
          ListObjectsStru queryBasesStru(in string consulta, in ListBases
bases);                                   }; };
```

Esquema 5.3 - Interface IDL de comunicação entre um dado GM e os tradutores relacionados.

```
Interface C
interface Container {
struct OI {
                     long numpagina;
                     long indice;
                     };
struct obstruct { OI identificador;
                       string fields[20];
                       string values[20]; };
struct objeto {OI Identificador;
                     string tipo;
                     string definicao;
                     };
struct objetopuro{ string OID;
          string classe;
          string definicao;};
struct Feature {
                     string Nome;
                     string Descricao;
                     };
typedef sequence<OI> ListOIDs;
typedef sequence<objeto> ListObjects;
typedef sequence<objetopuro> ListObjetos;
typedef sequence<Feature> ListFeatures;
```

```
//criação de base de dados
                     string create_base (in string nomebase, in string schema);
                     string open_base (in string nomebase);
                     string close_base (in string nomebase);
          string getBindedObject(in string name);
          string getObjectStr(in string oid);
//criação de metadados
                     string create_class (in string nome_tipo, in string def_tipo);
                     string get_class_description (in string nomeclasse);
                     string get_schema( );
                     OI create_short_object(in objeto obj);
                     OI create_long_object (in objeto obj);
                     objeto get_object (in string nome, in string nomecolecao);
                     obstruct get_objectstru (in string nome, in string
nomecolecao);

                     obstruct get_objectstru_oid (in OI ido);
          objeto get_object_oid (in long numpagina, in long indice);
                     string update_object(in OI ido, in objeto obj);
          ListObjetos obtem_Names(in string nome_name);
                     ListOIDs query (in string consulta);
          ListFeatures obtem_Ontologia (in string nome_base);};
```

Esquema 5.4- Interface IDL de comunicação entre um tradutor e um container relacionado.

O agente de armazenamento e recuperação foi implementado em C++ Builder (INPRISE, 2000), utilizando os serviços de gerência de dados do SGO GOA++ (MAURO *et al.*, 1997). Além disso, utiliza como interface de comunicação entre as várias bases de dados componentes da estrutura, uma infra-estrutura CORBA, disponibilizada através da ferramenta Visibroker.

A comunicação entre o agente de armazenamento e recuperação e a infra-estrutura Odyssey é realizada através de uma conexão soquete, utilizando o pacote GOA, descrito na seção 4.5.4, para realizar a intermediação dos serviços.

5.3 Técnicas utilizadas na implementação da Odyssey-Search

Na especificação da arquitetura da Odyssey-Search, diversos tipos de decisões arquiteturais em relação à modelagem do usuário e uso de técnicas de busca, realizadas tanto pelo agente de filtragem quanto pelo agente de recuperação, foram tomadas, baseadas no estado da arte das técnicas atualmente utilizadas nestes contextos e os requisitos de funcionalidades e desempenho da busca, armazenamento e recuperação de informações do domínio.

Levando-se em consideração estas técnicas e a organização da informação do domínio no contexto do Odyssey, apresentamos a seguir como são estruturados os principais componentes do Odyssey-Search.

5.3.1 Modelo do Usuário

A Figura 5.11 apresenta o diagrama de classes do modelo do usuário, utilizando notação UML (FOWLER, 1997). Onde *edGrauFamiliaridade* descreve o grau de conhecimento sobre o domínio de um dado usuário. Para um usuário inexperiente, este *edGrauFamiliaridade* tem o valor de Novato, mas ao longo do tempo, de acordo com o número de navegações realizadas e o número de características consultadas, este valor pode mudar. O relacionamento da classe *UsuárioDom* com a classe *Estereótipo* descreve qual estereótipo um dado usuário tem mais afinidade. Com isso, quando o usuário estiver navegando pelos modelos do domínio, ele será direcionado para informações relacionadas à informação consultada, mas também considerando o contexto dos estereótipos.

A classe *ItemHistórico* armazena fatos capturados pelo agente de filtragem ao longo da navegação do usuário, ou seja, a medida que o usuário navega, o agente vai "anotando" fatos a respeito das escolhas/decisões feitas pelo usuário ao longo da navegação e, a partir destes fatos, pode sugerir novos caminhos de busca.

No contexto da Odyssey-Search, a modelagem do usuário envolve a criação de:

- Categorias do domínio, identificando sub-domínios (contextos na infra-estrutura Odyssey);
- Estereótipos, classificando usuários com interesses similares, incluindo suas preferências por categorias do domínio semelhantes;

- Perfil por usuário, onde informações como nome, estereótipo relacionado, e informações sobre suas buscas passadas, incluindo pesos para cada uma das buscas, são armazenados.

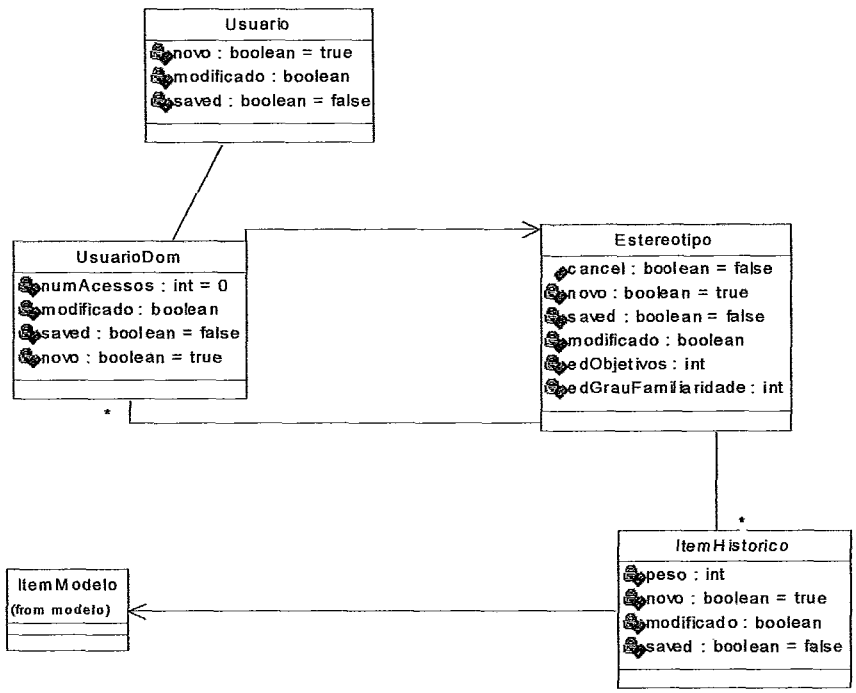

Figura 5.11– Diagrama de Classes do Modelo do Usuário

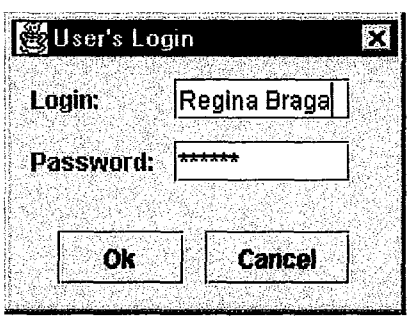

Figura 5.12 – Cadastramento inicial de usuário na Odyssey

Uma interação típica de um dado usuário com o módulo de modelagem do usuário, no contexto da Odyssey, é realizada da seguinte maneira: Primeiro, o usuário se registra como um usuário da infra-estrutura Odyssey, em geral (Figura 5.12), e da Odyssey-Search, em particular, através de um questionário (Figura 5.13). Este questionário da Odyssey-Search indica o nível de conhecimento em relação aos termos do domínio por parte do usuário e categorias do domínio que são de seu interesse. É importante ressaltar que a Odyssey-Search segue uma política de intervenção explícita mínima por parte do

usuário. Neste sentido, este questionário é o único retorno explícito que o usuário tem que prover para o sistema. Depois desta intervenção inicial, a Odyssey-Search, através de seus agentes, pode agir de maneira autônoma.

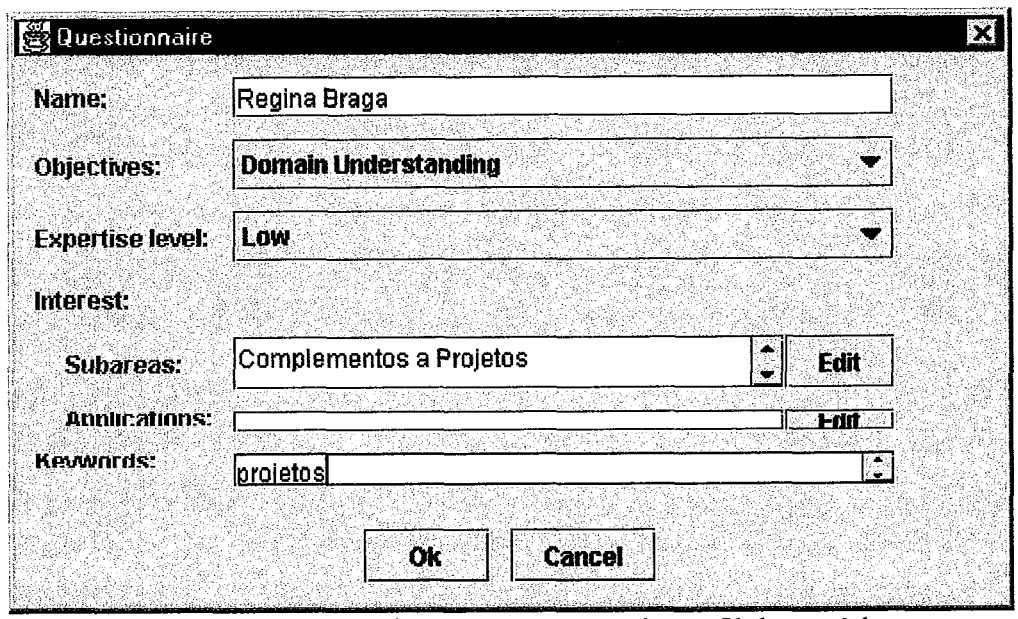

Figura 5.13 – Questionário para montagem do perfil do usuário

Os campos do questionário são:

a) **Objectives**: Indica o objetivo que o usuário deseja alcançar com a navegação através da Odyssey-Search. São as seguintes as possibilidades de escolha:

- Domain Understanding: O usuário objetiva apenas obter um conhecimento do domínio em termos de seus conceitos e funcionalidades, mas não objetiva desenvolver aplicações no domínio, ou seja, o usuário quer ser um especialista no domínio e por este motivo modelos de mais baixo nível não são de seu interesse.
- Application Development: O usuário deseja desenvolver aplicações no domínio e, por isso, um detalhamento em relação a modelos de classes, corpo de métodos, etc, são de seu interesse.

b) **Expertise level**: indica o nível de conhecimento do usuário em relação ao domínio. Existem três níveis possíveis:

- Low: O conhecimento do domínio é mínimo, i.e., a Odyssey-Search considera que o usuário nunca estudou o domínio. Assim, neste caso específico, é necessário que detalhes a respeito de cada termo/componente do domínio seja provido para o usuário.

- Medium: A Odyssey-Search considera que o usuário tem um conhecimento básico a respeito do domínio. O usuário, provavelmente, conhece o domínio de maneira superficial e no máximo desenvolveu uma aplicação simples neste domínio. Neste caso, a Odyssey-Search pergunta durante a navegação se o usuário deseja ter mais detalhes a respeito de modelos específicos do Odyssey.
- High: O usuário é um especialista no domínio e não existe a necessidade de se prover detalhes a respeito de itens do domínio.

c) **Sub-areas**: indica os contextos que são de interesse para o usuário.

d) **Applications**: aplicações já desenvolvidas no domínio que são de interesse ou são similares à aplicação que o usuário deseja desenvolver.

e) **Keywords**: Palavras-chave utilizadas pela Odyssey-Search que são utilizadas como base do algoritmo de aprendizado.

Depois que o usuário responde ao questionário, o agente tenta associar o novo usuário a um estereótipo já existente. Cada estereótipo armazena um perfil relacionado a um grupo de usuários. Um usuário deve sempre ser relacionado a um estereótipo. A Odyssey-Search considera que se um grupo de usuários tem o mesmo perfil, os interesses em relação ao domínio e, consequentemente, os caminhos de navegação podem ser os mesmos (MLADENIC, 1998), (MOUKAS, 1997). Desta forma, baseado no estereótipo e nas palavras-chave, a Odyssey-Search pode inferir caminhos de navegação que podem ser adequados para o usuário.

5.3.2 Base de Conhecimento

Um representação simples para o conhecimento usada na Inteligência Artificial (IA) são regras no formato *SE-ENTÃO*. Esta representação é bastante popular devido a sua simplicidade e facilidade de entendimento, podendo ser considerada como uma unidade de informação de uma base de conhecimento.

Devido a sua simplicidade de utilização e por serem as regras necessárias ao funcionamento adequado da Odyssey-Search de formato simples, optou-se por utilizar este formato de regras para a representação de fatos e um algoritmo de inferência para derivar o conhecimento necessário aos agentes de interface e filtragem. Para isto, a base de conhecimento da Odyssey-Search é composta de 3 elementos principais:

- Base de regras: a base de fatos relacionada ao processo de adaptação e mudança de perfil do usuário do Agente de Interface e de predição de novos links do Agente de Filtragem. Atualmente existem 66 regras especificadas.
- Memória de Trabalho: a informação capturada pelos agentes durante a navegação.
- Máquina de Inferência: para processar a base de fatos junto com a memória de trabalho, a máquina de inferência utiliza um algoritmo de *forward-chainning*[3] para gerar novos fatos na memória de trabalho.

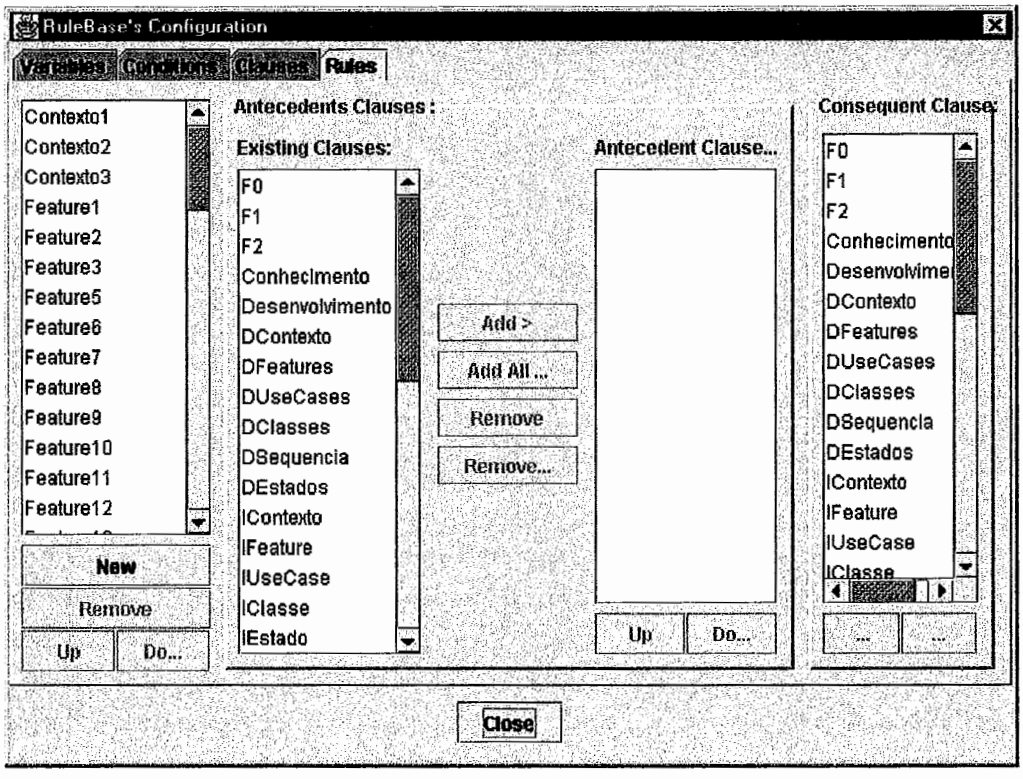

Figura 5.14 – Janela para cadastramento de regras da Odyssey-Search

A Figura 5.14 apresenta a interface para cadastramento das regras utilizadas pela Odyssey-Search. Atualmente, contamos com uma base de 64 regras, englobando regras para navegação do usuário, regras para mudança de seu perfil e regras para adaptação da interface do usuário. Cada uma das regras conta com várias cláusulas antecedentes, ligadas por relacionamento "E", e uma cláusula conseqüente. Considere o exemplo: REGRA1 (SE expertise = low and objective = domain_understanding ENTÃO developer = "Novice"), and REGRA2 (SE developer = "Novice" and Diagram =

Feature and Domain_Item = Feature ENTÃO Action = "Present Details"), onde a REGRA1 provê o fato "o desenvolvedor é inexperiente". Esta assertiva é utilizada para indicar a ação a ser tomada, i.e., apresentar detalhes sobre um certo item. Neste exemplo específico, esta inferência é utilizada para realizar a adaptação da interface.

5.3.3 Sugestão de caminhos de navegação

O Agente de Filtragem, para prever os melhores caminhos de navegação para o usuário, utiliza um algoritmo de predição de links. Para isso, o algoritmo realiza um batimento entre as palavras-chave especificadas pelo usuário, os contextos e aplicações escolhidas pelo mesmo com a descrição textual de cada item do domínio nos caminhos de navegação. O item do domínio que tiver o maior número de ocorrências relacionadas aos itens do perfil do usuário, é considerado como a melhor escolha nesta fase. No entanto, a ocorrência dos outros itens do domínio também são consideradas. Além disso, o algoritmo de predição também leva em consideração o histórico de navegações de usuários com perfis similares. Este histórico é uma base de itens já navegados por usuários com perfis similares e é associado ao estereótipo do usuário (ItemHistórico na Figura 5.11). Cada item possui um peso que indica o número de usuários relacionados ao estereótipo que já escolheram este item. Na Figura 5.15, é apresentado o algoritmo de predição em pseudocódigo.

```
Algoritmo de Predição de itens navegados (usado pelo agente de filtragem)

Vetor-de-itens-disponíveis = itens-disponíveis-para-navegação // i.e., (no
caso dos contextos seriam as características relacionadas)
    Para cada item ∈ Vetor-de-itens-disponíveis
      Busca uma entrada no histórico para o Item
      PesoNavegação=Achar o peso de acordo com a navegação do usuário
      Se item ∉ histórico
        PesoNavegação = 0;
      FimSe
      Adicionar peso ao Vetor de navegação(∑ Peso [PesoNavegação])
      Vetor-Palavras-Chave = ContagemPalavras (palavras-chave)
       PesoPalavra = contar o número de ocorrências da palavra no item.
       Adicionar peso ao Vetor_palavras (∑ peso [pesoPalavra])
       pesoFinal = peso(pesoPalavra)+ peso (pesoNavegação)
Adicionar pesoFinal to vetorPesos (∑ Peso[pesoFinal])
fimPara
    Fim.
```

Figura 5.15 – Algoritmo de Predição

[3] *Forward Chaining* é um processo de raciocínio onde um conjunto de regras é utilizado para derivar novos fatos de um conjunto inicial de dados (Bigus, 1998).

O resultado desta predição foi apresentado na Figura 5.6. Observe que na verdade é feita uma classificação dos itens do domínio de acordo com as preferências do usuário.

5.3.4 Retorno do usuário

As atitudes do usuário durante a navegação estabelecem a relevância de um item do domínio para aquele usuário. Este conceito é denominado no contexto de agentes inteligentes de retorno do usuário (*relevance feedback)* (WOODRIDGE, 2000). Este conceito auxilia o agente a identificar os itens mais importantes de um dado domínio, não só para um único usuário, mas para o grupo de usuários que compartilham um mesmo perfil.

Este aprendizado do agente é feito através de um algoritmo que captura as atitudes do usuário (*user feedback*), atualizando seu histórico de navegações. A maior vantagem do algoritmo de retorno do usuário utilizado pela Odyssey-Search é que esta captura das atitudes do usuário é realizada de maneira transparente. Quando o agente indica um determinado item do domínio para o usuário e este acata a sugestão, i.e., navega através deste item, o peso deste item no histórico de navegações relacionado ao estereótipo do usuário é acrescido de uma unidade. Da próxima vez que o algoritmo de predição de links for acionado, este item terá um peso maior na predição de itens mais importantes para o usuário.

Antes da execução do algoritmo de retorno do usuário, os seguintes passos devem ser executados:

1. O Agente de Filtragem realiza a predição de links (seção 5.4.3).
2. O Agente de Interface apresenta o grupo de itens do domínio sugeridos pelo agente de interface, levando em consideração as preferências do usuário em termos de apresentação dos itens (Figura 5.6, descricao de cada item do diagrama).
3. A partir destas sugestões, o usuário irá provavelmente continuar sua navegação através da escolha de um dos itens apresentados.
4. A partir da escolha de um dos itens de navegação apresentados, o módulo de retorno do usuário, através da execução do algoritmo mostrado na Figura 5.16, será acionado.

Conforme ressaltamos anteriormente, a captura do comportamento do usuário é realizada da maneira mais transparente possível. Desta forma, não é necessário que o agente interfira no processo de aprendizado durante a navegação. Basta que ele siga os caminhos de navegação que julgar mais adequados. Apenas com a seleção dos itens, o

agente aprimora seu aprendizado, sem a necessidade de um treinamento inicial e nem retreinamento quando a ferramenta for utilizada por outros usuários. A Figura 5.19 apresenta o pseudocódigo do algoritmo de retorno do usuário.

```
Algoritmo de Retorno do usuário

    Carrega a base de histórico de navegações a partir do Estereotipo do usuário.
    VetorItensIniciais = itens do domínio disponíveis para navegação
    Executa predição de links
    VetorItensSelecionadosPredição = itens selecionados pelo usuário para
                                     continuar a navegação dos que foram
                                     sugeridos pelo agente.
    Para cada item ∈ VetorItensSelecionadosPredição
        Busca por uma entrada do item no histórico de itens
        Se item ∈ histórico então
          pesoItem = pesoItem + 1;
        Senão
          Criar uma entrada para o item no histórico
          pesoItem = 1;
        fimSe
    fimPara
    salva o histórico de itens no Estereotipo
Fim.
```

Figura 5.16 – Algoritmo de Retorno do usuário

5.3.5 GOA ++, Gerente de Objetos Armazenados

Podemos observar pela descrição da arquitetura do agente de armazenamento e recuperação, que seus módulos principais, i.e., o GS e o GM, utilizam os serviços de um gerente de objetos para o armazenamento e consulta aos metadados armazenados. Este gerente de objetos é o SGO GOA++ (MAURO *et al.*, 1997).

O GOA++ possui uma arquitetura cliente/servidor de objetos. Em um servidor de objetos, a unidade de transferência entre o servidor e o cliente é o objeto. O servidor conhece o conceito de objeto e é capaz de executar consultas sobre os mesmos. Além disso, quando solicitado, o servidor fornece ao cliente uma cópia do objeto e registra o tipo de bloqueio desejado. Está fora do escopo desta tese uma descrição completa do SGO GOA++. No entanto, para o leitor interessado, maiores informações podem ser obtidas em (MAURO *et al.*, 1997).

O GOA++ possui diversos papéis no suporte à infra-estrutura Odyssey e no agente de armazenamento e recuperação em particular. Características específicas do GOA++ no apoio ao Odyssey podem ser encontradas em (MATTOSO *et al.*, 2000). A

principal função do GOA++ no agente de armazenamento e recuperação é ser utilizado como repositório dos metadados dos mediadores, tradutores e fontes de dados distribuídas com integração na arquitetura de mediação, atuando também como um repositório dos modelos ontológicos e da base de conhecimento utilizada. Além disso, a linguagem de consulta do GOA++ é utilizada para a consulta aos metadados armazenados. Todos os serviços providos pelo agente de armazenamento e recuperação são realizados com base em consultas OQL a bases GOA++. Assim, o GOA++ serve de gerente de dados tanto para o GS quanto para o GM. Outro ponto importante é que o GOA++ é também utilizado como repositório padrão dos componentes criados no processo de engenharia de domínio Odyssey-DE, através da infra-estrutura Odyssey.

Nesta última função, o GOA++ pode ser utilizado de maneira isolada para o armazenamento e recuperação de componentes relacionados a um único domínio no contexto da infra-estrutura Odyssey, ou utilizado, através do agente de armazenamento e recuperação, para ser o repositório de dados padrão da infra-estrutura Odyssey. No primeiro caso, o acesso a componentes armazenados em outros repositórios de dados e componentes relacionados a outros domínios não é possível. No entanto, no segundo caso, apesar do armazenamento também ser feito em um repositório GOA, os relacionamentos dos termos ontológicos do domínio, armazenados no GM e os componentes reutilizáveis armazenados no GOA é preservado, além de se ter acesso a outros repositórios do domínio. Além disso, como toda a estrutura está integrada ao GS, o acesso a informações de outros domínios de aplicação também é possível.

5.3.6 Armazenamento dos componentes reutilizáveis do domínio

Conforme ressaltamos nas seções anteriores, os componentes reutilizáveis, em todos os níveis de abstração, criados no contexto da infra-estrutura Odyssey, através do uso do processo Odyssey-DE, são armazenados, ou através do agente de armazenamento e recuperação ou diretamente, utilizando os serviços de um gerente de objetos. Para isso, a infra-estrutura Odyssey utiliza o pacote GOA, escrito na linguagem Java (JAVAWEB, 2000), detalhado nas seções seguintes deste capítulo, para realizar a comunicação com o agente de armazenamento e recuperação, que tratará de armazenar o componente reutilizável, de acordo com seu domínio de aplicação. Este armazenamento é realizado preservando-se a ligação de "rastro" entre o componente a ser armazenado e o termo ontológico, i.e., a característica do domínio. Desta forma, o armazenamento será feito a

partir do mediador específico do domínio do componente em um repositório de dados ligados a este.

No exemplo apresentado na Figura 5.8, um componente do domínio do legislativo municipal criado no contexto da infra-estrutura Odyssey será sempre armazenado, a partir do "Mediador para o Domínio Legislativo Municipal", no repositório "Componentes Reutilizáveis 4", sendo que este último é um repositório de dados GOA ++.

Para a realização deste armazenamento utilizando o agente de armazenamento e recuperação, é feita uma requisição, através do GS, para o armazenamento do componente na respectiva base GOA local. A chamada do serviço é realizada da seguinte forma:

```
GS->createGOAObject (char* MediatorName, char* OntologicalTerm, char* className, char* Structure);
```

Onde *MediatorName* é o nome do mediador relacionado ao domínio do componente a ser armazenado, *OntologicalTerm* é o nome ou os nomes do(s) termo(s) ontológico(s) mais diretamente relacionado (s) ao componente, *className* é o nome da classe relacionada ao componente, de acordo com os metadados do esquema de armazenamento dos dados da infra-estrutura Odyssey (usecase, objeto, estado, classe, etc.), e *Structure* são os dados do componente propriamente ditos.

Um exemplo de instanciação da chamada deste serviço seria:

```
GS->createGOAObject ("Mediador para o Domínio Legislativo Estadual", "Tramitação de Projetos", "NoUseCase", "{ Tramitação do Orçamento, Tramitação de projetos orçamentários.....} ");
```

No caso de não se utilizar os serviços do agente de armazenamento e recuperação, o armazenamento é realizado através da chamada dos serviços nativos do SGO GOA++, diretamente.

5.3.7 Mapeamento entre o mediador e os repositórios de componentes

Considere os dois repositórios de modelos do Odyssey e, portanto repositórios GOA++, na Figura 5.8 (*Componentes Reutilizáveis 2* e *Componentes Reutilizáveis 3*), contendo informações sobre Casos de uso relacionados à Tramitação de Projetos no Legislativo. Considere que seja necessário o acesso a estes repositórios através do mediador Legislativo Estadual. O repositório *Componentes Reutilizáveis 2* contém informações capturadas em um processo de ED realizada no Estado A, e o outro repositório, *Componentes Reutilizáveis 3*, informações capturadas no Estado B. O

repositório Componentes Reutilizáveis 2 contém o tipo *UseCase2*, cuja extensão é denominada *UseCases2*, e o repositório *Componentes Reutilizáveis 3* contém o tipo *UseCase3* cuja extensão é denominada *UseCases3* (estes dois tipos referem-se a casos de uso armazenados nestes repositórios, conforme definições de classe do Esquema 5.5:

```
Class UseCase2
(extent UseCases2)
{ attribute String nome
attribute String objetivo
attribute  String Descricao
attribute String Prioridade
}
```

```
Class UseCase3
(extent UseCases3)
{ attribute String Designação
attribute  String Descrição
attribute String Autor
}
```

Esquema 5.5 - Definições de classes para os tipos Documento1 e Documento2

Para a integração destas informações, os objetos de acesso relacionados aos repositórios de informações, aos tradutores relacionados e ao mediador devem ser criados. Apresentamos a seguir a sintaxe dos serviços de criação. Na seção 5.5, são apresentadas as interfaces visuais para criação destes elementos.

```
GS->mediatorCreation(char* name, char* description, char*
associatedContainers, char* generic, char* specific, char* associated);
GS->containerCreation(char * name, char* owner, char* associatedTranslator,
char* associatedMediator);
GS->translatorCreation(char* name, char* description, char*
associatedDataSources);

Instanciado para este exemplo específico:

GS->mediatorCreation("Mediador para o Dominio de Legislativo Estadual",
"mediador para acesso a dados relativos a legislativo estadual", "{Mediador
para o Dominio Legislativo }","{}", "{}");
GS->containerCreation("Componentes Reutilizáveis 2,", "Estado A",
"","Mediador para o Dominio de Legislativo Estadual");
GS->containerCreation("Componentes Reutilizáveis 3,", "Estado B",
"","Mediador para o Dominio de Legislativo Estadual");
GS->translatorCreation("TradutorGOA", "Tradutor para bases utilizando o SGO
GOA" , "Componentes Reutilizáveis 2, Componentes Reutilizáveis 3);
```

Uma vez criados os objetos de acesso aos repositórios de componentes, tradutores e mediador, também deve ser definido no mediador de Legislativo Estadual um termo ontológico que corresponda à funcionalidade de tramitação de projetos. A sintaxe do serviço que cria este termo ontológico no GS é:

```
GS-> createNewOntologyTerm (char* MediatorName, char* Structure);
```

Onde *MediatorName* é o nome do mediador relacionado e *Structure* é a estrutura do termo ontológico, de acordo com a classe OntologyTerm do Esquema 5.1. Instanciado para este caso específico:

```
GS-> createNewOntologyTerm ("Mediador para o Dominio de Legislativo
Estadual", "Name : Tramitação de Projetos; Synonym: { 13:2}; Hypernym:{};
Hiponym:{};");
```

É necessária também a criação de mapeamentos dos componentes nos repositórios para o termo ontológico. A sintaxe do serviço é:

```
GS->createMapping(char* MediatorName, char* OntologyTermName, char* Type,
char* ContainerName, char* map);
```

Onde *MediatorName* é o nome do mediador relacionado, *OntologyTermName* é o nome do termo ontológico relacionado, *Type*, designa o tipo de informação que pode ser encontrada no mapeamento, *ContainerName*, o nome do repositório de onde o componente está e *map* é o mapeamento necessário para se carregar o componente. Instanciado para o relacionamento com os repositórios *Componentes Reutilizáveis 1* e *Componentes Reutilizáveis 2.*

```
GS->createMapping(Mediador para o Domínio de Legislativo Estadual",
"Tramitação de Projetos", "UseCase", "Componentes Reutilizáveis 2", "Select x
from x in UseCases2 where x. nome = /" Projeto/"";);
GS->createMapping("Mediador para o Dominio de Legislativo Estadual",
"Tramitação de Projetos", "UseCase", "Componentes Reutilizáveis 3", "Select x
from x in UseCases3 where x. Designação = /" Projeto/"";);
```

Para acessar os dados de ambos os repositórios, utilizamos o seguinte serviço:

```
GS->queryMediatorByType(char*  Type,  char*  MediatorName,  char*
ontologicalTerm);
GS->queryMediatorByType("UseCase", "Mediador para o Dominio de Legislativo
Estadual", "Tramitação de Projetos");
```

Onde *type* corresponde ao tipo definido no mapeamento. Neste exemplo específico, o tipo *use case*, definido na criação das classes de mapeamento, *MediatorName*, que é o nome do mediador que se deseja acessar e *ontologicaTerm*, que é o termo ontológico de ligação com os componentes.

É importante ressaltar que, atualmente, a criação destes componentes e a formulação de consultas é realizada utilizando a interface visual dos componentes, como apresentado na seção 5.5.

5.3.8 Ontologia do Domínio

Uma característica importante da Odyssey-Search, em geral, e da camada de mediação, em particular, é o uso de ontologia do domínio para a busca por informações do domínio, no contexto de um dado domínio ou entre domínios correlatos.

Na Odyssey-Search, o componente que é responsável por realizar estas ligações ontológicas inter-domínios é o Gerente de Serviços (GS), da arquitetura apresentada na Figura 5.9. Este componente é responsável por qualquer tradução que seja necessária entre termos de diferentes domínios. Ele sempre tem informação atualizada a respeito do domínio. Esta informação é transmitida através do barramento CORBA (OMG, 2000) para os mediadores requisitados, sendo que o GS se encarrega de traduzir a requisição para as respectivas ontologias dos mediadores, de acordo com os relacionamentos inter-domínios capturados, usando desta forma os termos ontológicos corretos para cada domínio, representados pelos seus mediadores. Estes relacionamentos entre termos ontológicos de diferentes domínios envolvem ligações semânticas tais como relações de sinônimos, hipernônimos e hiponônimos. Um relacionamento de sinônimo associa termos ontológicos em diferentes domínios que são sinônimos entre si. Relacionamentos de hipernônimos e hiponônimos representam ligações entre termos ontológicos em vários domínios que são mais genéricos ou mais específicos que o termo ontológico considerado. Desta forma, com estes relacionamentos, é possível associar termos ontológicos de diferentes domínios, permitindo a busca integrada por informações reutilizáveis.

Para capturar o modelo ontológico de cada mediador, o GS usa o barramento CORBA (ORB), através do serviço `retrieve_Ontology()` da interface IDL de conexão com os mediadores (Esquema 5.2), para acessar cada um dos mediadores cadastrados, utilizando seus termos ontológicos. Com os termos ontológicos relativos a cada um dos mediadores, que representam os diferentes domínios disponíveis, o engenheiro do domínio realiza as associações necessárias entre termos de diferentes domínios, para que seja possível a posterior recuperação de componentes reutilizáveis baseados neste modelo integrador. Para que estas associações sejam possíveis de serem especificadas, o engenheiro do domínio atualmente necessita acessar a interface gráfica relacionada ao agente de recuperação, pois esta associação é feita manualmente. A Figura 5.20 apresenta a interface utilizada pelo engenheiro do domínio para a criação das ligações ontológicas inter-domínios. Neste exemplo específico, o engenheiro do

domínio está criando um relacionamento de sinônimo entre um termo ontológico no domínio Legislativo e outro termo ontológico no domínio de Justiça.

Feitas as associações entre termos de diferentes domínios, é possível a realização de consultas, via interface IDL através do serviço `query(in string consulta)`, que é a interface para conexão de outras ferramentas, inclusive a infra-estrutura Odyssey como um todo, ou através da interface gráfica do Agente de Armazenamento e Recuperação, disponibilizada através do componente Service Manager detalhado na subseção 5.5.1. Desta forma, o Service Manager acessa e recupera todos os mediadores relacionados, procurando por componentes que satisfaçam à semântica da consulta. Os componentes recuperados são transferidos então, pelo barramento ORB, para o Service Manager que os disponibiliza para as ferramentas adequadas.

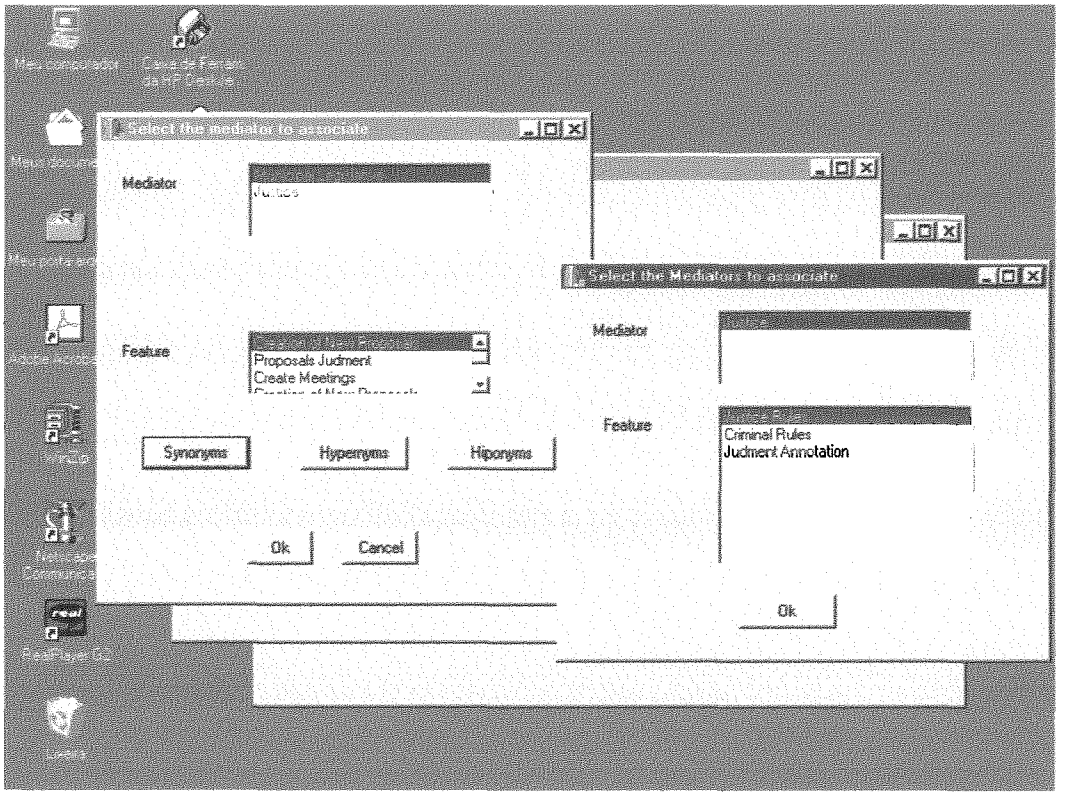

Figure 5.20 – Interface para associação entre múltiplas ontologias

5.4 *Odyssey-Search no contexto da infra-estrutura Odyssey*

Conforme ressaltamos no capítulo 4, todas as ferramentas implementadas no contexto da infra-estrutura Odyssey fazem parte do pacote ferramentas e seguem o padrão *Model-View-Controller*. A Odyssey-Search foi construída seguindo esta

padronização e é acoplada à infra-estrutura Odyssey através do pacote *agente*, que faz parte do pacote *ferramentas*.

Além do pacote *agente*, foi necessária a implementação do pacote *rule*, que é responsável pelo módulo da base de conhecimento do Odyssey, e o pacote GOA, que contém as classes que fazem o interfaceamento da infra-estrutura Odyssey como um todo com a camada de mediação (agente de armazenamento e recuperação).

A decisão de deixar o pacote *rule* fora do pacote *ferramentas* se deu, principalmente, por conta do pacote *rule* ser um pacote genérico para uso de uma base de conhecimento e neste caso outras funcionalidades do Odyssey podem futuramente utilizar este pacote de forma genérica e não somente as ferramentas do pacote ferramentas. A mesma decisão foi tomada em relação ao pacote GOA. Apresentamos a seguir uma breve descrição destes pacotes.

5.4.1 Goa

As classes básicas do pacote GOA estão representadas na Figura 5.21. A principal classe do pacote GOA é a classe *GoaServer*, que representa a conexão ativa com o SGO GOA++, ou seja, é a classe responsável pela interface de conexão com a camada de mediação ou com uma base GOA. Esta conexão é feita através da classe *GoaServer*, pois é ela que possui os serviços necessários à gerência de dados. Para tratar a conexão fisicamente, é utilizada a classe *GoaStub*, que na verdade implementa os serviços da conexão soquete através do recebimento e envio de *streams* de dados.

Para representar na infra-estrutura Odyssey os objetos vindos da camada de mediação ou diretamente do SGO GOA++, é utilizada a classe *GoaObject*, que possui atributos e serviços necessários ao tratamento de um objeto GOA. Para tratar as coleções de dados (names no GOA), é utilizada a classe *GoaCollection*, que representa uma coleção de objetos do GOA.

Todas estas classes são responsáveis pela gerência de dados para objetos armazenados na base GOA, implementando uma política de recuperação de dados por domínio, ou seja, cada domínio possui sua base GOA e a classe GoaServer é a responsável por tratar estas múltiplas conexões. Além disso, a conexão GOA adota a política de atualização da base, ou seja, a base é atualizada sempre que necessário. Não é adotada uma política onde, todas as vezes que é feita uma conexão, a base é limpa e é gerada uma nova base, como é o caso da serialização nativa de Java. Assim, a conexão

GOA tenta implementar todos os conceitos importantes em uma conexão de banco de dados.

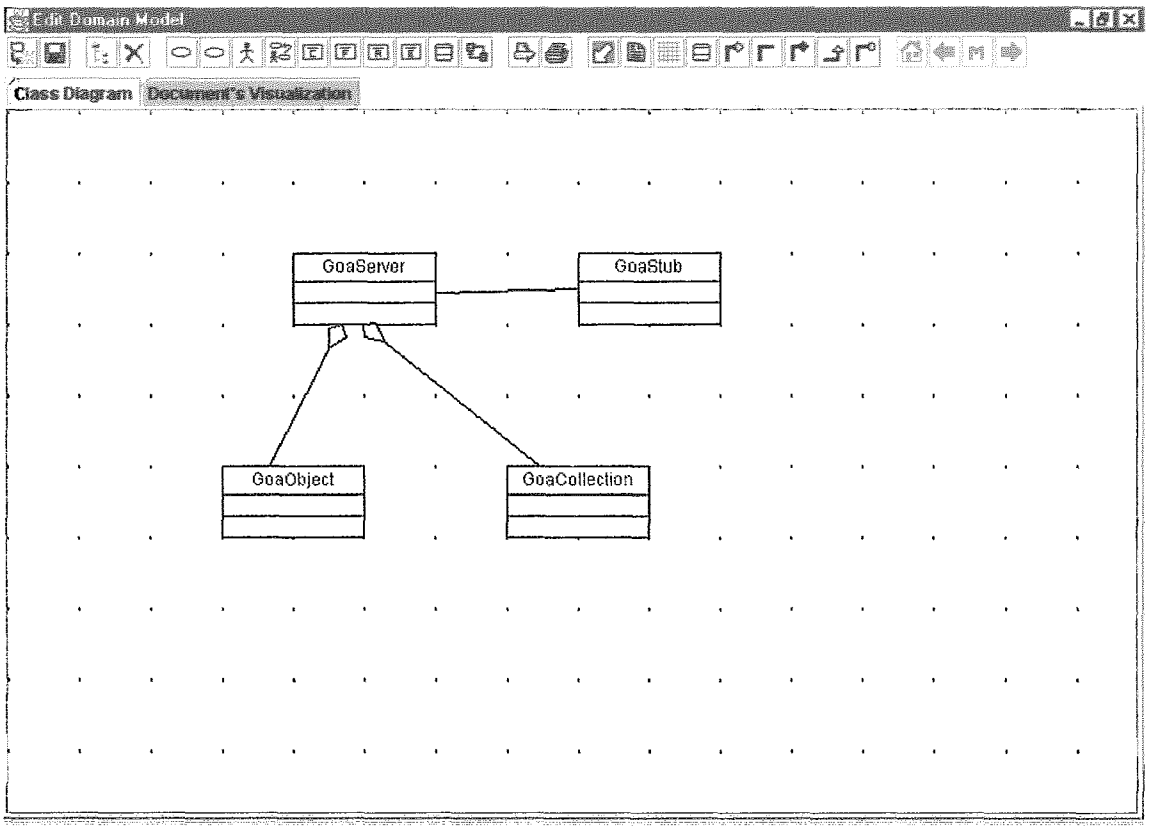

Figura 5.21– Principais classes do pacote GOA

5.4.2 Pacote Agente

O pacote *agente* é subdivido nos seguintes pacotes: *Model*, que apresenta as classes de domínio do pacote; *View*, que apresenta a forma de visualização das classes do pacote e o pacote *Controller*, que apresenta as classes controladoras do Agente. A Figura 5.22 apresenta o detalhamento do pacote *Controller* e os relacionamentos de suas classes com as classes dos outros pacotes e do pacote *Rule*.

A classe *GerenteAgente* é uma classe que segue o padrão de projeto *Singleton* (GAMMA et ali, 1994) e representa a classe de interfaceamento do Agente com os outros pacotes da Odyssey. Esta classe é responsável pela criação dos diversos agentes que atuarão nos domínios disponíveis na Odyssey. Cada domínio tem um agente específico para cada usuário. Quando um novo usuário é cadastrado na Odyssey, ele é criado como um objeto da classe *Usuário* e é associado ao *GerenteAgente* que se encarregará de criar, caso o usuário se interesse em visualizar um dado domínio, um perfil deste usuário para navegação neste domínio. Este perfil é armazenado na classe

UsuárioDom. Cada perfil de usuário é associado a um *Estereótipo*, que seria um perfil genérico de usuários com os mesmos interesses no contexto da Odyssey-Search.

Além disso, cada agente instanciado pode acessar as funcionalidades da base de conhecimento, através da classe *RuleInit* do pacote *rule*.

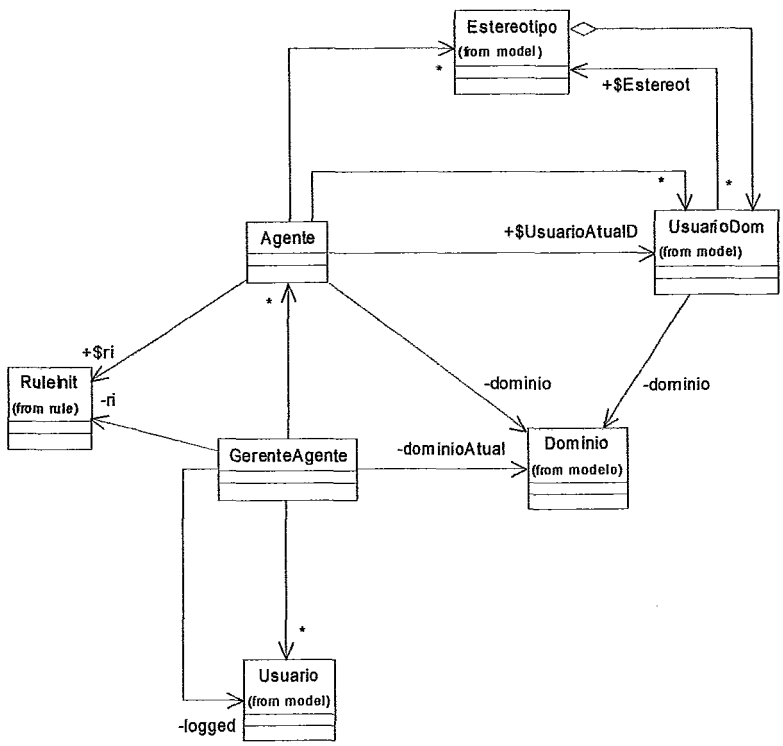

Figura 5.22– Principais classes do pacote Agente.Controller

5.4.3 Pacote Rule

O pacote *rule* é também subdivido nos seguintes pacotes: *Model*, *View* e *Controller*. A Figura 5.23 apresenta uma visão geral do pacote.

A classe *RuleInit* também segue o padrão de projeto *Singleton* (GAMMA et ali, 1994) e representa a "porta de entrada" para acesso às funcionalidades da base de conhecimento. Além disso, a classe *RuleInit* é responsável por iniciar qualquer processo de inferência sobre a base de regras. A classe *RuleInit* é ainda ligada à base de regras propriamente dita que é a classe *RuleBase*. A classe *RuleBase* é o repositório de todas as regras atualmente disponíveis na infra-estrutura Odyssey. Para a criação das regras, são utilizadas as classes *RuleVariable*, responsável pela criação da variáveis que vão fazer parte das regras, classe *Condition*, que cria as condições a serem utilizadas nas cláusulas

das regras, a classe *Clause*, que cria as cláusulas participantes das regras, e a classe *Rule* propriamente dita, que cria as regras a serem utilizadas.

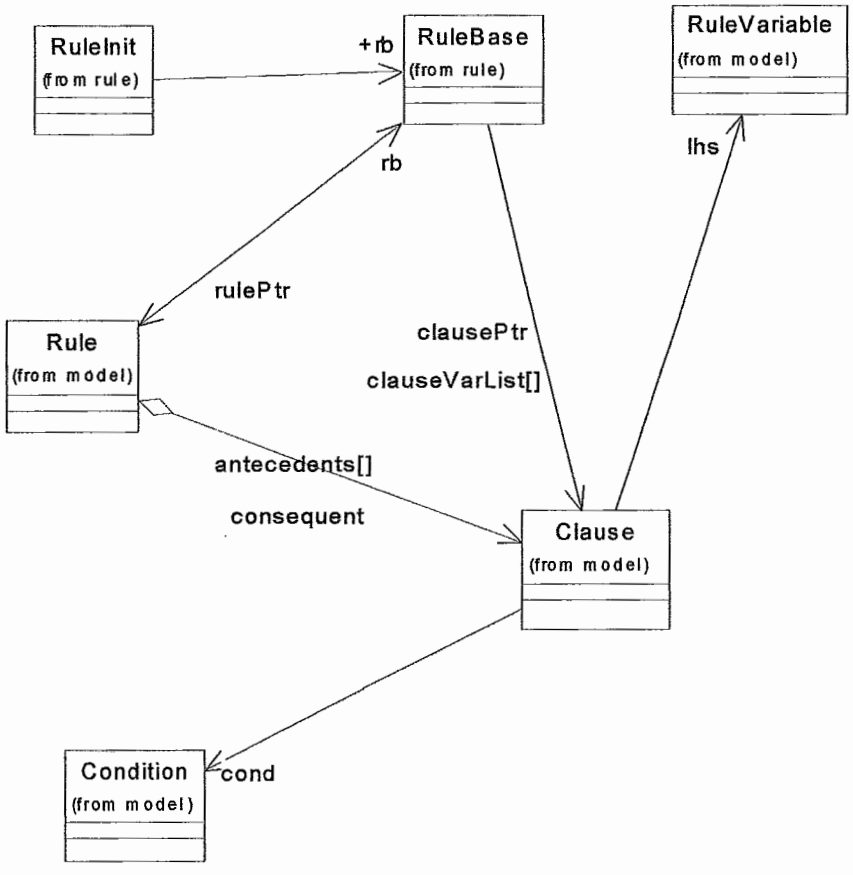

Figura 5.23 – Principais classes do pacote Rule

5.5 Características de Uso da Camada de Mediação

Um aspecto importante do agente de armazenamento e recuperação é que, apesar de fazer parte da Odyssey-Search como um todo, ele também pode ser utilizado de maneira isolada ou integrado com outro tipo de ferramenta que não a Odyssey-Search, uma vez que provê uma interface gráfica independente, o que permite que ele seja usado de maneira isolada.

Quando usado de forma isolada, o agente de armazenamento e recuperação possui dois módulos para interação, denominados *Service Manager* e *Mediator Manager*.

5.5.1 Service Manager

Para que se possa utilizar um dado mediador no contexto do Service Manager, o administrador deve registrar este mediador no gerente de serviços, informando as fontes de informação relacionadas a este mediador e os tradutores que devem ser utilizados para a conversão das informações armazenadas nas bases para um formato entendido pelo Agente de Recuperação. As Figuras 5.24 e 5.25 apresentam um exemplo das interfaces gráficas para o registro de mediadores e fontes de informação.

A Figura 5.24 apresenta o cadastramento do mediador para o Legislativo Municipal. Alguns dados básicos são: nome do mediador; o arquivo executável que deve ser carregado pelo barramento ORB (se ainda não estiver carregado), para que o mediador possa responder as requisições de consulta; palavras-chave relacionadas ao mediador, para facilitar uma possível busca pelo mediador; senha requerida pelo mediador para atender as requisições de consulta (se for necessário). A Figura 5.25 apresenta o registro de uma fonte de informação do Legislativo Municipal, associando uma fonte de dados específica (FileSystem2) com o mediador do Legislativo Municipal.

A Figura 5.26 apresenta um exemplo de consulta a múltiplos mediadores utilizando o Service Manager e a Figura 5.27 apresenta os resultados da consulta, inclusive com o nível de casamento semântico dos componentes recuperados.

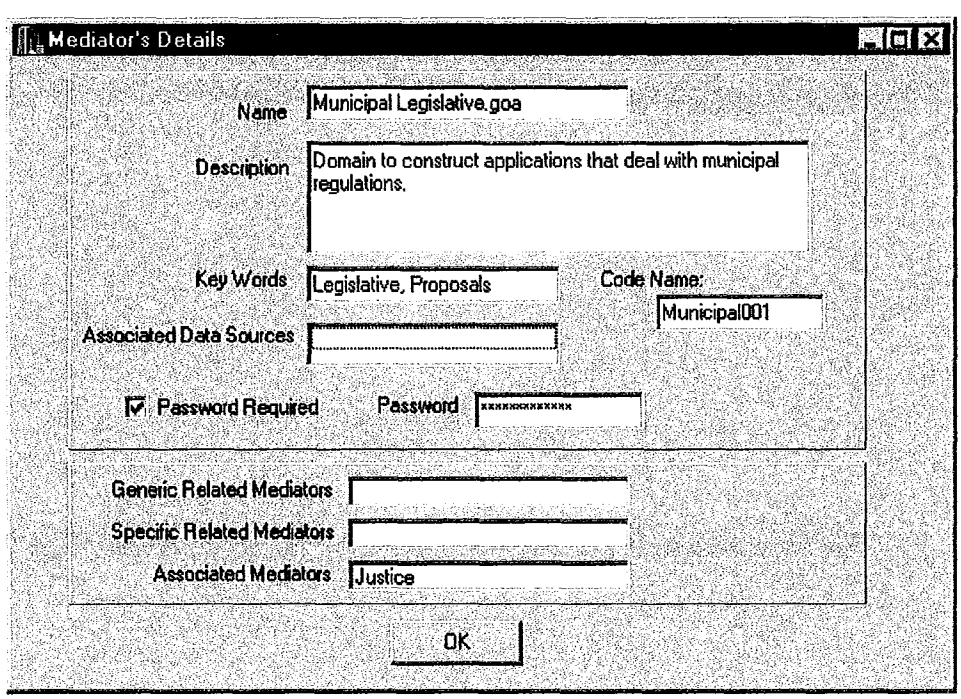

Figure 5.24 – Exemplo de registro de um mediador no *Service Manager*

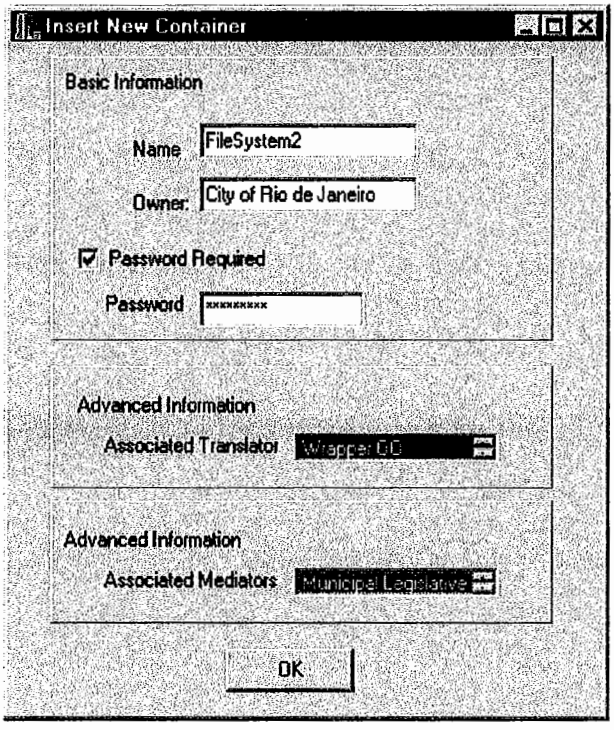

Figura 5.25 – Registro de uma fonte de dados no *Service Manager*

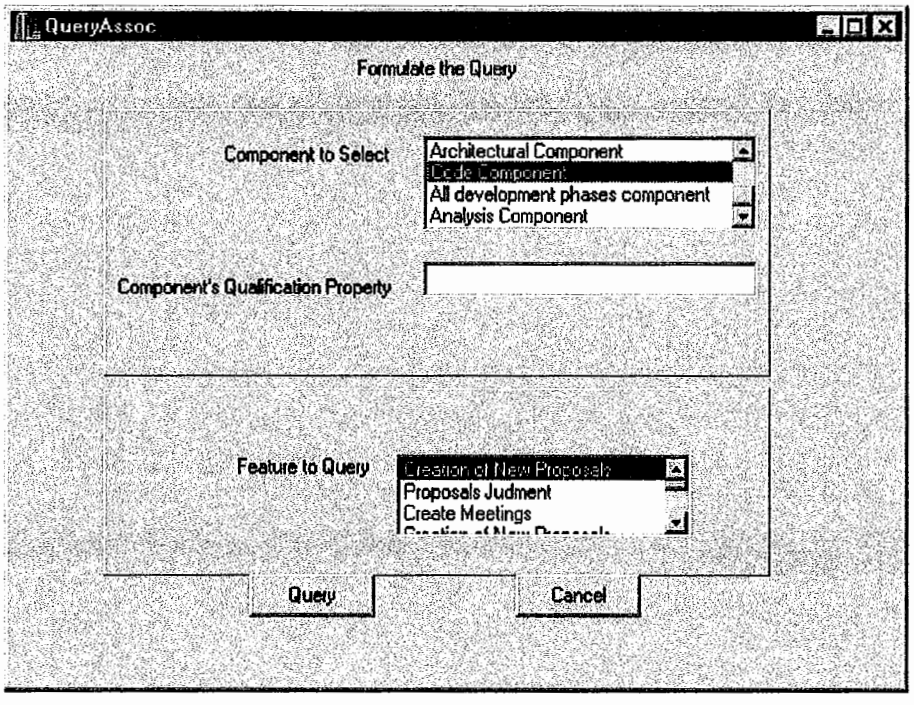

Figure 5.26 – Consulta utilizando a interface gráfica do *Service Manager*

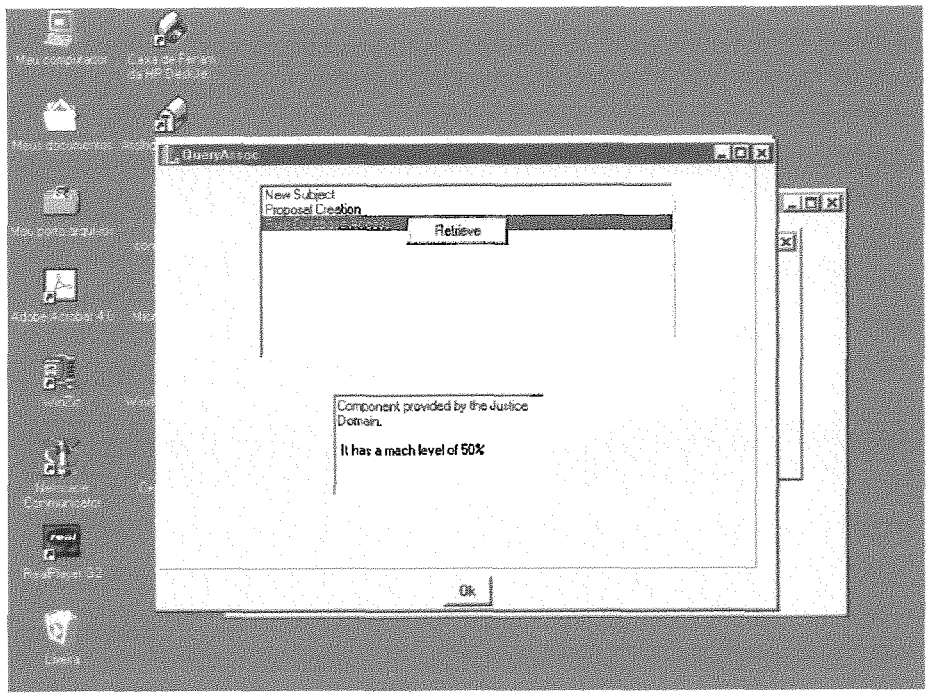

Figure 5.27 – Componentes recuperados pela consulta da Figura 5.26

5.5.2 Mediator Manager

Cada mediador tem seus próprios metadados, que na verdade é o modelo ontológico de cada domínio, incluindo os termos do domínio, seus relacionamentos e o relacionamento destes com as demais informações do domínio armazenadas nas diversas fontes de dados.

Quando o domínio foi criado utilizando a infra-estrutura Odyssey, o modelo ontológico é construído a partir do modelo de características utilizado pelo processo Odyssey-DE. No entanto, se o domínio não foi criado no contexto da infra-estrutura Odyssey, os termos do domínio devem ser cadastrados e seus relacionamentos com outros termos também. Neste último caso, é utilizada a interface do Mediator Manager para a criação destes termos. A Figura 5.28 apresenta o registro de um termo ontológico no domínio do Legislativo Municipal, através da Interface gráfica do Mediator Manager. Para cada termo ontológico é registrado o seu nome, seu tipo (funcionalidade ou conceito), sua importância no contexto do domínio e os termos relacionados. Este termos relacionados permitem a expansão da consulta no contexto do domínio, i.e., se não existem componentes do domínio diretamente relacionados a este termo, o Agente de Recuperação pode utilizar estes termos relacionados para recuperar componentes, devendo esta mudança ser comunicada ao usuário.

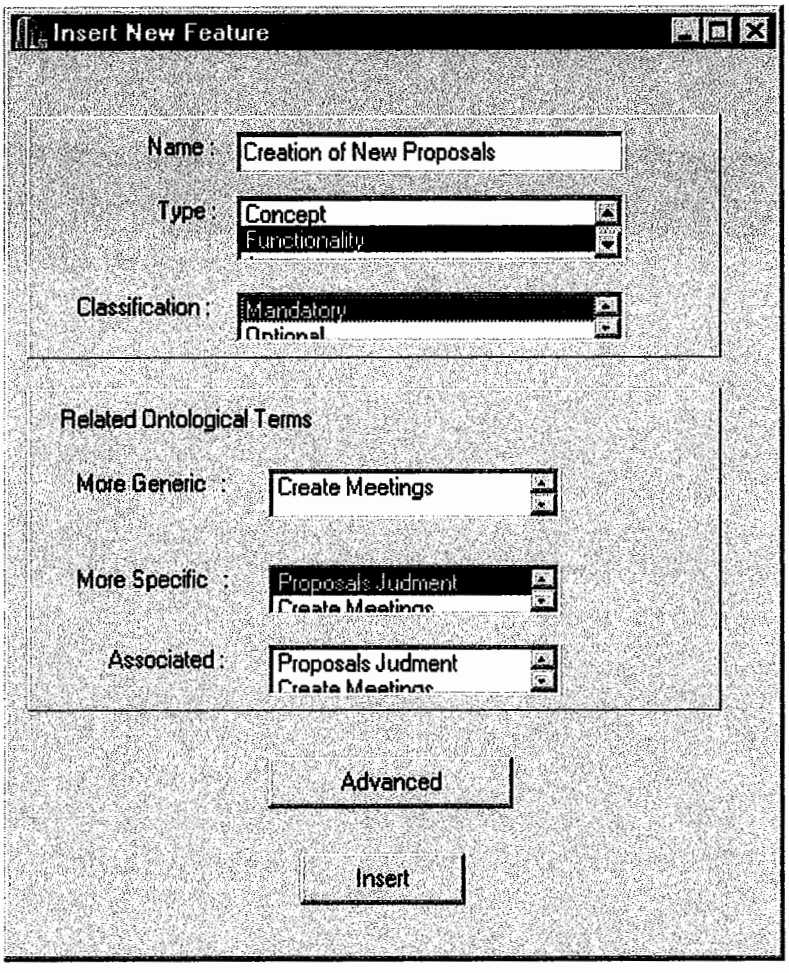

Figure 5.28 – Registro de um termo ontológico

5.6 Conclusões

O armazenamento, busca e recuperação por informações do domínio já foi largamente discutido na literatura (ARANGO, 1988), (MILLI *et al.*, 1995), (PRIETO, 1991), (SAMETINGER, 1997). No entanto, até hoje não se tem uma solução realmente definitiva para o problema, apesar de existirem abordagens que tratam do assunto (NEIGHBOURS, 1981), (MILLI, 1995). Algumas indicações em relação a quais caminhos devem ser seguidos é apresentada na literatura pontualmente por diversos autores:

- Particionamento das informações por domínio de aplicação (NEIGHBOURS, 1981), (KRUEGER, 1992). No caso específico da proposta DRACO (NEIGHBOURS, 1981), os componentes são organizados em uma bilbioteca , onde cada componente é parte da semântica da linguagem do domínio;
- Uso de redes semânticas para a busca (PRIETO, 1991);
- Permitir a busca por informações distribuídas (SEACORD, 1998);

- Antecipar as necessidades dos usuários em relação às informações do domínio (YE, FISCHER, 2000).

A Odyssey-Search tenta abranger todos estes pontos, através da utilização de tecnologias como mediadores, perfis de usuário e ontologias do domínio. Algumas características da Odyssey-Search que envolvem estas tecnologias são:

- As informações do domínio são particionadas por domínio de aplicação através da adoção da tecnologia de mediadores.
- As buscas são baseadas em ontologias do domínio, que é considerada uma rede semântica sofisticada.
- A distribuição e a heterogeneidade das informações é tratada através da utilização das camadas de mediação e do modelo de comunicação CORBA.
- As necessidades dos usuários são antecipadas através da utilização de técnicas inteligentes, como base de conhecimento e perfis de usuário, onde histórico de navegações, palavras-chave e preferências do usuário são utilizadas para refinar e melhorar a busca por informações;

Em relação ao **armazenamento** de componentes reutilizáveis, conforme ressalta SAMETINGER (1997), deve-se seguir duas abordagens principais: a divisão por domínios de aplicação e a utilização de um mecanismo de referência que permita a recuperação das informações por domínio. A abordagem utilizada pela Odyssey-Search está em conformidade com estas características, indo mais além, através do uso de relacionamentos ontológicos para realizar a busca em diferentes domínios.

Atualmente, sem a utilização da Odyssey-Search, um desenvolvedor de aplicações que quisesse reutilizar componentes do domínio no desenvolvimento de aplicações, i.e., **buscar e recuperar** componentes do domínio, teria como opção buscá-los através de ferramentas de busca na web genéricas, que se baseiam na busca simples por palavras-chave. Neste tipo de ferramenta, o desenvolvedor seria provavelmente inundado por informações irrelevantes, que apenas o fariam desistir da busca. Mesmo que a busca retornasse algum tipo de informação interessante, os componentes relacionados poderiam não estar em um formato adequado, requerendo algum tipo de conversão. Além disso, o desenvolvedor não teria disponível nenhum tipo de mecanismo que poderia levá-lo a informações relacionadas interessantes.

5.6.1 Análise das Técnicas de Busca utilizadas na Odyssey-Search

Na literatura de IA, diversas técnicas de aprendizado podem ser encontradas, com destaque atualmente para aquelas que utilizam mecanismos de aprendizado de máquina. As técnicas utilizadas na Odyssey-Search foram escolhidas principalmente levando-se em consideração as necessidades de navegação pelos itens do domínio, o volume destas informações e o desempenho necessário à Odyssey-Search. Assim, por conta destas necessidades, não utilizamos nenhum mecanismo como redes bayesianas ou neurais (WOOLDRIDGE, 2000).

Consultando a literatura e especialistas na área, pudemos comprovar que estas técnicas, apesar de serem muito eficientes, requerem dados de treinamento (dados históricos) e um retreinamento destes dados (reprocessamento dos dados), que por questões de desempenho não seria adequado para as navegações no contexto da Odyssey-Search.

Desta forma, optamos por utilizar técnicas mais simples como inferência, utilizando algoritmos de *forward chaining*, histórico de navegações e algoritmos caça-palavras. Apesar de não representarem o estado da arte em técnicas de aprendizagem, são adaptações que se mostraram adequadas para as necessidades da Odyssey-Search, o que não impede de futuramente, com o surgimento de novas técnicas ou o aprimoramento das existentes em termos de desempenho, que os algoritmos atuais sejam revistos.

6 Estudos de Casos

6.1 Introdução

Neste capítulo, apresentamos dois estudos de casos realizados com o objetivo de identificar os benefícios advindos da utilização das abordagens descritas nos capítulos 4 e 5 desta tese. Para isso, seguimos a hipótese levantada por KITCHENHAN *et al.* (1995) que ressalta que um bom estudo de caso, apesar de não ter o rigor dos experimentos formais, pode ser adequado para avaliar os benefícios advindos do uso de um processo ou técnica em um dado contexto.

Segundo Kitchenhan, um bom estudo de caso pode prover informação suficiente para ajudar a julgar se uma tecnologia específica irá trazer benefícios ou não em um dado contexto. O primeiro estudo descrito neste capítulo está de acordo com a definição dada por Kitchenhan para estudo de caso, uma vez que foi aplicado em um projeto típico, que é o projeto de desenvolvimento de componentes reutilizáveis no domínio de Processamento Legislativo, não apresenta replicação, o que é necessário no caso de experimentos formais, e possui muitas variáveis, de forma que é difícil especificar um conjunto de variáveis a serem controladas. O segundo estudo de caso não atende exatamente a este formato, uma vez que pode ser replicado. No entanto, não pode também ser classificado como experimento formal. Assim, classificaremos este como sendo um estudo de caso, mesmo sendo mais controlado do que um estudo de caso na classificação dada por Kitchenhan, mas não o bastante para ser considerado um experimento.

Nosso objetivo principal para a realização, em um primeiro momento, do estudo de caso, foi identificar um domínio típico, onde a reutilização de componentes pudesse ser aplicada. O domínio de processamento legislativo é um domínio típico e adequado para o desenvolvimento de componentes reutilizáveis, uma vez que é um domínio estável, compartilhado pelas diversas casas legislativas brasileiras e ainda necessitando de desenvolvimento de um grande número de aplicações. O segundo estudo de caso foi realizado no mesmo domínio de aplicação e teve como objetivo principal observar a utilização de ferramental específico para a busca e entendimento dos componentes do domínio especificados no primeiro estudo.

Neste contexto, o grupo de reutilização da COPPE/UFRJ e a Câmara Municipal do Rio de Janeiro (CMRJ) iniciaram uma parceria através de um projeto de pesquisa,

em 1998, para a realização de um processo de análise no domínio de tramitação de projetos em casas legislativas. Os resultados iniciais desta parceria estão descritos em ZOPELARI (1998). Neste trabalho, foi identificada a viabilidade e a necessidade de se desenvolver componentes reutilizáveis neste domínio. Esta necessidade se deve, principalmente, à carência por grande parte das casas legislativas brasileiras, de aplicativos na área de tramitação de projetos, o que demanda, atualmente, um desenvolvimento rápido e eficiente destas aplicações. No entanto, existe também uma carência, por parte destas casas legislativas, de desenvolvedores qualificados para tal desenvolvimento.

Devido a estes fatores, a CMRJ decidiu dar continuidade à parceria e se propôs a disponibilizar documentação, desenvolvedores e especialistas do domínio para envolvimento direto no projeto. Outro fator motivador para a realização dos estudos de caso aqui apresentados, foi a identificação em (ZOPELARI, 1998), da necessidade de adoção de uma sistemática bem definida para o desenvolvimento de componentes reutilizáveis no domínio. Nossa hipótese é que esta necessidade possa ser suprida pelo processo Odyssey-DE. Além disso, são conhecidas, pela comunidade de reutilização de software, as dificuldades encontradas na busca e seleção de componentes reutilizáveis para um dado domínio. Assim, o segundo estudo de caso contempla a utilização de uma ferramenta para a busca por componentes reutilizáveis no domínio, por grupos distintos de usuários, a saber: um grupo envolvendo alunos não familiarizados com o domínio e outro grupo envolvendo desenvolvedores do domínio.

Para a realização de ambos os estudos de caso, seguimos a sistemática apresentada em KITCHENHAN *et al.* (1995), que sugere três etapas distintas na elaboração de um estudo de caso:

- Planejamento: Consiste na definição do estudo de caso, identificando-se os participantes e os objetivos a serem alcançados, de acordo com a situação que se deseja experimentar;
- Monitoração: Diz respeito ao acompanhamento da execução de cada etapa do estudo, registrando-se o tempo de duração, as dificuldades encontradas e os resultados obtidos. Esta etapa foi ainda dividida, no contexto desta tese, em: técnica utilizada, avaliação do tempo e resultados, dificuldades e detalhamento;

- Avaliação dos resultados: Consiste na descrição dos resultados gerais obtidos no estudo, utilizando os dados da monitoração para avaliar a situação experimentada.

Descrevemos a seguir os estudos de caso realizados com base nas etapas definidas acima. Como se trata de um domínio não muito conhecido do público em geral, detalhamos a seguir como se dá o processamento legislativo. Para isso, escolhemos um projeto específico e de grande importância nas casas legislativas de forma geral, que é o projeto do orçamento. Este projeto tem algumas particularidades em relação aos demais, mas ilustra de maneira adequada a tramitação de projetos em geral.

6.1.1.1 Tramitação do projeto de Orçamento Municipal

O projeto orçamentário é enviado pelo poder executivo para a casa legislativa até a data estipulada por lei. Este projeto é enviado contendo sua redação original e uma mensagem do representante do poder executivo, neste caso o prefeito, encaminhando o mesmo. O **presidente da casa legislativa** é encarregado de receber o projeto e dar o despacho, o que permite que o projeto inicie seu processo de tramitação na casa legislativa.

Dado o despacho, o projeto é enviado para a **Secretaria Geral da Mesa Diretora (SGMD)**, que o numera, envia para publicação no jornal oficial de circulação diária e depois despacha o projeto para a **Comissão de Justiça e Redação (CJR)**, que avaliará se o projeto está de acordo com as normas jurídicas vigentes. Neste caso específico do projeto do orçamento, a passagem pela CJR é dispensável. No entanto, se o projeto passar pela CJR, esta emitirá parecer em relação ao projeto e o devolverá para a SGMD. A SGMD arquiva o original do parecer, envia cópia do parecer para publicação e envia o projeto para a **Comissão de Finanças, Orçamento e Fiscalização Financeira (CFOFF).** A CFOFF disporá de um certo número de dias para avaliar o projeto e emitir seu parecer. De posse do parecer, a CFOFF devolve o projeto para a SGMD, que arquiva o original do parecer da CFOFF, envia cópia do parecer para publicação e coloca o projeto na ordem do dia para Primeira Discussão. **Parlamentares, comissões, bancadas e lideranças** podem elaborar então emendas e subemendas por um certo número de dias. Estas emendas e subemendas são recebidas pela SGMD que as numera, envia para publicação e despacha para a CJR para emissão de parecer a respeito das emendas. A CJR dá o parecer para as emendas e subemendas e o envia para a SGMD.

Novamente, a SGMD arquiva o original do parecer, envia cópia do mesmo para publicação e envia as emendas e subemendas para a CFOFF. A CFOFF emite parecer às emendas e subemendas e envia o parecer para a SGMD. A SGMD arquiva o original do parecer, envia cópia para publicação e distribui as emendas, subemendas e substitutivos em avulsos para votação em plenário. O plenário vota então as emendas. De acordo com o resultado da votação, a SGMD envia o projeto para a CJR para que esta elabore a redação do vencido (que é a redação do projeto com as emendas, substitutivos e subemendas aprovadas). A CJR elabora a redação do vencido e envia para a SGMD, que posteriormente envia para publicação e para votação em plenário. Assim, a redação do vencido é votada em plenário. Se a redação do vencido for aprovada, esta é finalmente enviada para publicação como redação final. Caso não tenham sido apresentadas emendas em primeira discussão ou se tenha algum outro motivo, os projetos serão votados e voltarão na ordem do dia subsequente para Segunda discussão. A Segunda discussão transcorre como a primeira, podendo-se apresentar emendas, subemendas e substitutivos, sendo que findo o prazo da Segunda discussão, é elaborada a redação final, que é publicada e aprovada.

Sendo a redação final aprovada, esta é enviada para sanção, onde a mesma pode receber veto total, ser sancionada sem vetos ou sancionada com vetos. Se for sancionada com vetos, deve ser enviada para as comissões CJR e CFOFF para que as mesmas emitam parecer. De posse dos pareceres, é realizada uma discussão única a respeito dos vetos, que podem ser mantidos ou derrubados. Após esta etapa, o projeto vira lei.

6.2 *Desenvolvimento de componentes reutilizáveis no domínio de processamento legislativo utilizando o Odyssey-DE*

Em (ZOPELARI, 1998), foram descritos alguns estudos relativos à aquisição de conhecimento em um processo de análise de domínio e chegou-se à constatação que é necessária uma sistematização do processo de aquisição. Em um contexto mais amplo, que englobaria a aquisição do conhecimento e sua representação, gerando-se componentes reutilizáveis, esta necessidade é ainda maior, uma vez que o tempo de duração é elevado, com um grande número de atividades, sendo estas atividades mais complexas.

Assim, foi definido em 1998 o processo de engenharia de domínio Odyssey-DE, que, conforme descrito no capítulo 4 desta tese, contempla técnicas de engenharia de domínio com métodos de desenvolvimento baseado em componentes (BRAGA e

WERNER, 1999). Elaborado o processo original, partiu-se para a realização de um estudo de caso que pudesse auxiliar na avaliação dos benefícios trazidos pela adoção de tal processo em uma situação real.

Desta forma, o estudo de caso foi iniciado em janeiro de 1999 e teve duração de um ano e meio, sendo finalizado, para o contexto desta tese, em junho de 2000. O processo de engenharia de domínio como um todo foi realizado com o apoio principal da CMRJ, mas também utilizando especialistas e documentação da Assembléia Legislativa do Rio de Janeiro (ALERJ) e validação dos modelos por especialistas e desenvolvedores de outras casas legislativas, no contexto do projeto InterLegis, que é um projeto de integração e desenvolvimento de casas legislativas brasileiras. Desta forma, o escopo de abrangência do processo, apesar do apoio maior ter sido da CMRJ, contempla também outras casas legislativas. No caso específico de outras casas legislativas, pudemos contar com o apoio de 1 especialistas e 1 desenvolvedor da ALERJ, sendo o especialista profundo conhecedor do domínio de processamento legislativo. O apoio de especialistas e desenvolvedores das casas legislativas de Minas Gerais, Bahia, São Paulo, Espírito Santo e Pernambuco também foi decisivo para o processo, uma vez que auxiliaram no sentido de validar os modelos especificados.

Vale a pena ressaltar ainda que a parceria entre o grupo de reutilização da COPPE/Sistemas e a CMRJ continua ativa e está atualmente formalizada através de um projeto de transferência de tecnologia para o desenvolvimento de uma biblioteca de componentes reutilizáveis para o processamento legislativo. Descrevemos a seguir o estudo de caso, seguindo as etapas propostas por KITCHENHAN *et al.* (1995).

6.2.1 Planejamento

Um projeto piloto foi estabelecido entre a COPPE/Sistemas e a CMRJ para a aplicação do processo de engenharia de domínio Odyssey-DE, com o objetivo de se criar componentes reutilizáveis no domínio do legislativo.

Ficou estabelecido que todas as etapas constantes do processo seriam realizadas e ao final teríamos uma validação dos componentes por parte da equipe de desenvolvedores da CMRJ. Além disso, seria utilizada uma aplicação existente no domínio e demonstraria-se até que ponto os modelos/componentes especificados através do Odyssey-DE seriam adequados e poderiam auxiliar na rápida melhoria e evolução da aplicação. Assim, a hipótese do estudo de caso foi avaliar até que ponto o Odyssey-

DE se mostrou adequado para o desenvolvimento de componentes reutilizáveis no domínio.

A equipe participante para a realização do projeto foi assim composta: 2 gerentes de projeto, 2 desenvolvedores do domínio, 1 especialista do domínio e 1 engenheiro do domínio. Além disso, posteriormente o projeto contou com o auxílio de especialistas do domínio de outras casas legislativas. Ficou estabelecido que seriam realizadas reuniões para a captura do conhecimento do domínio através da análise dos sistemas existentes, documentação existente e especialistas disponíveis. Ficou estabelecido o prazo de um ano para a realização do estudo.

6.2.2 Monitoração

Durante a fase de monitoração, procurou-se registrar as dificuldades, o tempo e os resultados de cada etapa. Estes registros foram feitos com o objetivo de permitir uma avaliação dos resultados deste estudo. Apresentamos a seguir as diversas etapas realizadas, de acordo com as fases do Odyssey-DE.

6.2.2.1 Etapa: Estudo de Viabilidade do Domínio na CMRJ

Técnica utilizada:

Brainstorming

Avaliação do tempo e resultados:

Foi realizada uma reunião que durou 6 horas. Os participantes da reunião foram o engenheiro do domínio e dois gerentes de projeto. Na reunião, foram levantados os domínios passíveis de sofrerem um processo de ED, os sistemas existentes e a disponibilidade das fontes de informação.

Dificuldades:

- Agendamento da reunião: Foi difícil reunir o engenheiro de domínio e os dois gerentes de projeto envolvidos.
- Baixa produtividade, por ter a reunião sido realizada ao longo de um dia inteiro (6 horas), ao final, o aproveitamento ficou prejudicado.

Detalhamento da Etapa:

O objetivo principal da seleção de um domínio adequado é a minimização do risco através da identificação de um domínio que tem chance de ser analisado com sucesso, dados o tempo e recursos disponíveis. Foram realizadas as seguintes fases:

- **Seleção dos domínios candidatos ao processo de reutilização;**

  Foram selecionados pela equipe três domínios relacionados, o domínio de Processos Legislativos, o domínio de Processos Administrativos, e, englobando os dois anteriores, o domínio de Processos da Câmara em geral. Partindo-se do princípio de que em um domínio muito abrangente torna-se difícil a realização de uma análise adequada, decidimos pela realização do estudo em um domínio menos abrangente. Decidimos então pela realização do estudo de viabilidade no domínio de Processos Legislativos. O domínio de processos legislativos foi o escolhido principalmente pela sua importância estratégica no contexto das casas legislativas, em geral, e por ser este o domínio que mais necessita atualmente de apoio automatizado em seus procedimentos. Assim, por unanimidade, foi escolhido o domínio de processos legislativos para a realização do estudo.

- **Seleção de critérios para a avaliação de domínios**

  Na fase de seleção de critérios para a avaliação de domínios, são selecionados critérios gerais de reutilização e critérios específicos para a empresa. Após uma análise dos critérios apresentados, a equipe decidiu pela adoção somente dos critérios gerais, sem a adição de novos critérios específicos, sendo estes considerados suficientes para o objetivo proposto. As tabelas 6.1, 6.2 e 6.3 apresentam os resultados obtidos para cada critério analisado.

Técnicos

Critério	*Descrição*	*Resultados*
1. Número de sistemas existentes	Determina se existem sistemas suficientes no domínio para serem analisados para a criação dos modelos do domínio. Além disso, este critério mede também a maturidade do domínio	Foram identificados ao todo **12** sistemas já desenvolvidos no domínio 1. Sistema de Leis de Diretrizes Orçamentárias (desenvolvido em Delphi) 2. Sistema de Orçamento Anual (desenvolvido em Delphi) 3. Sistema de Regimento Interno (desenvolvido em Delphi, mas resultados são migrados para o Lotus Notes) 4. Sistema de Atas das Seções Plenárias (Desenvolvido em Lotus Notes) 5. Sistema de Processamento Legislativo, módulo de protocolos (Desenvolvido em Delphi e Lotus Notes) 6. Sistema de Ordem do Dia

		(Desenvolvido em Delphi e Lotus Notes) 7. Sistema de Lei Orgânica (Desenvolvido no Lotus Notes) 8. Sistema de Análise de Similaridades (Desenvolvido em Lotus Notes) 9. Sistema de Resoluções da Mesa Diretora (Desenvolvido em Lotus Notes) 10. Banco de Dados de Legislaturas (Desenvolvido em Lotus Notes) 11. VIP (Visualização do Plenário) 12. Home-Page da Câmara (HTML e Notes) * É importante ressaltar que a maioria destes sistemas, principalmente os desenvolvidos em Lotus Notes, são bases de dados cujo processamento é realizado pelo próprio Lotus Notes.
2. Futuros sistemas a serem desenvolvidos	Determina se existem necessidades futuras de desenvolvimento de sistemas no domínio	Foram identificados ao todo 7 sistemas a serem desenvolvidos no domínio, sendo o sistema de Tramitação Eletrônica o mais importante de todos. 1. Sistema de Racionalização da Taquigrafia, Debates e atas 2. Sistema de Consulta Popular 3. Painel de Votação Eletrônica 4. Digitalização do Acervo Legislativo 5. Catálogo Geral dos Parlamentares 6. Terminal Cidadão 7. Tramitação Eletrônica (60% de todo o domínio é composto por este sistema)
3. Melhoramentos nos sistemas existentes	Determina se existem sistemas no domínio que devem ser melhorados	Como principal melhoramento nos sistemas relacionados no item 1 foi identificada a necessidade de integração dos mesmos. Além disso, foi identificado que os sistemas 5 e 6 necessitam passar por uma remodelagem, 1 e 2 necessitam de pequenas modificações.
4. Recursos técnicos disponíveis para a ED	Determina se existem recursos técnicos disponíveis para o projeto. Ex. linguagem de programação OO, protocolo CORBA, componentes legados, etc.	Atualmente estão disponíveis para o trabalho os seguintes itens • Rede Novell sendo migrada para rede Windows NT • Ambiente de Programação Delphi, que possui uma linguagem de programação OO Híbrida • Sistema de Banco de Dados Relacional Oracle • Sistema de Banco de Dados Access • Sistema de Processamento de Documentos Lotus Notes
5. Estabilidade do domínio	Se a variação de um sistema para outro no domínio for grande, pode não ser possível produzir componentes reutilizáveis ou mantê-los corretos. Este critério mede até que ponto os sistemas no domínio são parecidos.	Foi identificado pela equipe que, em relação os sistemas internos na Câmara, a estabilidade do domínio seria Média e em relação aos sistemas que seriam necessários em outras câmaras, a estabilidade seria Alta.
6. Acessibilidade das informações	Este critério mede a disponibilidade de informações técnicas sobre o domínio,	• Documentação dos sistemas - Pouca Documentação

	ou seja, se os sistemas já desenvolvidos no domínio foram bem documentados.	• Disponibilidade de Especialistas e Desenvolvedores no domínio - Alta • Código dos sistemas Prontos - Alta • Referências Técnicas sobre o domínio - Alta
7. Metodologias utilizadas no desenvolvimento de sistemas no domínio	O uso de uma metodologia consistente faz com que seja fácil comparar os sistemas existentes e documentação para se identificar as similaridades e diferenças entre os sistemas existentes.	• Uso de Metodologia Estruturada na maioria dos sistemas (documentação básica é o MER) • Especificamente o sistema do orçamento foi desenvolvido usando metodologia OO, usando ferramenta CASE.

Tabela 6.1 – Critérios técnicos levantados

Organizacionais

Critério	*Descrição*	*Resultados*
1. Potencial para reutilização	Este critério mede o quão útil os produtos da ED podem ser se novos sistemas no domínio forem desenvolvidos ou se sistemas já existentes forem melhorados. Como o principal objetivo da ED é a reutilização, este critério, apesar de muito geral, é importante	• Produtos da análise de domínio servirão como fonte de informação (boa documentação) • Componentes em código que possam sempre evoluir • Facilitação de adaptações evolutivas • Desenvolvimento mais rápido a partir dos componentes existentes.
2. Conhecimento em reutilização na empresa e experiência prática na aplicação da reutilização	Um alto nível de experiência em reutilização torna fácil a explicação de conceitos e produtos da ED. No entanto, o treinamento em reutilização pode auxiliar no processo se o conhecimento em reutilização for um problema.	• Pouco ou nenhum conhecimento em Reutilização da equipe interna • Necessidade de treinamento
3. Compromisso a nível gerencial	Este é um dos critérios mais críticos para assegurar o sucesso da ED. Havendo o comprometimento dos gerentes com o projeto, recursos e tempo serão adequadamente alocados.	• ASSIMA (compromisso total) • Mesa Diretora da Câmara Municipal do Rio de Janeiro (neutra) • Prodasen, através do projeto Interlegis (compromisso total)
4. Conhecimento do Domínio	É muito importante que se tenha pessoas que entendam do domínio como um todo e de algumas áreas em particular. Os especialistas do domínio são uma fonte de informação e validação importantes para a ED.	• Grande conhecimento do domínio
5. Escopo	Os domínios variam muito em tamanho. Em um domínio muito abrangente é difícil se fazer uma análise adequada. A literatura demonstra que estórias de sucesso em ED ocorrem em domínios pequenos e bem delimitados.	Médio a Pequeno

Tabela 6.2 – Critérios organizacionais levantados

Sociais

Critério	*Descrição*	*Resultados*
1. Necessidade de se ter informações acerca do domínio para o treinamento de novatos	O treinamento do grupo pode ser uma característica necessária no domínio	• Treinamento no domínio de aplicação - pouco • Treinamento Técnico - Grande
2. Participação dos	Para capturar as informações dos	• Grande Facilidade

especialistas no domínio	especialistas no domínio e assegurar que estes especialistas vão reutilizar os produtos da ED em suas novas aplicações, a participação adequada dos especialistas do domínio é crítica.	• Muitas pessoas interessadas
3. Consistência dos especialistas do domínio	Os especialistas do domínio podem perceber o domínio de diferentes maneiras. Se novos especialistas forem consultados durante a ED, os produtos da mesma podem mudar significativamente. Isto pode causar atrasos e invalidação dos resultados prévios.	• Entre os especialistas - consenso • Entre os especialistas e desenvolvedores - algumas divergências
4. Comprometimento dos desenvolvedores de aplicações no domínio com as técnicas utilizadas pelo processo de ED	O grau de comprometimento e conhecimento dos desenvolvedores com as técnicas utilizadas pelo Odyssey. Ex. paradigma OO, CORBA, Java, desenvolvimento baseado em componentes.	• A princípio sem problemas (a conquistar)

Tabela 6.3 – Critérios sociais levantados

- **Pontuação e totalização dos critérios de seleção**

  Uma vez definidos quais são os critérios a serem analisados, inicia-se a fase de pontuação e totalização dos critérios de seleção.

  Nesta fase são definidos pesos para cada um dos critérios. Assim, apresentou-se como uma possível proposta o seguinte procedimento: cada participante na seleção dos domínios define individualmente pesos para cada critério, baseando-se na importância do critério, onde 1 é a mais baixa pontuação e 10 a mais alta. Foi realizada então uma reunião entre o engenheiro do domínio e um dos gerentes para se formular um peso mediano para cada critério. As tabelas 6.4, 6.5 e 6.6 apresentam os pesos e as devidas justificativas para a escolha dos mesmos.

Técnicos

Critério	*Pesos*	*Justificativa para o Peso*
1. Número de sistemas existentes	5	Os sistemas existentes são uma boa fonte para a Eng. de domínio, mas não a única fonte.
2. Futuros sistemas a serem desenvolvidos	7	Este critério mede o potencial que o domínio tem de expansão e por isso tem importância acentuada. No entanto, a engenharia de domínio pode ser utilizada também com outros propósitos, como, por exemplo, o treinamento de pessoal, apesar do objetivo principal do projeto ser o desenvolvimento de componentes no domínio
3. Melhoramentos nos sistemas existentes	6	Mede também o potencial que os componentes reutilizáveis poderão ter no domínio para mudança dos sistemas existentes. Sua importância pode ser comparada a do item 3.
4. Recursos técnicos disponíveis para a ED	6	A disponibilidade de recursos técnicos para a realização da ED é um item importante. No entanto, no caso do

		projeto este recursos podem ser adquiridos ao longo do tempo
5. Estabilidade do domínio	9	É de acentuada importância que o domínio seja estável. Quanto mais estável o domínio mais componentes reutilizáveis poderão ser desenvolvidos.
6. Acessibilidade das informações	10	Se as informações não estiverem acessíveis, será impossível realizar a ED.
7. Metodologias utilizadas no desenvolvimento de sistemas no domínio	7	O uso de uma metodologia de desenvolvimento consistente facilita a comparação entre os sistemas, comparando suas similaridades e diferenças.

Tabela 6.4 – Pesos para critérios técnicos

Organizacionais

Critério	*Pesos*	*Justificativa para o Peso*
1. Potencial para reutilização	10	Como o principal objetivo da ED é a reutilização, seu potencial é crucial para qualquer esforço neste sentido
2. Conhecimento em reutilização na empresa e experiência prática na aplicação da reutilização	3	Conhecimento e experiência em reutilização facilita o entendimento dos conceitos e produtos da ED. No entanto, o treinamento pode suplantar esta deficiência.
3. Compromisso a nível gerencial	10	Este é um dos fatores mais críticos para assegurar o sucesso da ED. Com o comprometimento a nível gerencial recursos e tempo necessários serão alocados para o projeto.
4. Conhecimento do Domínio	8	É importante se ter pessoas com grande conhecimento do domínio. Estas pessoas são importantes fontes de informação e validação dos produtos da ED. No entanto, não são a única fonte de informação.
5. Escopo	7	O escopo do domínio deve ser de pequeno a médio para que seja possível a realização de uma boa ED.

Tabela 6.5– Pesos para critérios organizacionais

Sociais

Critério	*Pesos*	*Justificativa para o Peso*
1. Necessidade de se ter informações acerca do domínio para o treinamento de novatos	8	Os produtos da ED podem ser uma boa fonte de treinamento para novatos no domínio.
2. Participação dos especialistas no domínio	10	A participação dos especialistas do domínio é critica para a validação do produto. Mesmo não sendo a única fonte de informação para o domínio, sua participação para validação é crucial.
3. Consistência dos especialistas do domínio	7	Se as opiniões divergem muito entre os especialistas pode ser difícil o desenvolvimento de produtos realmente reutilizáveis.
4. Comprometimento dos desenvolvedores de aplicações no domínio com as técnicas utilizadas pelo processo de ED	10	Se os desenvolvedores não considerarem as técnicas utilizadas adequadas, eles não reutilizarão os produtos desenvolvidos com aquelas técnicas, seja por não entenderem a técnica seja por não acharem útil.

Tabela 6.6 – Pesos para critérios sociais

Definidos os pesos para cada critério, coube a definição da faixa de valores de pontuação para cada um dos critérios. Os valores permitidos são 0, 1, 2, 3. Ficou definido que a pontuação de cada critério seria dada pela seguinte fórmula:

$$\text{Pontuação}_{\text{Critério}} = \text{Peso}_{\text{Critério}} * \text{Valor}_{\text{Critério}}$$

A pontuação total do domínio foi dada pela soma dos pontos de todos os critérios. A sugestão da pontuação mínima é feita com base em projetos anteriores realizados e o mínimo de pontuação alcançada nos mesmos. No caso específico da CMRJ, a pontuação mínima considerada foi a pontuação mínima descrita em projetos detalhados em (ARC, 1996), uma vez que não existia anteriormente na Câmara registro de projetos anteriores feitos neste formato. Assim, apesar dos domínios tratados pelo projeto ARC serem bastantes distintos do domínio legislativo, após uma reunião com os participantes do projeto, decidimos por adotar os mínimos propostos em projetos ARC anteriores e a partir de agora criarmos um histórico de estudos de viabilidade no domínio legislativo.

Esta pontuação mínima de projetos ARC é disponibilizada através de uma base de dados ACCESS (MICROSOFT, 2000), contando com cerca de 20 registros de projetos anteriores. Estes projetos são todos projetos na área de aeronáutica. Estes dados foram utilizados para considerarmos a pontuação mínima utilizada neste estudo de caso. As tabelas 6.7, 6.8 e 6.9, apresentam estas pontuações.

Técnicos

Critérios	*Faixa de Valores*	*Valor*	*Pesos*	*Pontos*
1. Número de sistemas existentes	(medido em número de sistemas) 0-1 = 0 1-2 = 1 3-5 = 2 6-n = 3	3	5	15
2. Futuros sistemas a serem desenvolvidos	(medido em número de sistemas a serem desenvolvidos) 0 = 0 1-2 = 1 3-4 = 2 5+ = 3	3	7	21
3. Melhoramentos nos sistemas existentes	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	2	6	12
4. Recursos técnicos disponíveis para a ED	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	2	6	12
5. Estabilidade do domínio	Nenhum = 0 Pouco = 1	3	9	27

Critérios	*Faixa de Valores*	*Valor*	*Pesos*	*Pontos*
	Médio = 2 Alto = 3			
6. Acessibilidade das informações	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	3	10	30
7. Metodologias utilizadas no desenvolvimento de sistemas no domínio	Não = 0 Em parte = 1 Sim = 3	1	7	7
Total de Pontos				124
Mínimo de Pontos neste critério para que o domínio seja viável				57

Tabela 6.7 – Pontuação para critérios técnicos

Organizacionais

Critérios	*Faixa de Valores*	*Valor*	*Pesos*	*Pontos*
1. Potencial para reutilização	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	3	10	30
2. Conhecimento em reutilização na empresa e experiência prática na aplicação da reutilização	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	1	3	3
3. Compromisso a nível gerencial	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	3	10	30
4. Conhecimento do Domínio	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	3	8	24
5. Escopo	Imensurável = 0 Grande = 1 Médio = 2 Pequeno = 3	2	7	14
Total de Pontos				101
Mínimo de Pontos neste critério para que o domínio seja viável				70

Tabela 6.8 – Pontuação para critérios organizacionais

Sociais

Critérios	*Faixa de Valores*	*Valor*	*Pesos*	*Pontos*
1. Necessidade de se ter informações acerca do domínio para o treinamento de novatos	Inadequado = 0 Adequado = 3	3	8	24
2. Participação dos especialistas no domínio	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	3	10	30
3. Consistência dos especialistas do domínio	Inconsistente = 0 Alguma inconsistência = 1 Alguma Consistência = 2 Consistente	1	7	7

Critérios	*Faixa de Valores*	*Valor*	*Pesos*	*Pontos*
4. Comprometimento dos desenvolvedores de aplicações no domínio com as técnicas utilizadas pelo processo de ED	Nenhum = 0 Pouco = 1 Médio = 2 Alto = 3	2 (ainda a conquistar)	10	20
Total de Pontos				81
Mínimo de Pontos neste critério para que o domínio seja viável				37

Tabela 6.9 – Pontuação para critérios sociais

Pontuação Total = 124 + 78+ 81 = 283

Pontuação Total Mínima para que o domínio seja viável = 57 + 70 + 37 = 164

Desta forma, concluiu-se que a realização de um processo de Engenharia de domínio no domínio de processos legislativos seria viável.

6.2.2.2 Etapa: Análise do Domínio

Devido à importância desta etapa e o seu nível de detalhes, descrevemos cada uma das fases:

- Descrever o domínio no contexto da organização;
- Elicitar o conhecimento do domínio, que é dividida em duas subfases:
  - Adquirir conhecimento mais específico do domínio;
  - Representar o conhecimento do domínio.

Fase: Descrever o domínio no contexto do organização

Técnica utilizada:

Uso dos resultados da etapa anterior para a modelagem do diagrama de contexto, utilizando o ferramental da infra-estrutura Odyssey.

Avaliação do tempo e resultados:

Esta fase foi realizada pelo engenheiro do domínio com o auxílio dos resultados da etapa anterior. O tempo de duração foi de 2 horas de trabalho, sendo que ao final foi especificado o diagrama de contexto.

Dificuldades:

Devido ao pouco conhecimento do engenheiro do domínio em relação ao domínio de processos legislativos, foi encontrada uma relativa dificuldade na especificação do diagrama de contexto, que foi refinado nas etapas posteriores.

Detalhamento:

Na Figura 6.1 é apresentado o diagrama de contexto para o domínio de processamento legislativo, que foi o domínio escolhido na etapa de estudo de viabilidade, onde é apresentado seu relacionamento com os principais atores e domínios relacionados. As ligações com os outros domínios, i.e., o Tribunal de Contas do Município, Poder Executivo e Imprensa Oficial, indicam que aplicações destes domínios trocam informações com aplicações do domínio em questão (no exemplo, o domínio de Processamento Legislativo). Os principais atores que interagem com o domínio estão também representados.

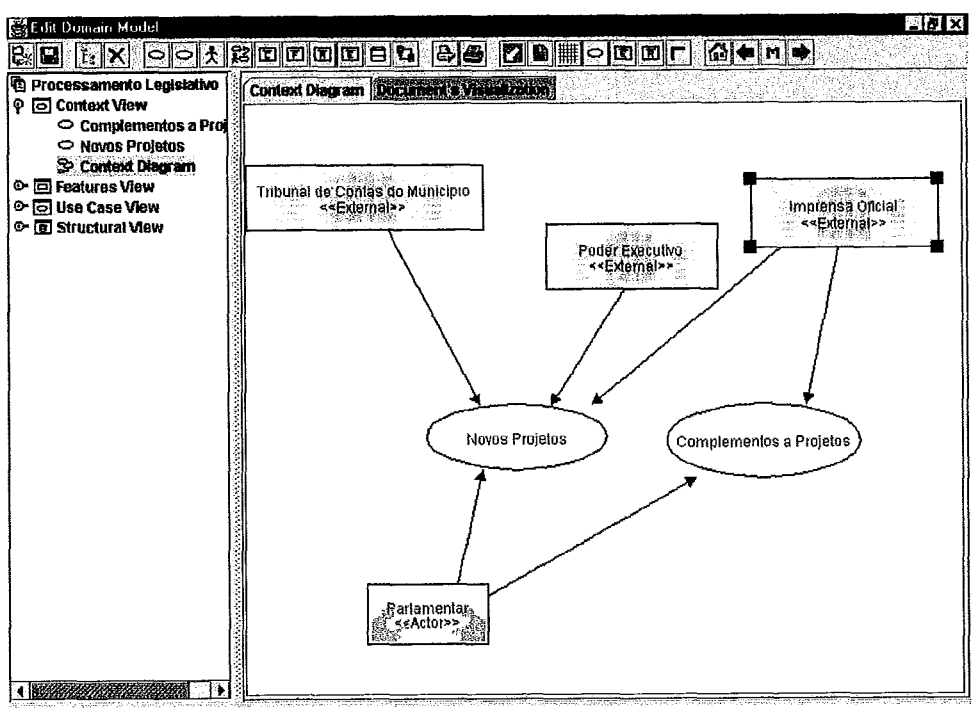

Figura 6.1 – Diagrama de contexto para o domínio de Processamento Legislativo

Fase: Adquirir conhecimento específico do domínio

Técnica utilizada:

Entrevistas com os desenvolvedores e especialistas do domínio. Descrição dos cenários a partir das entrevistas e documentação existente.

Avaliação do tempo e resultados:

Esta fase foi realizada em alternância com a fase de representação, uma vez que o conhecimento era capturado, representado e avaliado pelos especialistas e desenvolvedores no domínio. Ao longo das entrevistas foi disponibilizado para o engenheiro do domínio um material já existente de modelagem das etapas do processamento legislativo, utilizando análise essencial (CMRJ, 1988). Este material foi de grande importância na aquisição do conhecimento das principais funcionalidades do domínio. Além deste, foi disponibilizado o regimento interno da CMRJ, que auxiliou no entendimento dos principais conceitos relacionados ao domínio. A documentação dos sistemas existentes também auxiliou na captura das principais funcionalidades do domínio.

Esta fase transcorreu de acordo com a descrição a seguir, tendo como participantes o engenheiro do domínio, os desenvolvedores de aplicações e especialistas da CMRJ e de outras casas legislativas. Uma primeira reunião foi realizada, com duração de 3 horas, com um dos gerentes de projeto. Nesta reunião foi disponibilizado o material com a modelagem em análise essencial e o regimento interno da Câmara. Com um intervalo de 20 dias, foi realizada uma segunda reunião com um dos desenvolvedores de sistemas no domínio. Esta reunião teve duração de 5 horas e meia e serviu para a captura de mais informações e também para validar os casos de uso modelados até o momento. Nesta reunião, foram disponibilizados documentos de modelagem de aplicações já existentes no domínio. Com um intervalo de 3 meses, foi realizada uma terceira reunião, com outro desenvolvedor do domínio, com duração de 4 horas. Nesta reunião, uma nova validação dos modelos foi realizada e mais material acerca do domínio foi disponibilizado. Com os modelos de componentes já detalhados, surgiu a oportunidade de validar os modelos com especialistas e desenvolvedores de outras casas legislativas. Esta oportunidade surgiu no ENIAL 1999 (Encontro Nacional de Informática aplicada ao Legislativo), realizado em Salvador, em dezembro de 1999. Neste encontro, os modelos especificados até então foram apresentados para especialistas e desenvolvedores de casas legislativas de Minas Gerais, Bahia, São Paulo, Espírito Santo, Pernambuco e Rio de Janeiro, o que permitiu a captura de mais informações acerca do domínio, e como conseqüência, um posterior refinamento dos modelos. É importante ressaltar que, devido ao Odyssey-DE ser um processo em espiral, as atividades de captura, modelagem e validação ocorrem diversas vezes ao

longo do ciclo, o que pode ser observado pela descrição acima, onde se alterna a captura de informações, modelagem do domínio (detalhada na próxima fase) e validação. Outro ponto que vale a pena ressaltar é que ao longo de todo o período, o documento com a modelagem essencial de aplicações no domínio legislativo foi consultado, constituindo desta forma valiosa fonte de aquisição de conhecimento do domínio.

Dificuldades:

Houve uma certa dificuldade para o agendamento das reuniões, por conta de compromissos anteriores dos participantes. No entanto, as informações capturadas comprovaram que as funcionalidades do domínio são um bom começo para a identificação dos componentes reutilizáveis. Em relação à especificação dos modelos utilizando a infra-estrutura Odyssey, houve uma certa dificuldade por não haver ainda suporte para a definição das interfaces do componente, uma vez que ao final desta etapa necessitamos, conforme podemos ver na Figura 6.7, de um detalhamento neste sentido.

Detalhamento:

As principais fontes para a aquisição do conhecimento do domínio foram:

- Documento com modelagem dos processos legislativos (análise essencial);
- Regimento interno da CMRJ;
- Desenvolvedores de aplicações no domínio;
- Especialistas do domínio.

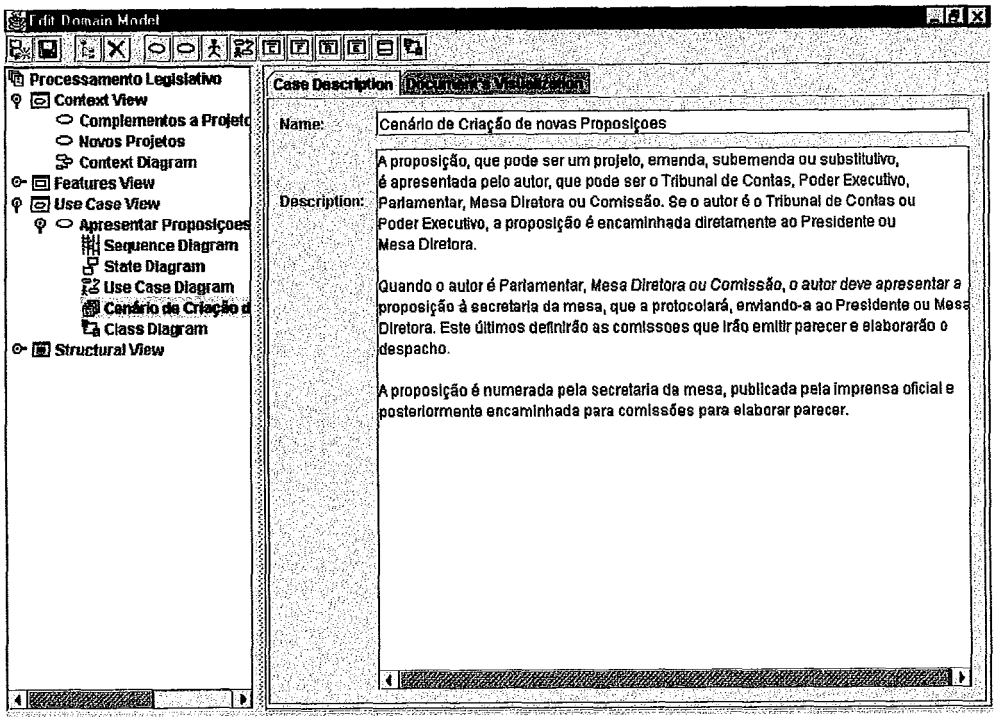

Figura 6.2– Exemplo de um cenário no domínio de Processamento Legislativo

Destas fontes, foram extraídas as descrições dos cenários, como, por exemplo, o apresentado na Figura 6.2. Ao todo foram detalhados nesta fase 17 cenários.

Fase: Representar o conhecimento do domínio

Técnica utilizada:

Modelagem do domínio, utilizando as técnicas, diretrizes e modelos especificados no contexto do processo Odyssey-DE e a infra-estrutura Odyssey.

Avaliação do tempo e resultados:

O processo de modelagem do conhecimento do domínio foi iniciado em fevereiro de 1999 e durou, considerando somente o contexto desta tese, cerca de um ano e meio. Durante este período, os modelos passaram por diversos refinamentos e evoluções, com base na aquisição de novas informações do domínio. Basicamente, o único participante desta fase de representação foi o engenheiro do domínio. No entanto, a validação dos modelos teve a participação dos especialistas e de desenvolvedores no domínio, da CMRJ e de outras casas legislativas mais esporadicamente.

Dificuldades:

- Agendamento de reuniões: o engenheiro do domínio teve dificuldades de sanar suas dúvidas com a rapidez necessária. Uma ferramenta de colaboração específica para o contexto poderia auxiliar em muito esta fase e a fase anterior. Neste sentido, o projeto Odyssey como um todo está evoluindo com o objetivo de agregar técnicas de colaboração. Nesta nova fase, o projeto será denominado Odyssey-Share;
- Falta de conhecimento do domínio: Por conta do engenheiro do domínio inicialmente não conhecer o domínio, dúvidas a respeito do vocabulário empregado, funcionalidades do domínio e principalmente o fluxo destas funcionalidades foi preponderante.

Detalhamento:

Atualmente, a modelagem do domínio de processamento legislativo utilizando o Odyssey-DE contempla aproximadamente 80% do domínio. Assim, já foram especificados e detalhados 60 características e 23 casos de uso do domínio. Como este projeto de pesquisa resultou, recentemente em um projeto de transferência de tecnologia, esta modelagem continua em evolução. Não é nosso objetivo apresentar todos os componentes especificados e seus modelos no contexto desta tese, uma vez que são muitos os modelos. Apresentamos a título de ilustração, a seqüência de criação de modelos de componentes utilizando o Odyssey-DE, incluindo o detalhamento de um componente específico do domínio (sendo este componente também utilizado para ilustrar a etapa de projeto) e uma parte do modelo de características.

A partir da descrição dos cenários, o engenheiro do domínio, através de um processo de abstração de funcionalidades e conceitos relacionados, iniciou o processo de criação dos casos de uso do domínio e do modelo de características. Apresentamos a seguir alguns diagramas relacionados a esta atividade.

Um exemplo parcial do diagrama de características para o domínio, englobando as visões de conceitos e funcionalidades, é apresentado na Figura 6.3, seguindo a notação visual proposta em (MILLER, 2000). O detalhamento da característica "Proposição", apresentada no diagrama de características (Figura 6.3) é apresentado na Figura 6.4a e 6.4b, através de seu padrão de domínio.

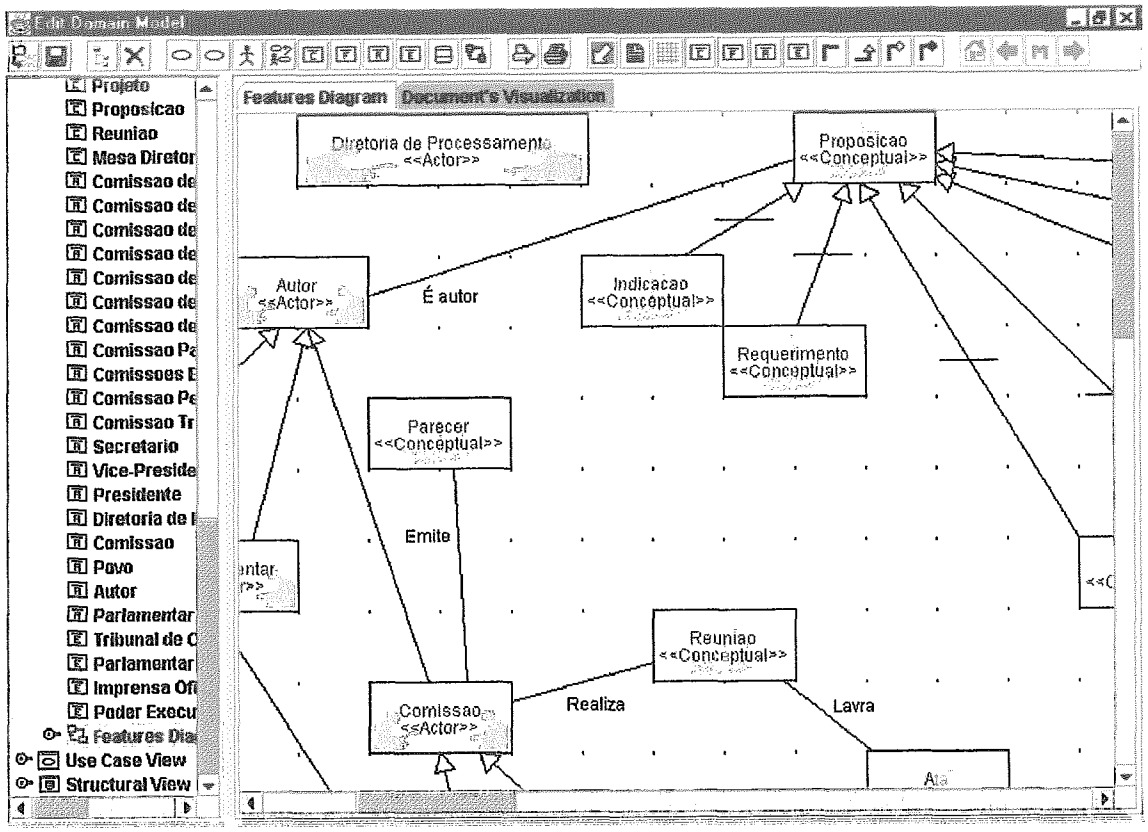

Figura 6.3 – Diagrama Parcial de Características para o domínio de processamento Legislativo

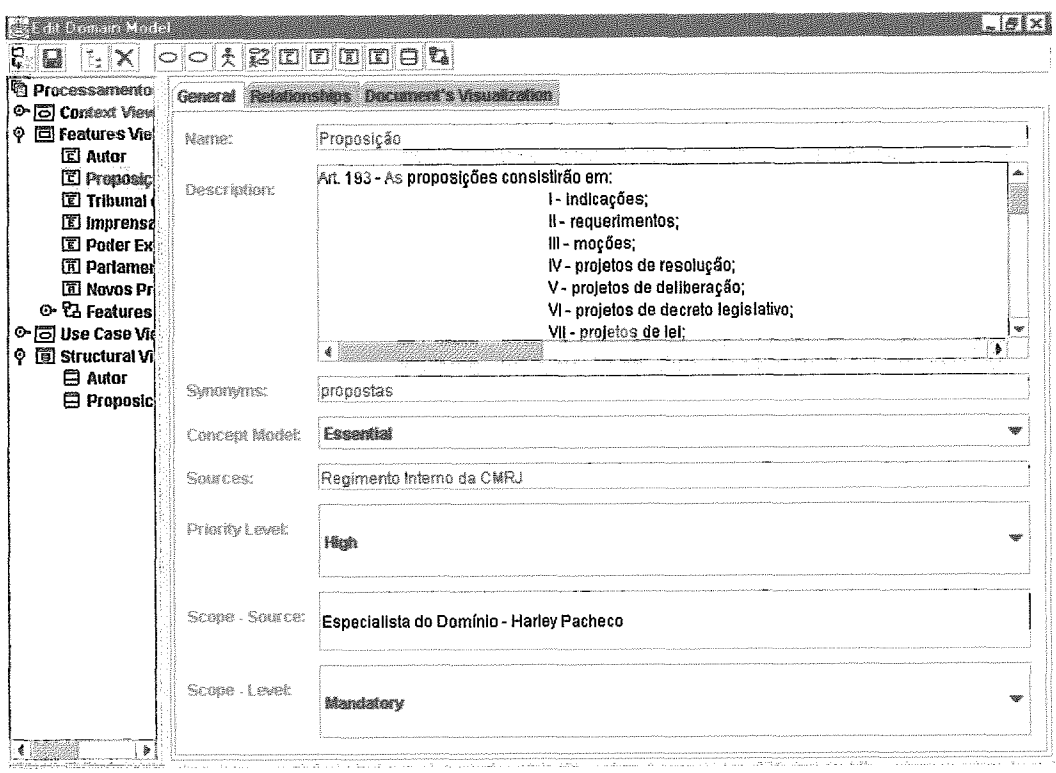

Figura 6.4a - Exemplo de instanciação de um padrão de domínio no domínio de processamento legislativo

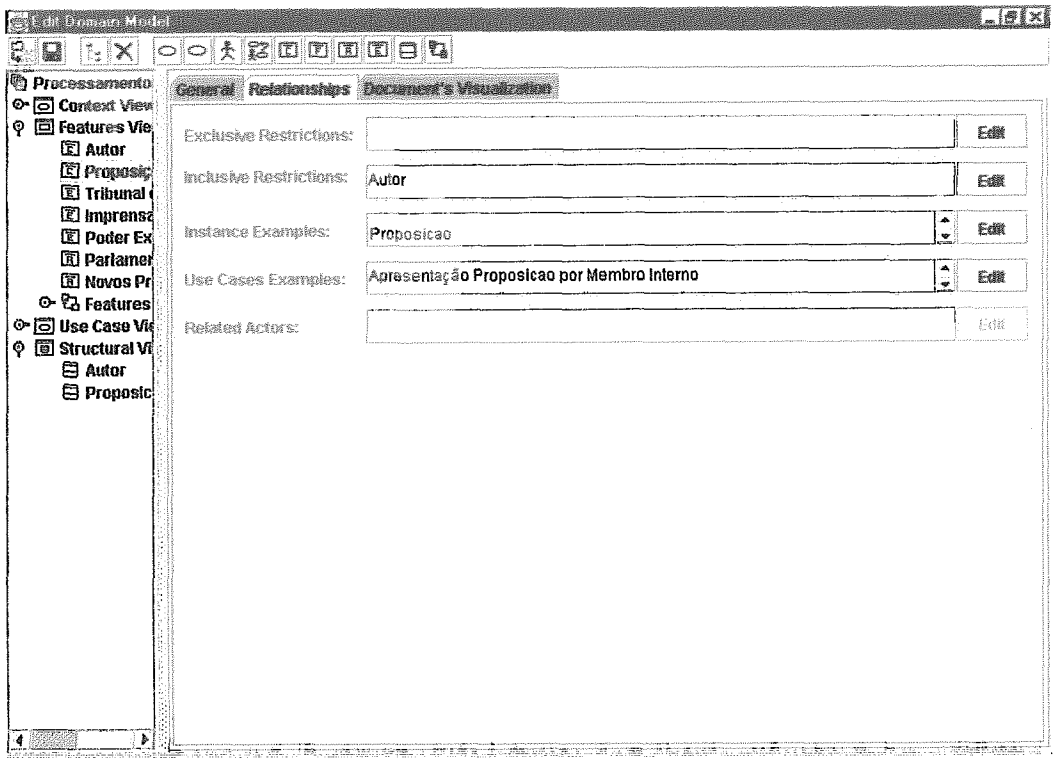

Figura 6.4b – Continuação do exemplo de instanciação de um padrão de domínio no domínio de processamento legislativo

Na Figura 6.5a e 6.5b, a instanciação de um caso de uso do domínio derivado diretamente do cenário apresentado na Figura 6.2 é apresentada.

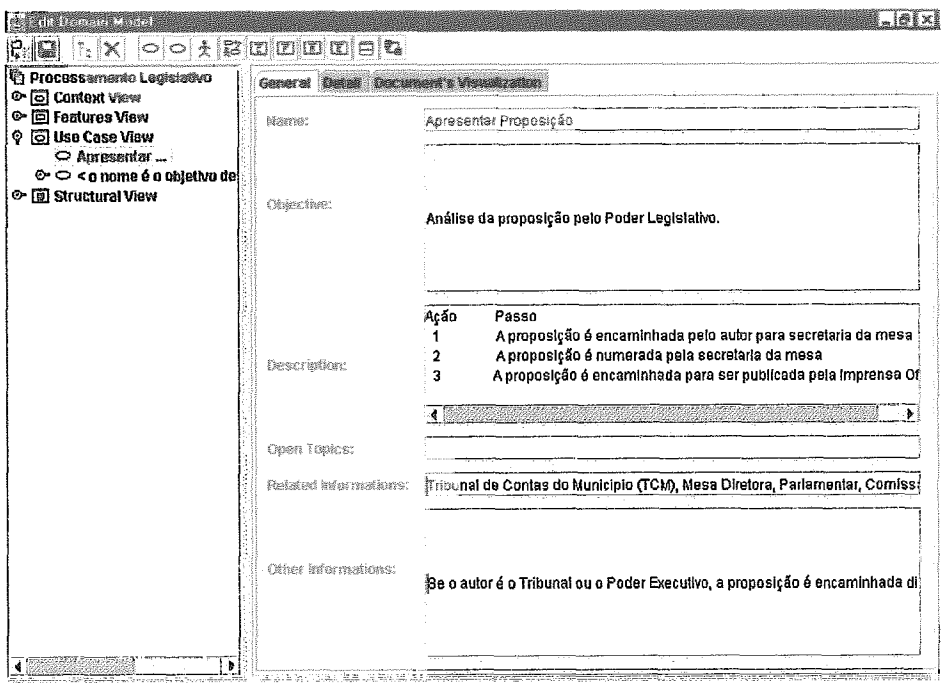

Figura 6.5a – Instanciação de um caso de uso do domínio de Processamento Legislativo

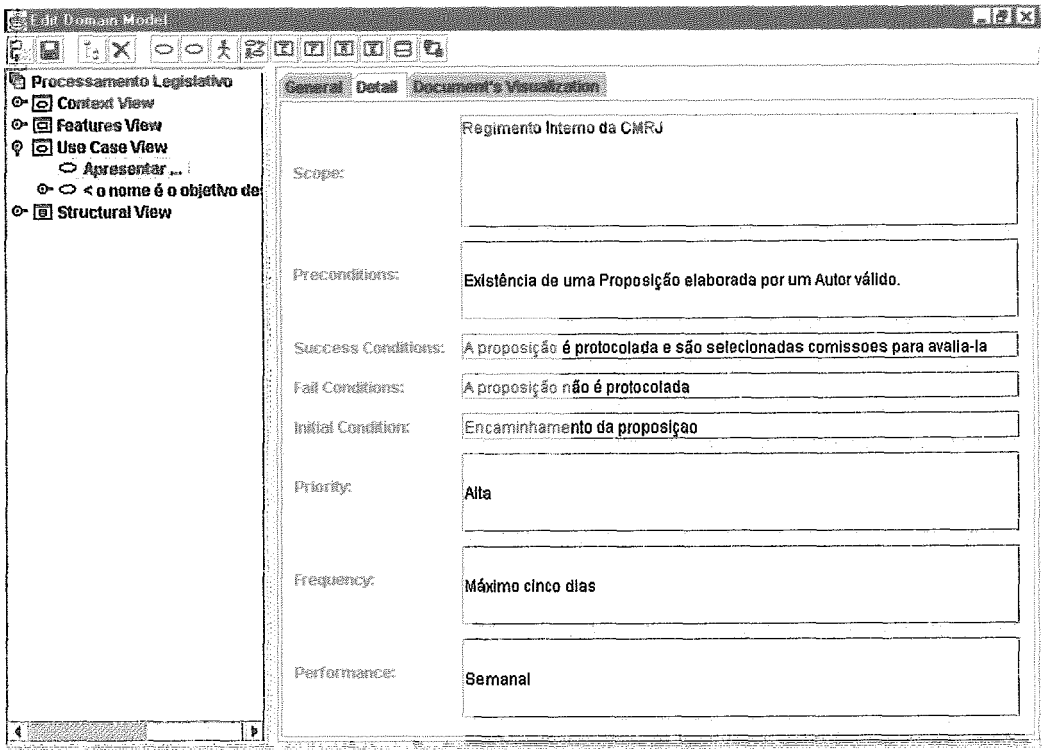

Figura 6.5b – Continuação do exemplo de instanciação de um caso de uso do domínio

Com base nos casos de uso especificados, o engenheiro do domínio identificou os relacionamentos entre os mesmos. A Figura 6.6 apresenta uma visão diagramática do caso de uso do domínio "Apresentar Proposição" com outros casos de uso relacionados.

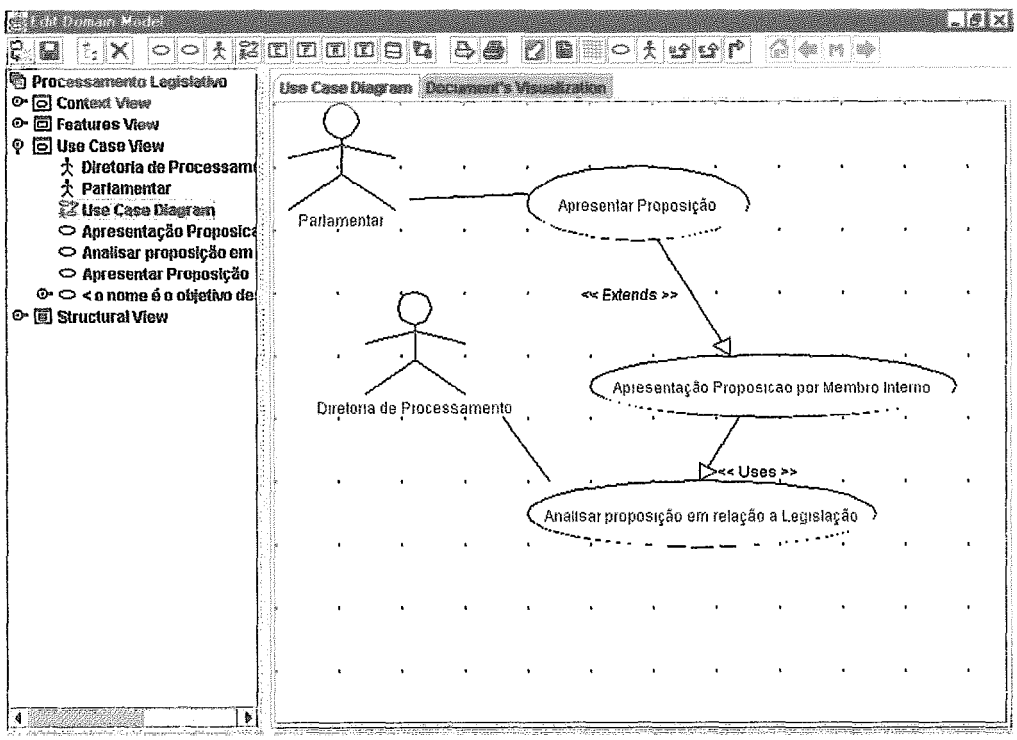

Figura 6.6 – Visão diagramática do caso de uso e seus relacionamentos

De posse da descrição dos casos de uso, partiu-se para o detalhamento de cada um, através da descrição de seus modelos internos. Cada caso de uso do domínio descreve um componente do domínio abstrato. As Figuras 6.7 e 6.8 apresentam o diagrama de classes e de seqüência com o detalhamento do caso de uso/componente do domínio "Apresentar Proposição". No detalhe da Figura 6.7, podemos notar a preocupação com a especificação da interface funcional (ainda em alto nível de abstração) com as principais funcionalidades disponibilizadas pelo caso de uso/componente do domínio conceitual.

É importante ressaltar que este foi um dos pontos onde o processo original sofreu mudanças em função do estudo de caso. Na versão original, a definição das interfaces neste nível não era contemplada. No entanto, através do estudo de caso, pudemos observar que neste nível já estão definidas as funcionalidades a serem disponibilizadas pelo componente e, portanto, um detalhamento como o apresentado na Figura 6.7 se fez necessário. No diagrama seqüencial da Figura 6.8, vale destacar que este é especificado de maneira genérica, ou seja, sem ser representado por objetos de uma aplicação do domínio em particular, como seria o caso do diagrama de seqüência original da notação UML.

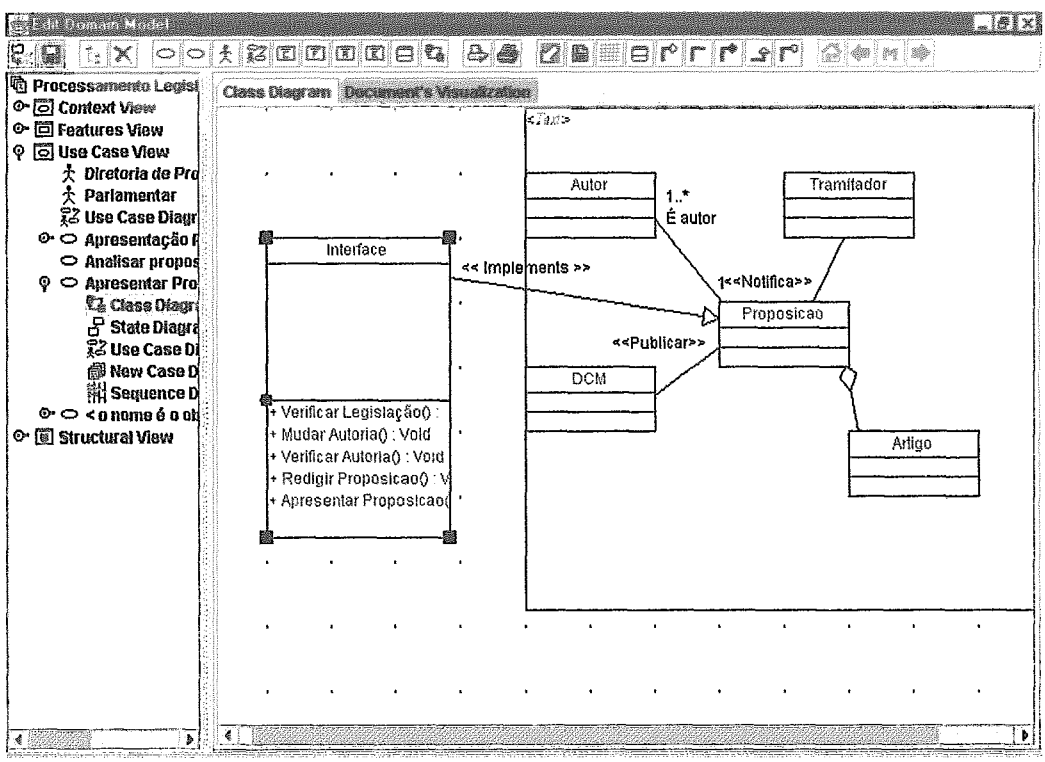

Figura 6.7 – Exemplo diagramático de modelo de colaboração do caso de uso no domínio "Apresentar Proposição"

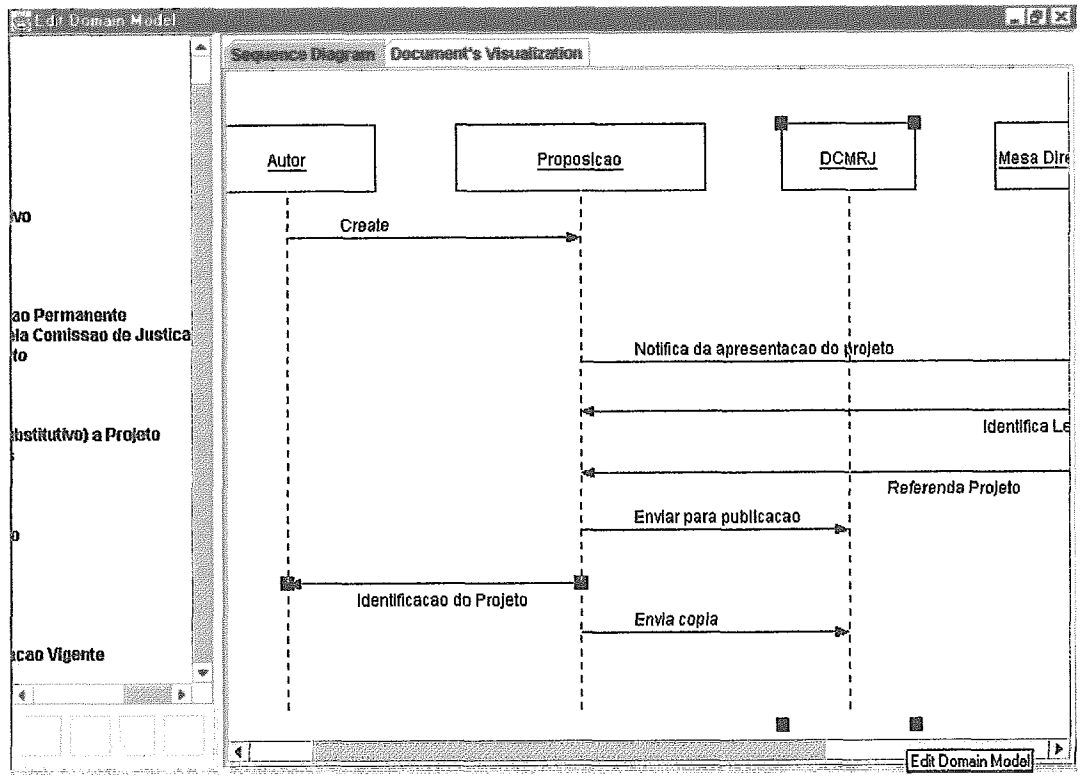

Figura 6.8 – Diagrama seqüencial para caso de uso "Apresentar Proposição"

6.2.2.3 Etapa: Projeto do Domínio

A exemplo da análise do domínio, detalhamos a seguir cada uma das fases desta etapa, respectivamente:

- Definição dos componentes e suas interfaces;
- Definição de um modelo geral de colaboração entre os componentes (modelo arquitetural de componentes), levando em consideração a utilização de componentes de suporte;
- Definição do projeto interno dos componentes.

Fase: Definição dos componentes e suas interfaces

Técnica utilizada:

Técnicas e modelos propostos pelo Odyssey-DE, utilizando a infra-estrutura Odyssey como suporte.

Avaliação do tempo e resultados:

Uma vez definidos os componentes abstratos, partiu-se para a definição das interfaces dos componentes, vislumbrando-se não só as funcionalidades disponibilizadas mas também a interação entre os componentes para a realização destas

funcionalidades (interfaces requeridas). Esta etapa, realizada pelo engenheiro do domínio e parcialmente validada por desenvolvedores de aplicações no domínio, teve duração aproximada de 2 meses (40 horas). Esta validação por parte dos desenvolvedores do domínio foi realizada através de correio eletrônico.

Dificuldades:

Em termos do projeto de transferência de tecnologia, esta fase ainda não foi cumprida. No entanto, no contexto desta tese, alguns componentes e seus relacionamentos foram detalhados. A maior dificuldade encontrada foi a necessidade do detalhamento dos tipos que compõem o componente. Com isso, a fase de projeto interno do componente ocorreu em paralelo a esta. Outra dificuldade encontrada foi a falta de suporte da infra-estrutura Odyssey para esta fase específica. Em relação a fase de análise, conforme podemos visualizar pelas descrições da fase anterior (fase de análise), a infra-estrutura Odyssey já provê suporte completo. No entanto, para a fase de projeto foram necessárias algumas adaptações nos diagramas disponíveis para a representação dos componentes nesta etapa. Desta forma, o diagrama apresentado na Figura 6.9 é uma tentativa de modelar os componentes nesta fase com os atuais recursos disponíveis. No entanto, não corresponde a situação ideal. Trabalhos nesta direção já estão em andamento (XAVIER, 2000), (WERNER et al., 2000).

Detalhamento:

Um exemplo pontual do modelo desta fase, para dois componentes no domínio de processamento legislativo, é apresentado na Figura 6.9.

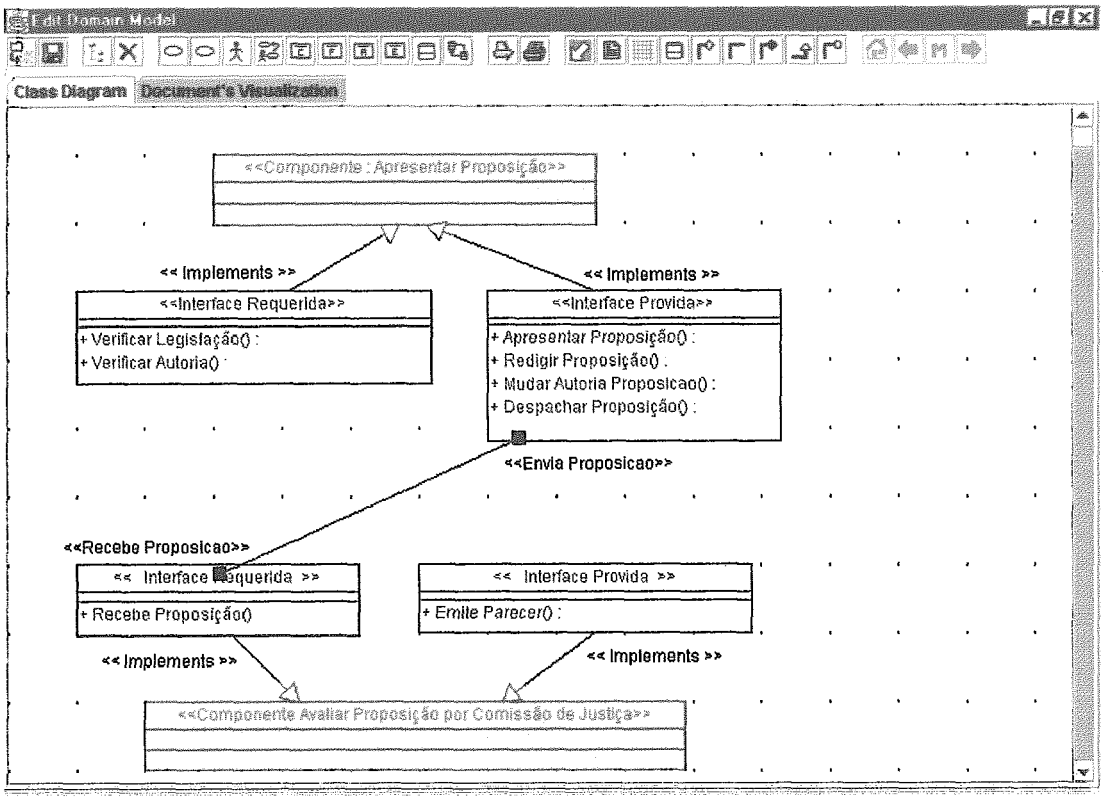

Figura 6.9 – Exemplo diagramático do Modelo de Serviços dos Componentes.

Fase: Definição de um modelo arquitetural de colaboração entre os componentes

Técnica utilizada:

De acordo com as diretrizes do processo Odyssey-DE e consulta a padrões arquiteturais de BUSHMMAN (1996).

Avaliação do tempo e resultados:

Devido ao projeto de transferência de tecnologia não ter ainda atingido esta fase, um estudo empírico, considerando-se os padrões arquiteturais disponíveis, foi realizado para o contexto desta tese. Este estudo durou 5 horas e contou com o auxílio de um especialista em arquitetura de software, mas não especialista no domínio e da literatura disponível sobre arquitetura de software. Como resultado, foi escolhido um determinado estilo arquitetural que foi considerado condizente com as aplicações relacionadas ao domínio.

Dificuldades:

Como o projeto de transferência de tecnologia ainda não está nesta fase, o estudo realizado foi baseado em suposições do engenheiro do domínio e do especialista em arquitetura de software. Ficou evidenciada a falta que o auxílio de um desenvolvedor de aplicações no domínio faz nesta fase. No entanto, pouco adiantará se este desenvolvedor não tiver um conhecimento dos diferentes estilos arquiteturais existentes, suas características, vantagens e desvantagens. Se este for o caso, um treinamento pode ser necessário. Outro ponto de dificuldade foi a falta de suporte adequado da infra-estrutura Odyssey para esta fase.

Detalhamento:

Apesar da decisão de qual estilo arquitetural utilizar ser mais ligada à aplicação em si do que ao domínio, conforme ressaltamos no capítulo 3, existem alguns domínios onde o tipo de interação entre os componentes observados através das aplicações já desenvolvidas podem nos levar a privilegiar um dado estilo arquitetural como sendo o mais recomendado àquele domínio. Mesmo assim, podem existir aplicações que não obedeçam necessariamente este dado estilo arquitetural. No caso do domínio de processamento legislativo, pela própria interação entre os componentes identificados, podemos identificar que o estilo arquitetural que melhor caracterizaria o domínio seria uma variação do estilo arquitetural "dutos e filtros" (*pipe and filters)*, (BUSHMMAN, pp. 53-70, 1996) levando-se também em consideração características de fluxo de trabalho (*workflow)*. O fluxo de informações flui entre os componentes do processo legislativo de uma maneira ordenada. Assim, neste caso específico, teríamos os diversos componentes organizados de modo que a tramitação das proposições fluíssem entre os mesmos.

Fase: Definição do projeto interno dos componentes

Técnica utilizada:

Definição de atributos e métodos das classes que compõem o componente, utilizando os recursos da infra-estrutura Odyssey.

Avaliação do tempo e resultados:

Novamente, como o projeto de transferência de tecnologia ainda não está nesta fase, foram escolhidos alguns componentes e detalhados seus tipos. Esta fase, que foi realizada concomitantemente com a fase de definição das interfaces dos componentes, teve duração aproximada de 30 horas e contou com o suporte da infra-estrutura Odyssey para este detalhamento.

Dificuldades:

Embora previsto o uso de padrões de projeto para apoiar esta atividade, este recurso ainda não se encontra automatizado.

Detalhamento:

O diagrama de classes do componente "Apresentar Proposição", com detalhamento de atributos e métodos é apresentado na Figura 6.10.

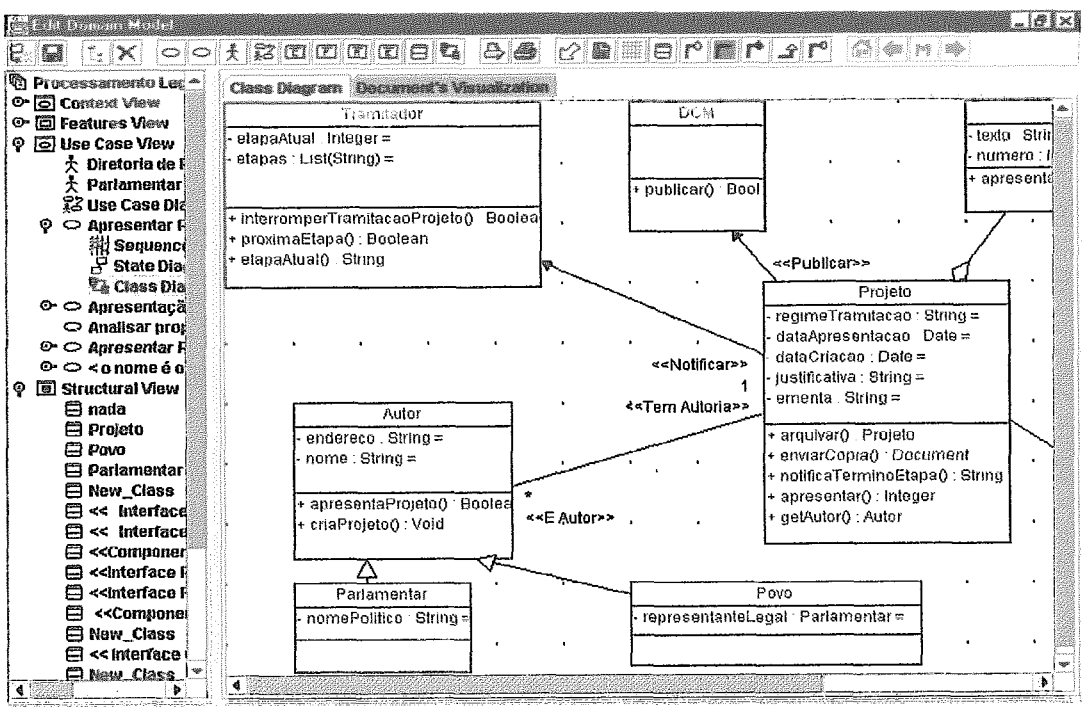

Figura 6.10 – Exemplo diagramático do Modelo Interno do componente Apresentar Proposição

6.2.2.4 Etapa: Implementação do Domínio

Esta etapa não foi contemplada no projeto de transferência de tecnologia. No entanto, algumas considerações podem ser feitas, levando-se em conta as etapas anteriores. A estratégia de codificação dos componentes em uma linguagem de programação OO é mais adequada de ser realizada na engenharia de aplicações e não na engenharia de domínio, uma vez que os modelos podem sofrer alterações por conta do estilo arquitetural escolhido e estas alterações podem ser refletidas na codificação final do componente. No entanto, nada impede que esta codificação seja feita na engenharia de domínio. Neste caso uma adaptação terá que ser realizada no componente para que este se encaixe no dado estilo arquitetural.

A infra-estrutura Odyssey conta com suporte automatizado para esta etapa. No entanto, devido à ênfase em desenvolvimento de componentes, a codificação deve ser gerada levando-se em consideração as interfaces do componente, as classes internas do mesmo e o relacionamento entre estas. Atualmente, esta codificação é feita por classe e não por componente, como apresentado na Figura 6.11.

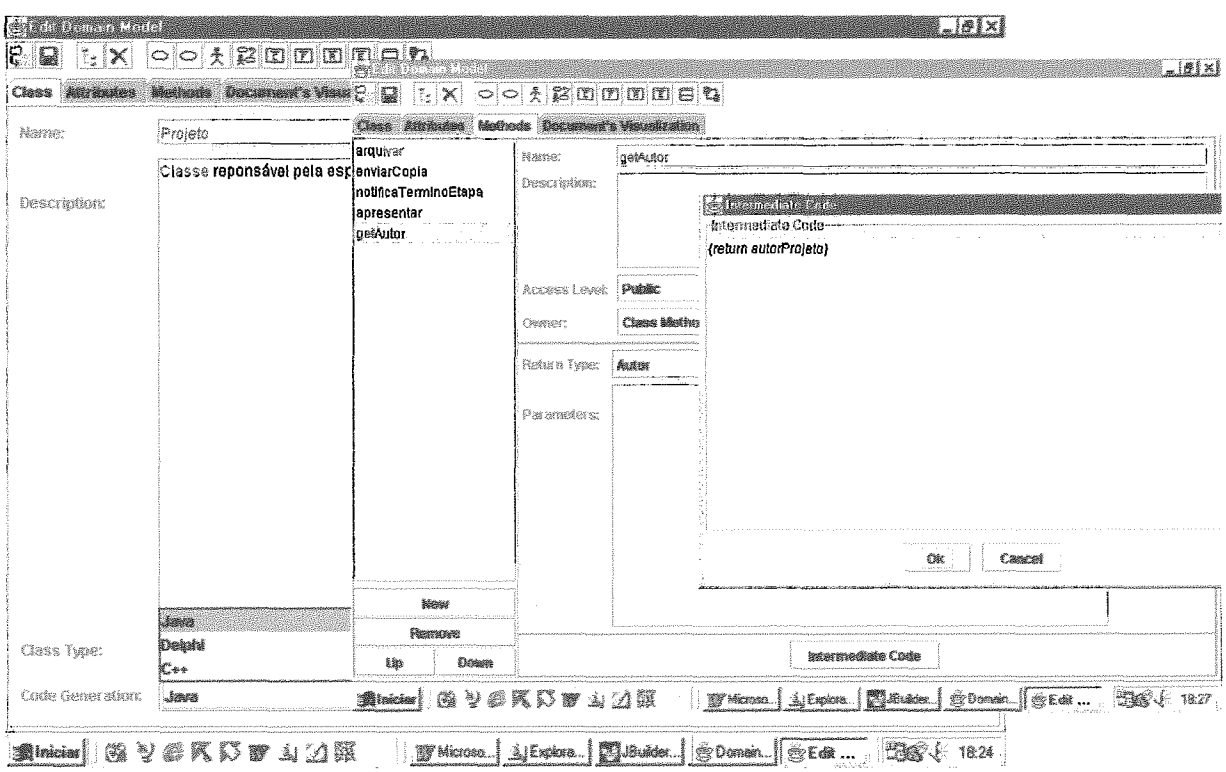

Figura 6.11 – Codificação semi-automática de componentes no domínio de Processamento Legislativo.

6.2.3 Uso dos modelos especificados no domínio para adaptação/evolução do Sistema de protocolo de emendas ao orçamento

Esta seção objetiva demonstrar a aplicabilidade dos modelos especificados através da utilização do processo Odyssey-DE no domínio de processamento legislativo. Para isso, apresentamos um exemplo de aplicação real desenvolvida no domínio e como esta aplicação pode reutilizar os modelos previamente especificados. Com isso, supomos apresentar um exemplo, mesmo que simplificado, da aplicabilidade dos componentes produzidos através do processo Odyssey-DE.

O sistema desenvolvido engloba uma pequena parte do processamento legislativo, tratando especificamente do protocolo de emendas, subemendas e substitutivos a um dado projeto. Mais especificamente, o aplicativo em questão trata do protocolo de emendas ao projeto do orçamento municipal plurianual e anual. O processamento legislativo do projeto de lei orçamentária como um todo possui particularidades em relação ao processamento de outros projetos, principalmente em relação às comissões por onde passa, prazos estipulados, entre outros.

6.2.3.1 Sistema de protocolo de emendas ao projeto orçamentário

Conforme apresentamos na subseção 6.2.1, a tramitação do projeto orçamentário é composta de diversas fases. O protocolo de emendas, subemendas e substitutivos corresponde a uma das fases do processo e pode ser utilizada em dois momentos da tramitação: em primeira discussão e, se for o caso, em segunda discussão.

Apesar de ser uma aplicação relativamente simples, ela tem a vantagem de ser uma aplicação real, já utilizada na tramitação de projetos orçamentários na **Câmara Municipal do Rio de Janeiro (CMRJ)**, e, devido a seu escopo pequeno, é entendida pelo leitor não familiarizado com o domínio.

Assim, a aplicação tem como objetivo principal permitir o protocolo eletrônico de emendas, sua consulta e modificação. Os parlamentares, comissões, lideranças e bancadas podem criar emendas e depois as protocolar, de acordo com o prazo liberado pela Comissão de Finanças e Secretaria da Mesa. O sistema deve obedecer estes prazos. Este protocolo de emendas é realizado pela rede de computadores da CMRJ.

De posse destas informações, descrevemos a seguir, passo a passo, como um engenheiro de aplicação poderia reutilizar os modelos de componentes criados utilizando-se o processo Odyssey-DE no domínio de processamento legislativo.

6.2.3.2 Utilização dos modelos de componentes do domínio de processamento legislativo

O engenheiro da aplicação, a partir de agora denominado desenvolvedor, objetivando desenvolver a aplicação de protocolo de emendas, deve, em um primeiro momento, navegar pelos diagramas que representam os modelos de componentes do domínio para que possa identificar quais modelos seriam adequados para o desenvolvimento de sua aplicação. Para facilitar o processo, o desenvolvedor pode contar com o auxílio da ferramenta Odyssey-Search, que o ajudará a encontrar os modelos mais adequados para o desenvolvimento de sua aplicação.

Para iniciar o processo de busca por componentes do domínio, o desenvolvedor seleciona o domínio mais relacionado à aplicação que deseja especificar. Um questionário (Figura 6.12) é apresentado, o qual o desenvolvedor deve preencher de forma que a ferramenta Odyssey-Search possa atuar de maneira adequada. Dentre as informações fornecidas, o desenvolvedor preenche seu nível de conhecimento do domínio (médio), seu objetivo (desenvolvimento de aplicações), aplicações do domínio já desenvolvidas utilizando o processo e que tenham relacionamento com a aplicação a ser desenvolvida (neste caso, ainda nenhuma), sub-domínios relacionados (neste caso, o desenvolvedor selecionou o sub-domínio de complementos a projetos) e palavras-chave relacionadas à aplicação (no caso, emenda, substitutivo, subemenda e protocolo).

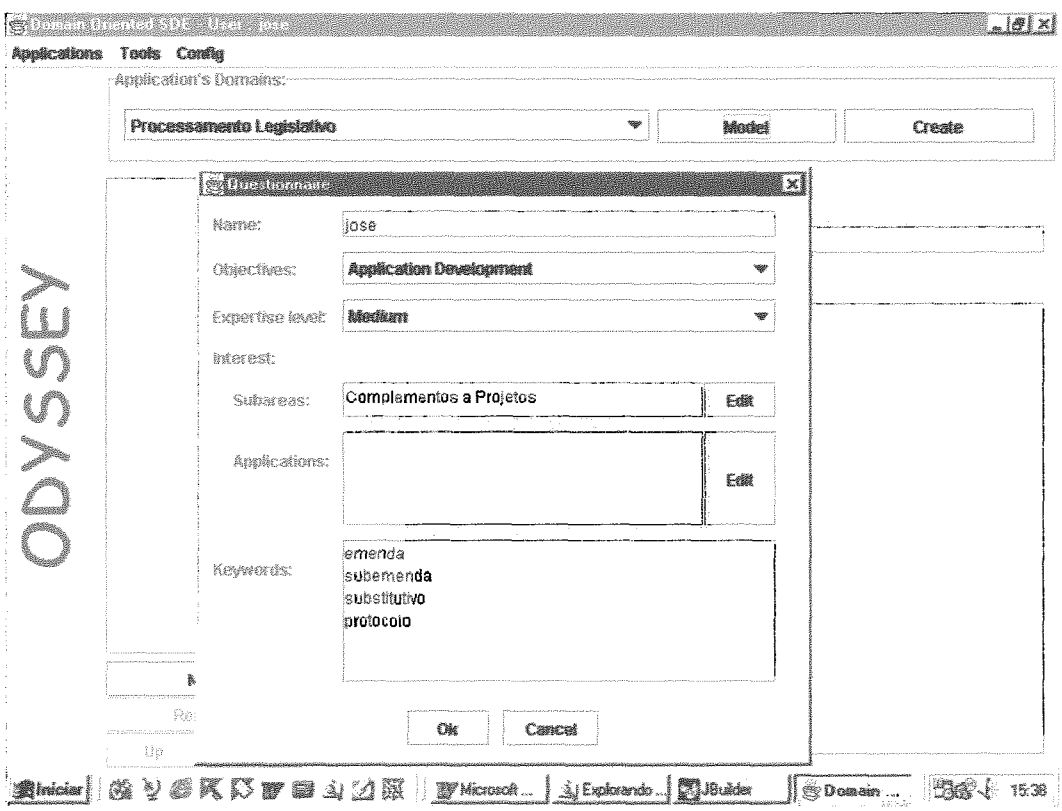

Figura 6.12 – Questionário inicial para navegação pelos modelos de componentes

Preenchido o questionário, são apresentados diagramas com todos os modelos de componentes do domínio disponíveis e o desenvolvedor pode selecioná-los e consultá-los com o auxílio da Odyssey-Search. Assim, o usuário, partindo do sub-domínio ou sub-domínios associados a sua aplicação, é guiado na seleção dos conceitos e funcionalidades mais fortemente relacionados ao seu desenvolvimento. A Figura 6.13 apresenta a seleção do sub-domínio "complementos a projetos" e a opção de navegação, pelos itens relacionados. Com esta navegação inicial, o desenvolvedor é levado a uma visão diagramática do modelo abstrato do domínio (modelo de características), onde os conceitos e funcionalidades relevantes do domínio, relacionados ao sub-domínio selecionado, são apresentados (Figura 6.14). Estes conceitos e funcionalidades (termos do domínio) são classificados de acordo com a sua relevância para o usuário, levando-se em consideração as opções selecionadas no questionário inicial.

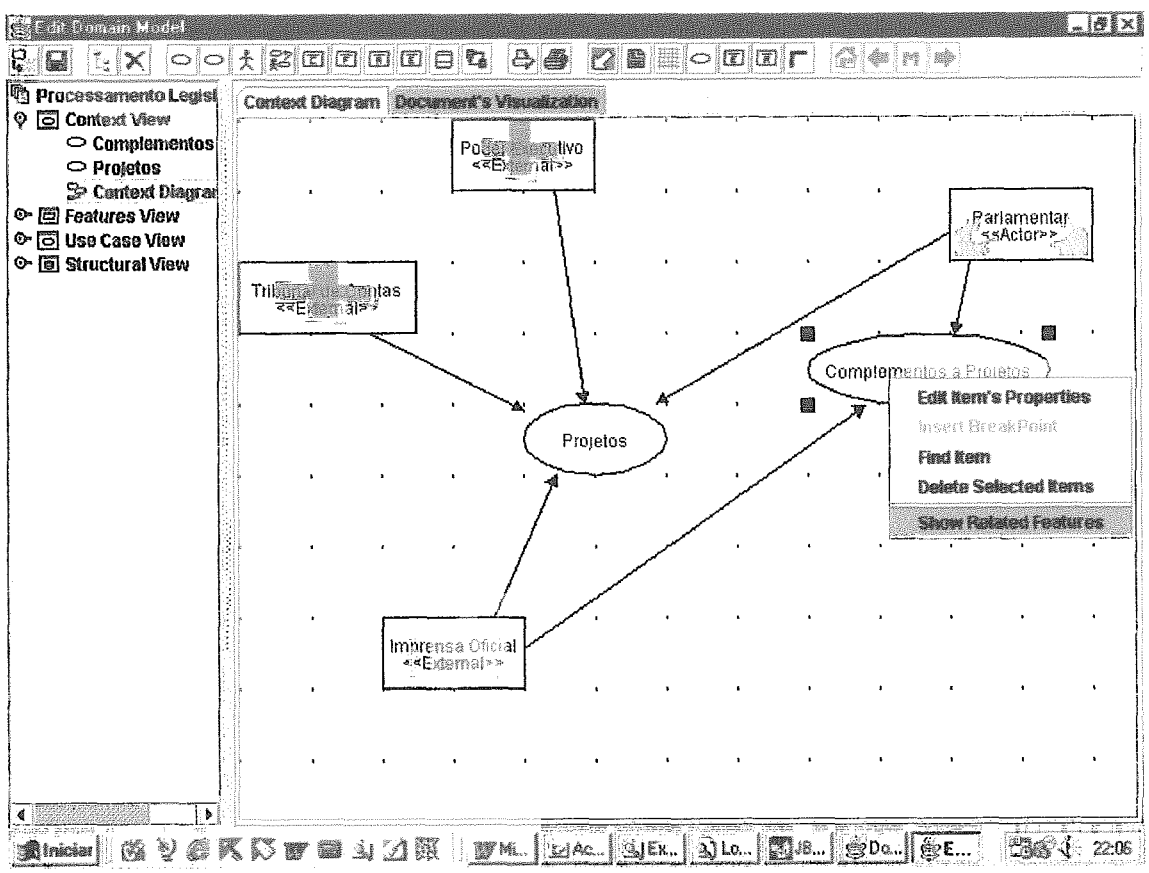

Figura 6.13 – Seleção de subdomínios relacionados para navegação

O desenvolvedor pode continuar sua navegação, examinando, se for o caso, as descrições associadas a cada termo do modelo abstrato, ou navegando para os casos de

uso do domínio relacionados, o que o leva as visões diagramáticas dos modelos conceituais dos componentes do domínio. Esta navegação para os casos de uso relacionados apresenta, segundo as seleções de navegação do usuário até o momento e as opções selecionadas no questionário, quais são os casos de uso (componentes conceituais) mais relevantes para a aplicação a ser desenvolvida, classificando-os pelo seu nível de importância (Figura 6.15). O usuário pode então examinar a descrição textual dos casos de uso (Figura 6.16), diagramas de classes (Figura 6.17), diagramas de seqüência (Figura 6.18) e estados.

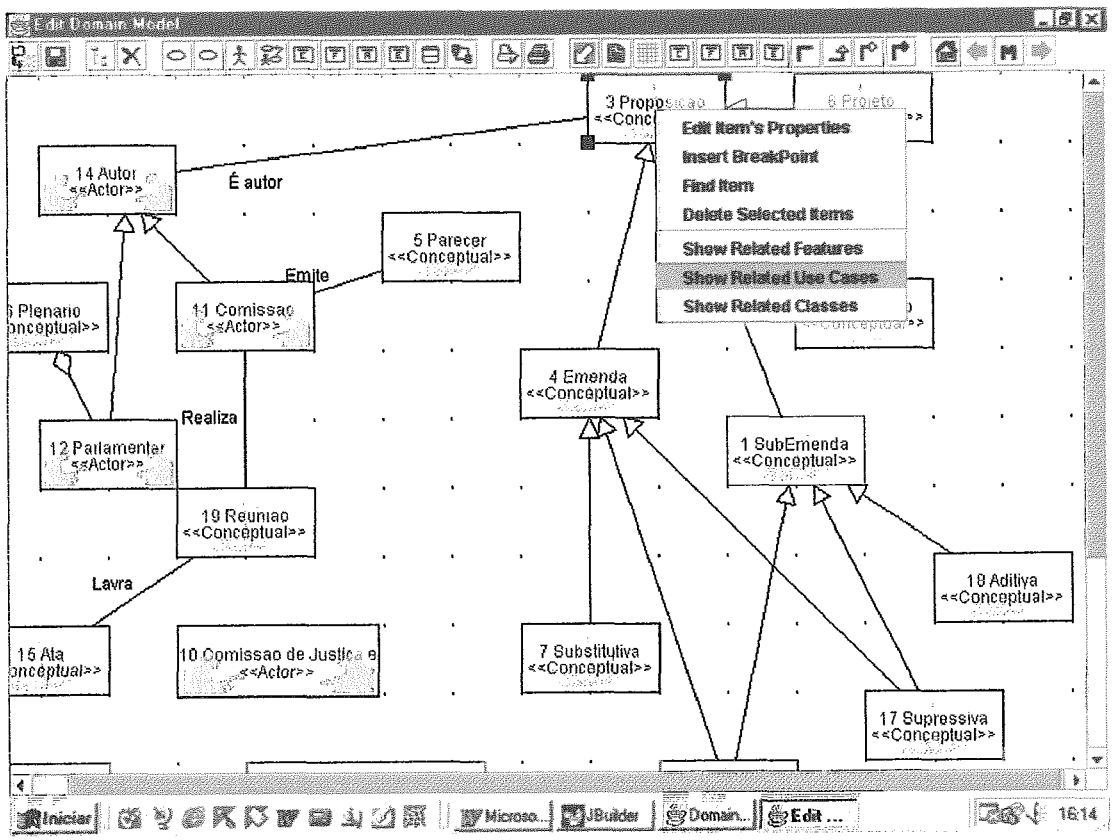

Figura 6.14 – Termos do domínio relacionados e classificados por ordem de relevância

Neste momento, o desenvolvedor tem a possibilidade de examinar quais são os componentes, de acordo com as funcionalidades dos casos de uso apresentados, que são mais adequados para a aplicação. No caso específico da aplicação de protocolos de emendas, o desenvolvedor, com a ajuda da ferramenta Odyssey-Search, identificou que os casos de uso "Propõe complemento" (Figuras 6.17 e 6.18) e "Análise de Complemento por comissão" seriam os mais relacionados à aplicação. É importante observar que neste momento estamos tentando identificar os componentes e seus

modelos em nível conceitual. Considerações a respeito da arquitetura da aplicação, linguagem de codificação, etc., são feitas durante o desenvolvimento da aplicação, pelas razões explicitadas no capítulo 4.

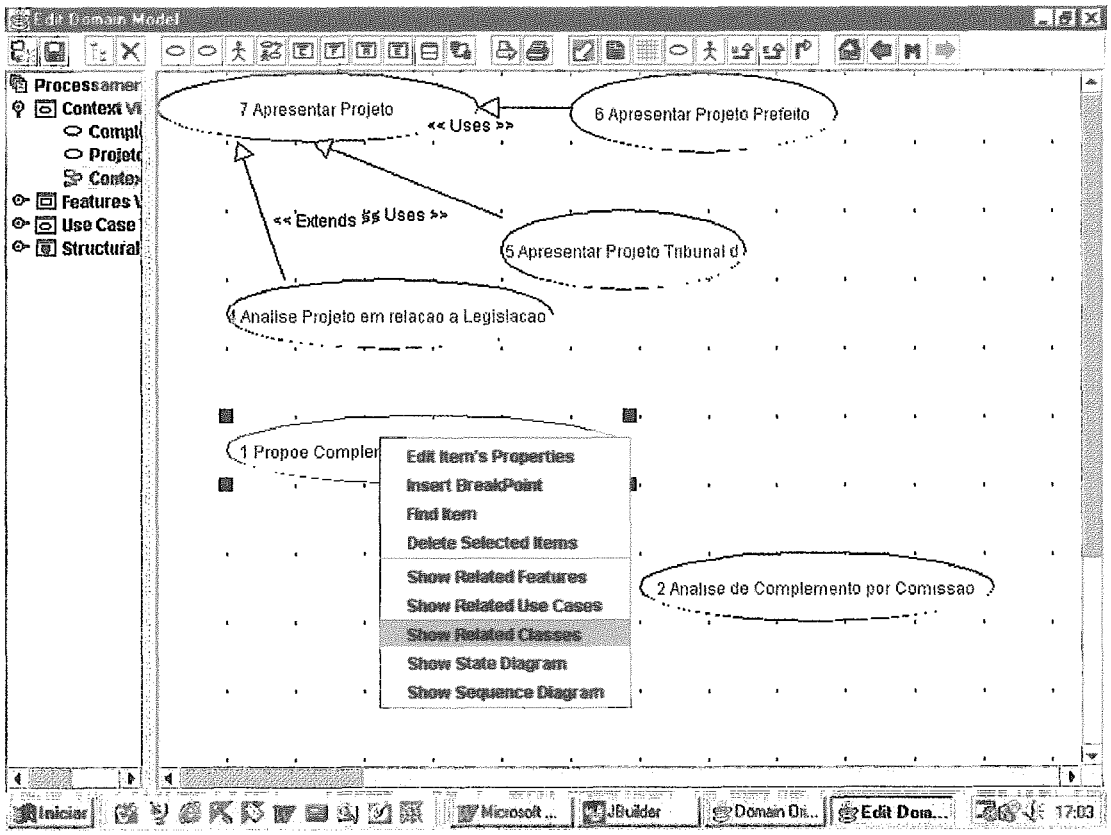

Figura 6.15 – Sugestão de modelos conceituais de componentes para a aplicação protocolo de emendas

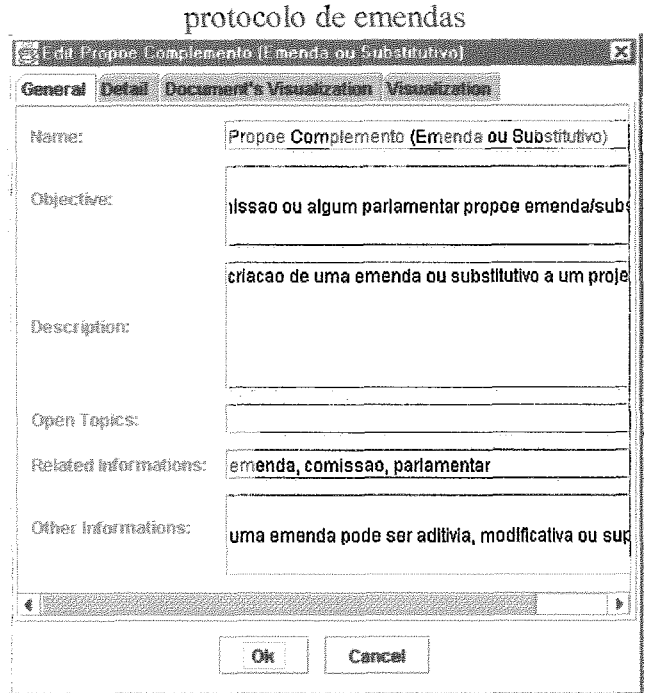

Figura 6.16 a - Detalhamento textual de um caso de uso do domínio

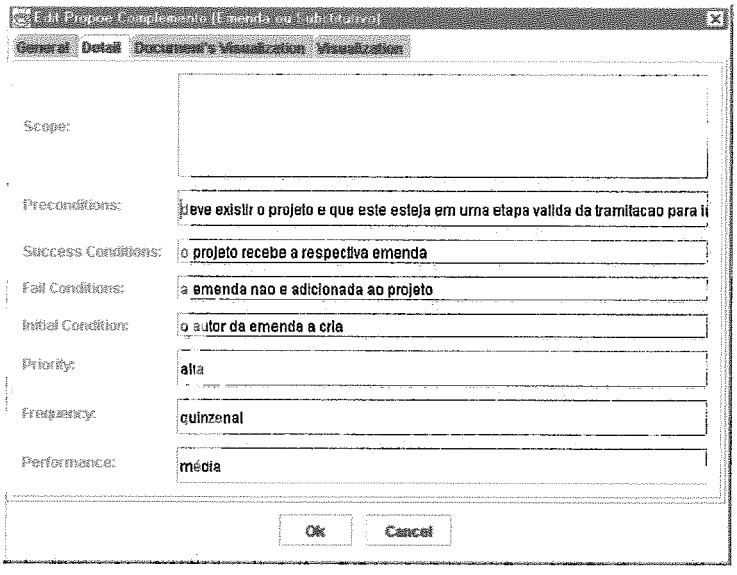

Figura 6.16 b – Continuação do detalhamento textual de um caso de uso do domínio

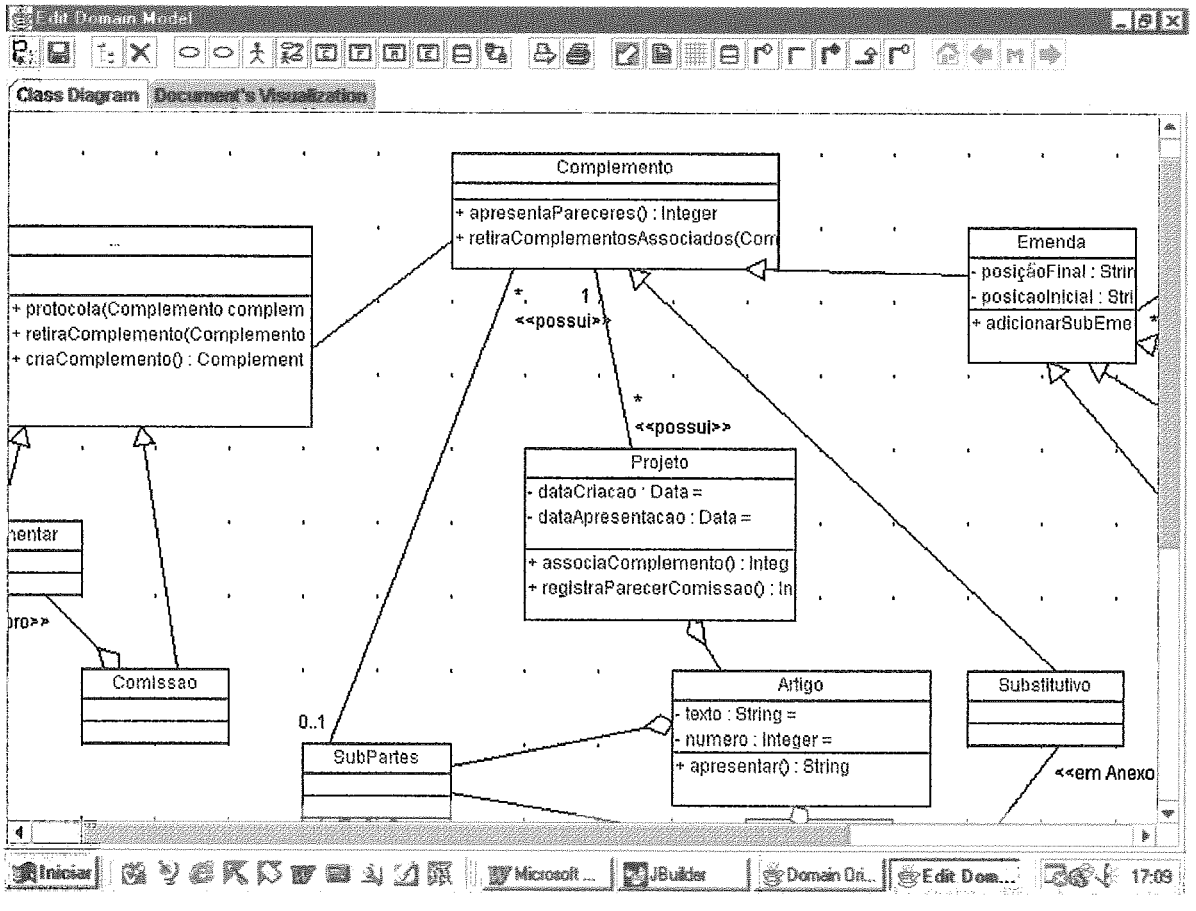

Figura 6.17 – Visão diagramática do Modelo de Classes relacionado ao caso de uso "Propõe Complemento"

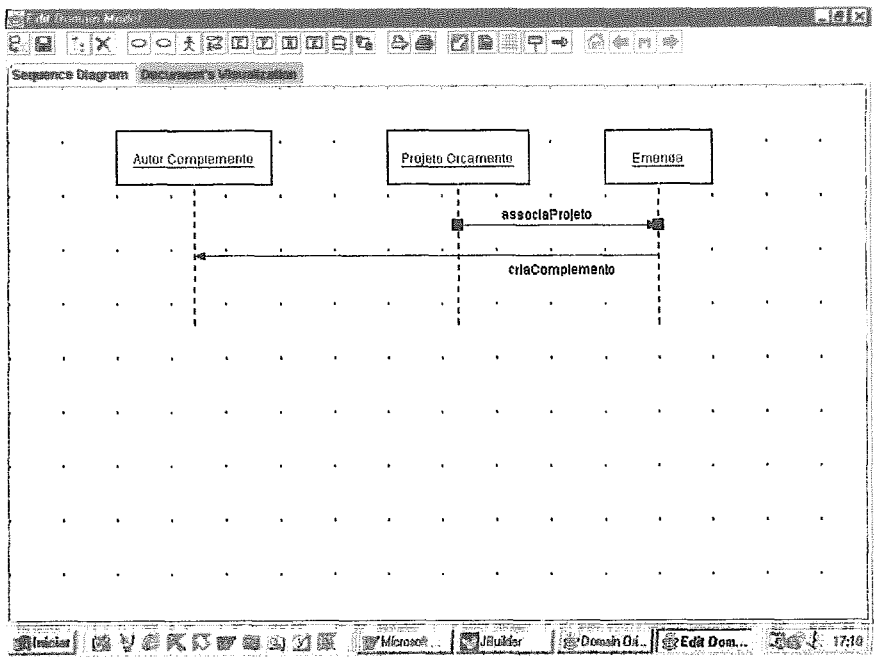

Figura 6.18 – Diagrama de Seqüência relacionado ao caso de uso "Propõe Complemento"

Identificados os componentes conceituais do domínio mais adequados para o desenvolvimento da aplicação, o desenvolvedor parte para a especificação da aplicação, através do processo de engenharia de aplicações, Odyssey-EA, especificado em MILLER (2000). Neste caso específico, o usuário cria a aplicação, denominada protocolo de emendas. Quando da criação da aplicação, é questionado se o desenvolvedor deseja reutilizar modelos de componentes do domínio relacionado (Figura 6.19 a e b). Assim, o usuário seleciona o sub-domínio "Complementos a projetos" (Figura 6.20) e todos os termos relacionados são disponibilizados para o desenvolvedor. De acordo com a navegação realizada, o desenvolvedor seleciona os termos identificados previamente como adequados a sua aplicação (Figura 6.21).

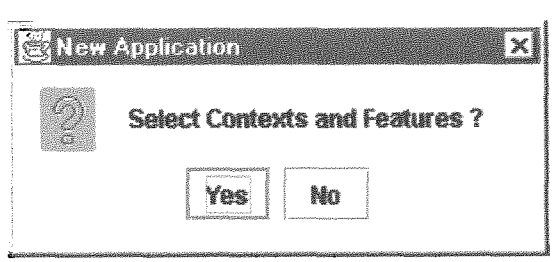

Figura 6.19 a – Seleção de componentes do domínio

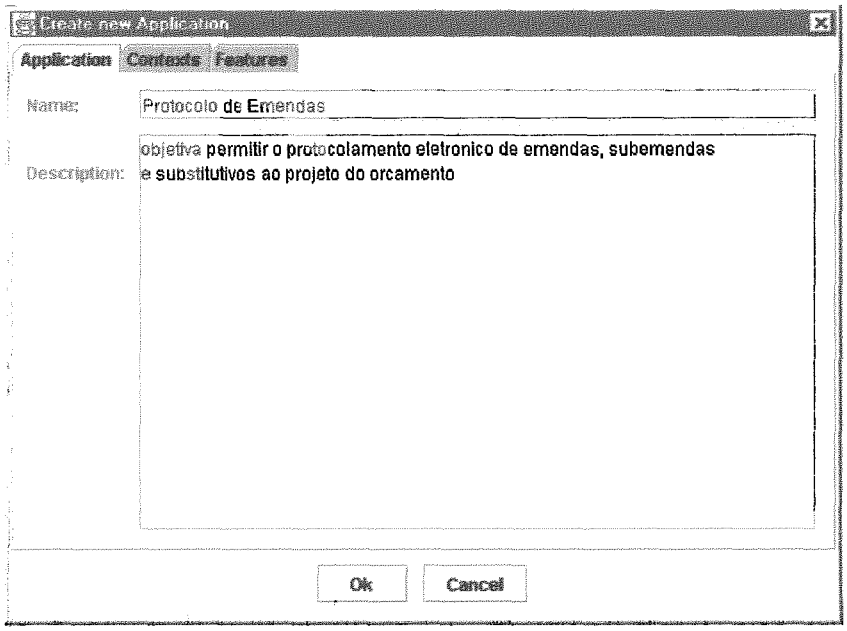

Figura 6.19 b – Criação da aplicação "Protocolo de Emendas" reutilizando sub-domínios e termos do domínio relacionados.

Desta forma, como podemos visualizar na Figura 6.22, todos os componentes relacionados aos termos escolhidos pelo desenvolvedor são disponibilizados. Cabe ao desenvolvedor excluir da aplicação aqueles componentes cujas funcionalidades não vão ser contempladas na aplicação. No caso específico da aplicação de protocolo de emendas, somente o caso de uso do domínio "Propõe complemento" será contemplado. Portanto, os outros casos de uso devem ser descartados. Além disso, como a aplicação refere-se ao protocolo de emendas a projetos do orçamento, o desenvolvedor deve adicionar algumas classes aos modelos já existentes, de forma a contemplar as especificidades do orçamento e da aplicação.

Uma questão importante diz respeito aos componentes de suporte, tais como componentes para interfaceamento com banco de dados, componentes de interface com usuário, entre outros. Componentes deste tipo devem ser agregados aos modelos da aplicação, seguindo o modelo arquitetural desejado. No caso específico da aplicação de protocolo de emendas, o modelo arquitetural escolhido é o Modelo-Visão-Controle (*Model-View-Controller)* (BUSCHMANN, 1996) e desta forma, a interação entre os componentes deve seguir esta abordagem. No caso específico dos componentes do domínio, estes são componentes "do negócio" e, portanto, estariam relacionados a *Model.*

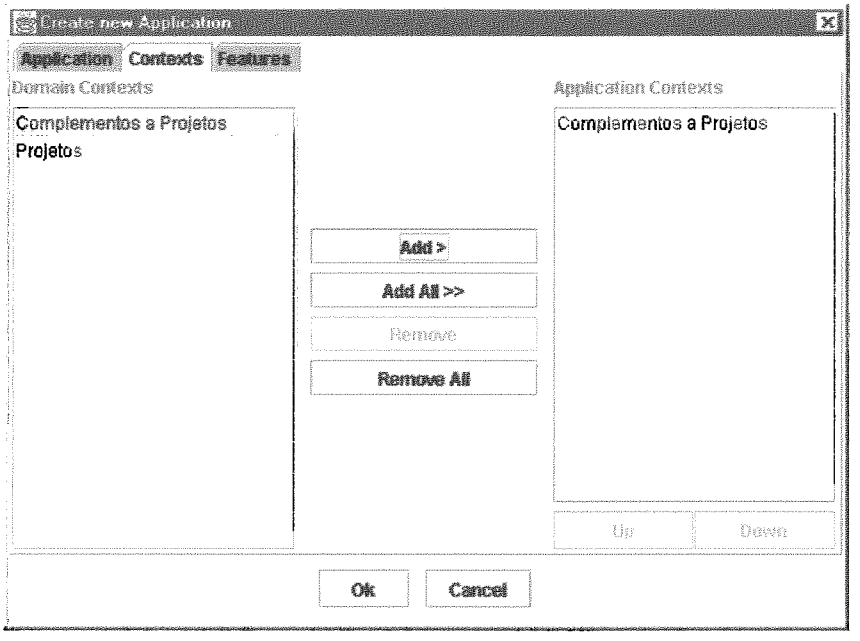

Figura 6.20 – Seleção do subdomínio "Complementos a projetos" como sub-domínio mais relacionado à aplicação.

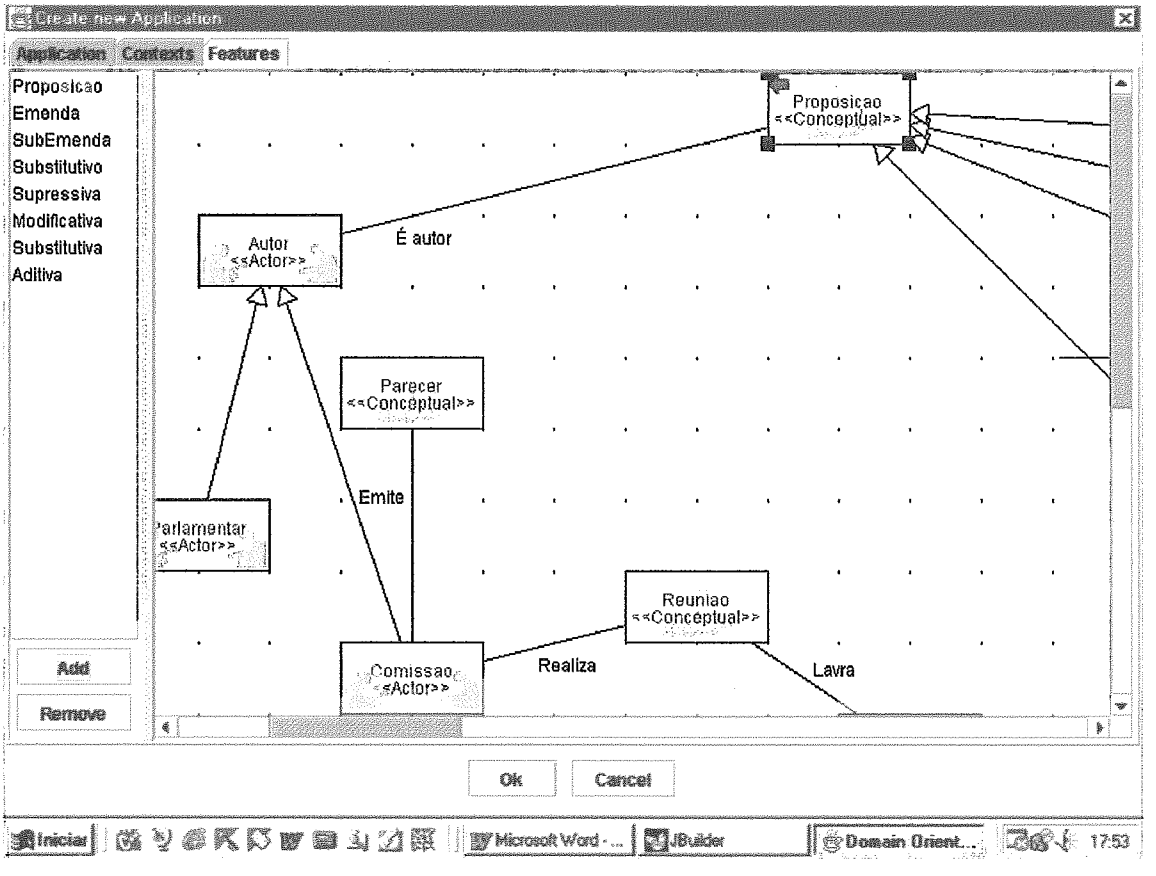

Figura 6.21 – Seleção dos termos do domínio mais importantes para a aplicação.

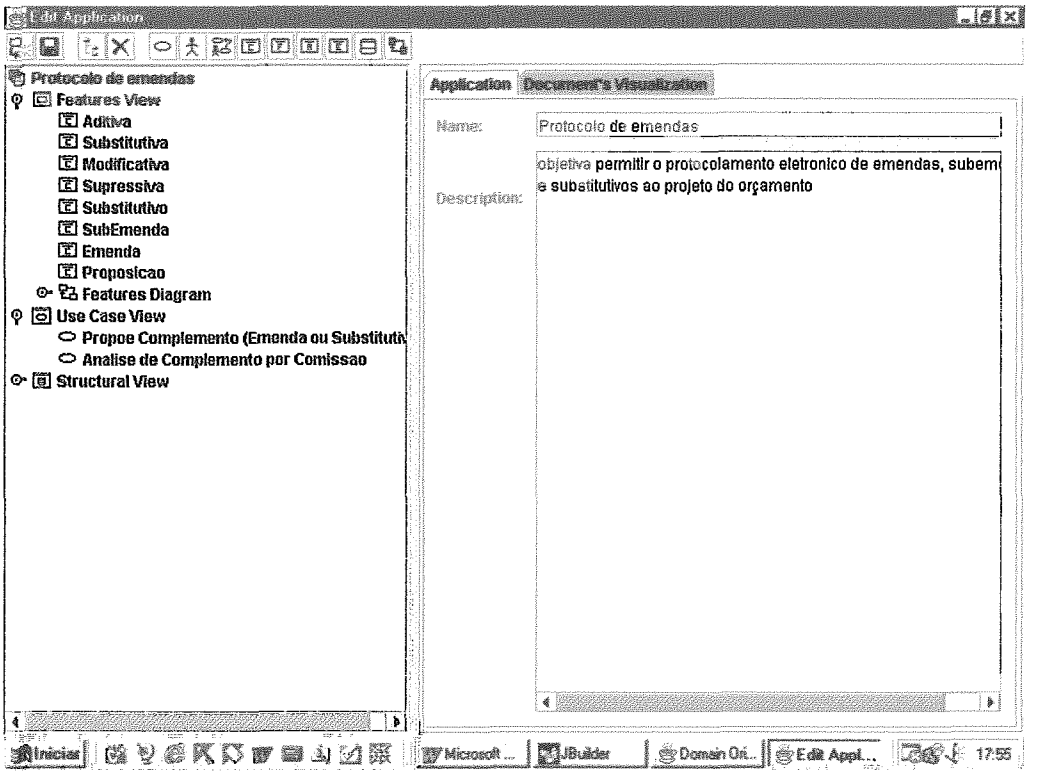

Figura 6.22 – Modelos de Componentes reutilizados pela aplicação Protocolo de Emendas

Conforme ressaltado no início deste capítulo, a aplicação de protocolo de emendas é uma aplicação real da CMRJ. Confrontando os modelos originais da aplicação com os modelos reutilizados, podemos notar que os modelos reutilizados do Odyssey refletem de maneira adequada as funcionalidades necessárias. Vale ainda a pena ressaltar que um dos objetivos da especificação de componentes reutilizáveis no domínio é desenvolver novas aplicações no domínio mas também facilitar a evolução das já existentes. No caso específico da aplicação de protocolo de emendas, atualmente esta somente contempla o protocolo de emendas e não o acompanhamento posterior destas nas comissões que irão realizar a análise. Assim, uma evolução cabível e interessante da aplicação seria também contemplar esta análise pelas comissões. Com a utilização do Odyssey-DE, um componente específico para esta análise poderia ser reutilizado, componente este que já existe disponível na infra-estrutura Odyssey. Assim, o engenheiro de aplicação teria que se concentrar nas adaptações necessárias para que o componente se adequasse à arquitetura da aplicação e interagisse com os demais componentes.

6.2.4 Avaliação dos resultados do estudo de caso

Ao longo de um ano e meio de estudo de caso no domínio de processamento legislativo, pudemos comprovar o quão útil foi o uso do processo em um domínio real, uma vez que pudemos vislumbrar eventuais inconsistências e propor melhorias neste sentido. No entanto, nenhum resultado conclusivo em relação à eficácia do processo como um todo pôde ser ressaltado, uma vez que estudos em outros domínios seriam necessários para realizar esta comprovação.

Desta forma, com este estudo de caso, a necessidade de algumas mudanças nas etapas do Odyssey-DE foi ressaltada. Algumas delas são:

- O Domínio de processamento legislativo pode ser considerado um domínio de porte médio e como tal gerou um conjunto considerável de características (60 até o momento). A visualização e o entendimento das mesmas fica prejudicado se estas etapas não forem divididas por sub-domínios. Esta divisão por sub-domínios não foi proposta no processo Odyssey-DE original, mas devido à necessidade ressaltada pelo estudo de caso, já está contemplada na nova versão do processo, apresentada no capítulo 4;
- Pudemos comprovar a importância da captura das funcionalidades do domínio e sua correspondência quase que direta com os casos de uso do domínio. Esta constatação mostra que o direcionamento do Odyssey-DE, voltado para a captura de funcionalidades, é adequado para o desenvolvimento de componentes;
- Com o detalhamento dos modelos e análise dos sistemas existentes, pudemos também observar que a especificação de um modelo arquitetural é muito mais dependente da aplicação do que do domínio em si. Assim, na engenharia de domínio, deve-se no máximo ter uma sugestão do melhor estilo arquitetural, mas a modelagem efetiva da arquitetura deve ser feita na engenharia de aplicação;
- Devido à complexidade do processo como um todo, ficou clara a necessidade de um ferramental adequado para dar suporte ao processo. Neste sentido, a infra-estrutura Odyssey se mostrou adequada até a fase de análise. No entanto, ainda existem alguns pontos em aberto, principalmente em relação às fases de projeto e implementação, tais como:
    - O apoio de diagramas específicos para o detalhamento das interfaces providas e requeridas pelos componentes, além da geração de código a partir do componente como um todo, incluindo suas interfaces;

- Em relação à infra-estrutura Odyssey, ficou evidente a necessidade do suporte ao controle do processo pela própria infra-estrutura. Sem este controle, o desenvolvimento dos componentes torna-se ad-hoc e o engenheiro do domínio fica perdido no controle das visões dos componentes em diversos níveis de abstração;
- Na etapa de projeto do domínio, as fases de definição dos componentes e definição do projeto interno do componente se mostraram, na prática, como concomitantes, o que denotou a necessidade de serem executadas em paralelo. Assim, pela constatação de que a modelagem arquitetural dos componentes está mais ligada à implementação da aplicação, pudemos notar também a necessidade de evolução das etapas de projeto e implementação no processo de engenharia de aplicações Odyssey-EA. Uma abordagem neste sentido já está sendo especificada (XAVIER, 2000).

Como resultado final relativo a este estudo de caso, devemos ressaltar que este serviu de base para experimentarmos as técnicas e idéias embutidas no Odyssey-DE e termos uma primeira visão a respeito de seus resultados, seus principais benefícios e possíveis evoluções. No entanto, para comprovar a eficácia do Odyssey-DE, o projeto de transferência de tecnologia estabelecido com a CMRJ deve ser finalizado, e outras avaliações em outros domínios devem poder ser realizadas. Neste sentido, iniciamos um projeto em parceria com a Universidade Federal de Juiz de Fora, Embrapa e Núcleo de Qualidade Softex – Juiz de Fora, para a aplicação do Odyssey-DE no domínio agropecuário. Os primeiros resultados do projeto, basicamente a criação de um modelo de abstrações do domínio com o objetivo de classificar aplicativos já existentes, podem ser encontrados em (CAMPOS *et al.*, 2000). Este estudo de caso, apesar de não fazer parte do escopo desta tese, pode ser considerado como uma continuação dos trabalhos aqui desenvolvidos.

6.3 *Utilização da ferramenta Odyssey-Search para busca, compreensão e recuperação de componentes reutilizáveis do domínio*

O objetivo deste segundo estudo de caso é observar a utilização da ferramenta Odyssey-Search na busca, compreensão e recuperação de componentes reutilizáveis do domínio. Em um segundo momento, objetivamos avaliar até que ponto a ferramenta incentiva a reutilização de componentes no desenvolvimento de uma dada aplicação.

Podemos dividir da seguinte maneira as hipóteses a serem observadas com o estudo realizado:

1. **A utilização da ferramenta Odyssey-Search auxilia na compreensão do domínio;**
2. **A utilização da ferramenta Odyssey-Search sugere componentes realmente úteis para o desenvolvimento de aplicações no domínio.**

Com o objetivo de discutir pontos específicos relativos a estas duas hipóteses, foi estruturado um questionário de avaliação cujo objetivo, além de verificar as duas hipóteses levantadas acima, foi também avaliar algumas questões a respeito do funcionamento da ferramenta de forma geral. Assim, o questionário foi estruturado com o objetivo de responder as seguintes questões:

1. **A ferramenta auxiliou na obtenção de conhecimento sobre o domínio em questão?**
2. **As sugestões dadas pela ferramenta em relação às informações do domínio são realmente úteis?**
3. **A interface com o usuário da ferramenta é adequada ?**
4. **A ferramenta traz benefícios se comparada com outras ferramentas de busca gerais e ferramentas de busca de componentes?**

Algumas considerações que devem ser feitas em relação a estas questões é que as respostas às três primeiras questões podem ser influenciadas pelo conhecimento prévio do domínio e também pelo conhecimento de técnicas para modelagem de informações, uma vez que as informações são apresentadas utilizando diagramas baseados na representação UML. Assim, este item também é avaliado através das respostas do questionário.

5. **Conhecimento prévio do domínio e de técnicas de modelagem de informação.**

Apresentamos a seguir a estruturação do questionário direcionada para avaliar as questões 1, 2, 3, 4 e 5.

Conhecimento do Domínio e de técnicas de modelagem de informação

A primeira parte do questionário teve como objetivo levantar o perfil do avaliador em relação a sua experiência na área de informática e também em relação ao

domínio modelado. O fato do avaliador ter formação na área de informática é um indício de que já tenha tido contatos com técnicas de modelagem de informação e de que também tem o conhecimento necessário para avaliar a adequabilidade da interface e execução da ferramenta. O nível de conhecimento do domínio é também outro fator que pode influenciar na avaliação dos resultados, uma vez que o conhecimento sobre o domínio pode implicar em uma avaliação menos positiva em relação ao grau de ajuda da ferramenta na compreensão do domínio. Para avaliar este item, as seguintes questões do questionário foram utilizadas:

Formação: a) Informática b) Área tecnológica c) Área Humana d) Outra:

Atuação: a) Professor b) Pesquisador c) Aluno de Pós-Graduação d) Aluno de Graduação e) Outro:

Área com maior experiência a) Gerência b) Especificação c) Projeto d) Programação e) Outra:

Nível de conhecimento do domínio legislativo. a) Alto b) Médio c) Baixo

Auxílio na compreensão do domínio e desenvolvimento de aplicações no mesmo

Esta parte do questionário teve como objetivo levantar até que ponto a ferramenta auxiliou no grau de compreensão do domínio e de desenvolvimento de aplicações do domínio. É importante ressaltar que, apesar de uma questão do questionário ter como objetivo avaliar explicitamente o auxílio no desenvolvimento de aplicações do domínio, esta avaliação é subjetiva uma vez que não foi requerido dos avaliadores que os mesmos desenvolvessem aplicações no domínio. Este aspecto específico não pôde ser avaliado em função da atual infra-estrutura ainda não estar totalmente desenvolvida para dar este tipo de suporte. Assim, consideramos que sem o desenvolvimento efetivo de uma aplicação, está análise específica fica prejudicada. As questões relativas à avaliação desta parte são as seguintes:

Se o nível de conhecimento do domínio é Médio ou Baixo, em que grau o agente ajudou a melhorar seu nível de conhecimento a respeito do domínio?
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

O Sr (a) como desenvolvedor de software acredita que este tipo de agente auxilia no processo de desenvolvimento de aplicações no domínio de uma forma geral.
a) Sim b) Não

Comentários:

Adequabilidade das sugestões do agente

O principal objetivo desta análise é observar, de acordo com o preenchimento do questionário sobre o perfil do usuário, se as interações e indicações do agente foram adequadas. Com esta avaliação, supomos conseguir estimar até que ponto o usuário sente confiança nas sugestões do agente e utilizaria estas sugestões no desenvolvimento de aplicações do domínio. As questões relativas a este aspecto são:

Facilidade para preenchimento do questionário, ou seja, se os itens do questionário são fáceis e intuitivos de serem selecionados.
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

O formato de apresentação das informações do domínio.
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

Adequabilidade das telas de alerta do agente, em relação a serem intuitivas e fáceis de serem percebidas
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

Interações do agente durante a navegação
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

Como o agente atendeu suas expectativas, no sentido de apresentar caminhos de navegação que levaram a informações úteis?
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

Adequabilidade da interface do usuário

Esta parte do questionário visou avaliar até que ponto a interface do usuário utilizada foi adequada para realizar a busca pelos termos. Neste sentido, cabe aqui avaliar menus e opções de navegação utilizadas, i.e., facilidades para a identificação das opções de funcionamento da ferramenta.

Layout do questionário de montagem do perfil do usuário.
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

Facilidade de identificação e entendimento dos ícones e opções do menu *pop-up* para navegação.
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

O tipo de indicação do agente (borda vermelha e numeração) para apresentar a melhor solução.
a) Ótimo b) Bom c) Regular d) Ruim

Comentários:

Adequabilidade da ferramenta se comparada com outras ferramentas de busca

As questões deste grupo de perguntas visavam comparar a ferramenta com outras propostas no mercado e/ou literatura, com o objetivo de observar pontos fracos e fortes percebidos pelos avaliadores.

O Sr (a) já tinha utilizado algum tipo de agente de busca anteriormente?
a) Sim b) Não

Qual?

Qual?

Se na questão acima respondeu sim, como avaliaria a *Odyssey-Search* em relação a este agente?

a) melhor b) igual c) inferior

Comentários:

Você conhecia algum agente de busca de componentes reutilizáveis?

a) Sim b) Não

Qual?

Se na questão acima respondeu sim, como avaliaria a *Odyssey-Search* em relação a este agente?

a) melhor b) igual c) inferior

Comentários:

Descrevemos a seguir as três etapas do estudo de caso , seguindo também as atividades definidas em (KITCHNHAN *et al.*, 1995).

6.3.1 Planejamento

O estudo de caso foi definido e planejado em setembro de 2000. O universo a ser avaliado seria o de desenvolvedores do domínio de aplicação, especialistas deste domínio e desenvolvedores sem nenhum conhecimento a respeito do domínio. Assim, pela facilidade de contatos feitos através da universidade, ficou definido que os participantes seriam alunos de graduação, que representam o universo de desenvolvedores sem conhecimento prévio do domínio, e desenvolvedores e especialistas do domínio de processos legislativos, que é o domínio onde o processo Odyssey-DE foi aplicado (estudo de caso 1), representando assim o universo de desenvolvedores com algum conhecimento do domínio.

Assim, foi elaborado um questionário estruturado conforme descrito anteriormente, de forma a observar a utilização da Odyssey-Search por parte dos avaliadores, segundo as hipóteses apresentadas acima. Este questionário somente seria disponibilizado após a utilização da ferramenta. Com isso, tentamos evitar que a

utilização da ferramenta fosse conduzida de acordo com as respostas a serem preenchidas no questionário.

Após um período inicial de acerto do número de participantes da avaliação (esta etapa é detalhada na fase de monitoração), ficou acertada a participação de 6 alunos de graduação e 2 analistas de sistemas do domínio, totalizando 8 avaliadores. Todos estes com formação em informática e maior experiência na área de programação, com exceção de um analista cuja maior experiência é em levantamento de requisitos. Vale a pena também ressaltar que nenhum dos participantes teve contato anterior com a ferramenta.

Consideramos este conjunto de avaliadores, neste primeiro momento de avaliação da ferramenta, adequado para os nossos propósitos uma vez que todos possuem experiência em desenvolvimento de software, ou seja, são potenciais reutilizadores de software, que é exatamente o público para o qual a ferramenta foi inicialmente projetada. Além disso, contamos com um certo número de avaliadores sem nenhum conhecimento do domínio, o que para os nossos propósitos também é adequado, uma vez que uma das hipóteses a ser avaliada é a de aumento no nível de conhecimento do domínio. Por outro lado, podemos também ter uma avaliação, mesmo que inicial, de desenvolvedores com certa experiência no domínio, uma vez que contamos também com representantes deste universo.

Como a ferramenta atuará somente sobre os modelos de componentes gerados na fase de análise, denominamos estes componentes, no contexto deste estudo de caso, de informações do domínio.

6.3.2 Monitoração

Técnica Utilizada:

Execução e utilização da ferramenta, uso de um questionário estruturado, uso de correio eletrônico.

Detalhamento:

Em relação à escolha dos alunos de graduação, inicialmente foi feita uma explanação geral a respeito da ferramenta Odyssey-Search e do domínio de processos legislativos para estes alunos, durante uma aula da disciplina Engenharia de Software. Esta explanação foi realizada em meados do mês de setembro e durou cerca de 15

minutos. Ao final da explanação, os alunos puderam formular perguntas e voluntariamente se disporem a avaliar a ferramenta.

Alguns alunos mostraram interesse imediato e ficou acertado o contato, via correio eletrônico, para que fosse realizada uma outra apresentação mais detalhada da ferramenta e disponibilizado o protótipo para instalação e utilização. Na primeira semana do mês de outubro, estes contatos forma realizados e formaram-se 3 grupos de 3 pessoas para participar de sessões de apresentação da ferramenta. Cada sessão durou cerca de 1 hora e meia e foram realizadas ao todo 3 sessões. Ao final, os alunos levaram uma cópia da ferramenta e de parte dos modelos do domínio de processos legislativos. Ficou acertado que teriam um prazo de 3 semanas para avaliar a ferramenta e ao final seria disponibilizado o questionário de avaliação. Durante este período, eventuais dúvidas poderiam ser sanadas por correio eletrônico ou pessoalmente.

Em relação à escolha dos especialistas e desenvolvedores do domínio, foi realizada uma reunião com os gerentes de projetos da CMRJ e ficou acertada a participação de dois desenvolvedores do domínio na avaliação. Pelo excesso de trabalho de final de ano na CMRJ, não foi possível a disponibilização de outros desenvolvedores e especialistas do domínio, ficando acertado que a avaliação por parte de outros desenvolvedores e dos especialistas seria realizada *a posteriori*. Assim, da mesma forma como ocorreu com os alunos de graduação, foi realizada uma explanação a respeito da ferramenta e sua posterior disponibilização. Esta explanação detalhada durou cerca de 1 hora e foi estipulado um prazo de duas semanas para avaliação. O tempo da explanação foi menor do que o dos alunos de graduação, uma vez que os desenvolvedores já possuíam um conhecimento prévio do domínio e do projeto Odyssey de forma geral, apesar de não conhecerem a ferramenta a ser avaliada.

Durante o período de avaliação da ferramenta, algumas dúvidas surgiram, sendo todas sanadas através do uso de correio eletrônico. No entanto, somente o grupo de alunos de graduação utilizou este recurso.

Ao final de etapa de 3 e 2 semanas respectivamente, os alunos e os desenvolvedores do domínio receberam o questionário, via correio eletrônico, e tiveram um prazo de 2 dias para retorno do questionário preenchido. Todos os participantes preencheram o questionário e uma totalização e sumarização dos comentários foi realizada. Este processo de totalização e sumarização, realizado pelo engenheiro do domínio, durou cerca de 3 horas. A avaliação dos resultados é detalhada na próxima atividade.

Dificuldades:

Devido ao acúmulo de trabalhos na CMRJ, somente dois desenvolvedores do domínio puderam participar da avaliação. Em relação aos alunos de graduação, houve uma certa dificuldade de agendamento para a explanação da ferramenta.

Avaliação do tempo e resultados:

A tabela 6.10 a), b), c), d), e) e f) resumem os resultados das avaliações realizadas, divididas pelos itens avaliados no questionário.

Pergunta	*Resultados*	*Resumo dos Comentários Gerais*
Formação	Informática - 8	
Atuação	Aluno de Graduação – 6 Analista de Sistemas – 1 Gerente de Informática - 1	
experiência	Programação – 7 Gerência e especificação - 1	
Nível de Conhecimento do Domínio	Baixo – 6 Médio – 2	

Tabela 6.10 a - Conhecimento do Domínio e de técnicas de modelagem de informação

Pergunta	*Resultados*	*Resumo dos Comentários Gerais*
Em que grau o agente ajudou a melhorar seu nível de conhecimento a respeito do domínio	Bom – 7 Regular - 1	**Pontos Fortes:** • Detalhamento dos itens • Descoberta progressiva dos conceitos • Forma como as informações foram descritas • Descrição mais objetiva e resumida **Pontos Fracos:** • O exemplo do domínio era simples e não permitiu um aprofundamento do nível de entendimento.
Auxílio no desenvolvimento de aplicações do domínio	Sim - 8	**Pontos Fortes:** • Recuperação rápida das informações • Interações do agente levando a informações corretas • Base de componentes prévia • Familiarizando aos poucos com termos e assuntos do domínio

Tabela 6.10 b – Auxílio na compreensão do domínio e desenvolvimento de aplicações no mesmo

Pergunta	*Resultados*	*Resumo dos Comentários Gerais*
Facilidade de preenchimento do questionário de montagem de perfil	Ótimo – 1 Bom – 3 Regular - 4	**Pontos Fortes:** • Nível de conhecimento do domínio é um ponto interessante do questionário. **Pontos Fracos:** • Para usuários que só desejem conhecer o domínio, o campo aplicações deveria ficar desabilitado, pois pode confundir o usuário; • Falta de ajuda na semântica dos itens
Formato de apresentação	Ótimo – 4	**Pontos Fortes:**

das informações do agente	Regular - 4	• Detalhamento dos termos do domínio no diagrama para usuários não conhecedores do domínio. • Apresentação Completa. **Pontos Fracos:** • Descrição dos termos do domínio foi dificultada pela forma de visualização;
Telas de Alerta	Ótimo – 5 Bom – 2 Não soube responder - 1	**Pontos Fortes:** • Telas auxiliam a descartar informações desnecessárias • Alertas bem intuitivos
Interação do Agente durante a navegação	Ótimo – 2 Bom – 3 Regular - 3	**Pontos Fortes:** • *Ranking* de importância dos termos é interessante pois o usuário pode encontrar uma informação sem ter que navegar exaustivamente. • Forma de interação similar em todos os diagramas; • Telas de alerta auxiliam durante a navegação.
Atendimento às expectativas	Ótimo – 2 Bom - 6	**Pontos Fortes:** • Satisfeito • Fácil de identificar o que estamos procurando. **Pontos Fracos:** • Sugestão de informações às vezes são discordantes do que a priori achamos e não tem uma maneira de se verificar se o agente está correto.

Tabela 6.10 c -Adequabilidade das sugestões do agente

Pergunta	*Resultados*	*Resumo dos Comentários Gerais*
Interface do questionário de montagem de perfil do usuário	Ótimo – 3 Bom – 2 Ruim – 3	**Pontos Fortes:** • *Layout* simples e direto • Bem estruturado e objetivo **Pontos Fracos:** • Interface "desarrumada" • Parte de aplicações de palavras-chave desarrumada; • Parte visual deixa a desejar. • Interface em inglês. • Pode apresentar explicações resumidas para os itens
Facilidade de entendimento dos ícones e opções de menu	Ótimo – 2 Bom – 5 Regular – 1	**Pontos Fortes:** • *Hints* e figuras dos botões auxiliam; **Pontos Fracos:** • As opções do menu *pop-up* poderiam também ter ícones • Necessidade de se saber inglês, i.e., necessidade de versão em português.
Indicação do agente (Borda vermelha e numeração)	Ótimo – 5 Bom – 2 Regular - 1	**Pontos Fortes:** • Boa resposta visual ao usuário **Pontos Fracos:** • Deveria-se explicar que a opção é esta.

Tabela 6.10 d -Adequabilidade da interface com o usuário

Pergunta	*Resultados*	*Resumo dos Comentários Gerais*
Uso de algum agente de busca anteriormente	Não - 8	
Avaliação deste agente de busca em relação ao Odyssey-Search	Sem resposta - 8	
Uso de algum agente de busca de componentes	Não - 8	
Avaliação deste agente de busca de componentes em relação a Odyssey-Search	Sem resposta - 8	

Tabela 6.10 e - Adequabilidade da ferramenta comparada com outras ferramentas de busca

Pergunta	*Resultados*	*Resumo dos Comentários Gerais*
Avaliação Geral	Ótimo - 1 Bom – 4 Regular - 3	**Pontos Fortes:** • Software inovador para busca por informações para auxiliar no desenvolvimento de software. • Utilizaria se fosse desenvolver uma aplicação no domínio. • Informações importantes do domínio de fácil acesso. **Pontos Fracos:** • Assunto com alto grau de complexidade; • Necessidade de refinamento visual • Necessita de interface em português • *Help* • Necessita-se saber a partir de onde foram originadas as informações.

Tabela 6.10 f – Avaliação geral da ferramenta

6.3.3 Avaliação dos resultados do experimento

Alguns resultados preliminares podem ser obtidos, com base nas questões levantadas no início do experimento.

A ferramenta auxiliou no conhecimento do domínio em questão?

- No geral, os avaliadores consideraram adequada a utilização do agente para o desenvolvimento de aplicações no domínio, sendo esta adequabilidade principalmente relacionada ao entendimento do domínio.

A interface com o usuário da ferramenta é adequada ?

- Um ponto de muitas críticas por parte dos avaliadores foi a interface da infra-estrutura Odyssey, e consequentemente da Odyssey-Search, ser toda em inglês. Este ponto, segundo os avaliadores, dificulta o entendimento das interações do agente;

- Vários avaliadores apontaram a necessidade de um texto de ajuda na própria ferramenta para auxiliar o entendimento do agente;

As sugestões dadas pela ferramenta em relação às informações do domínio são realmente úteis?

- Outro ponto ressaltado foi o formato de apresentação das informações, que alguns acharam confuso. Este problema já havia sido constatado quando da apresentação de grande quantidade de informações do domínio e novas técnicas para apresentação destas informações já estão sendo avaliadas;
- Em relação a navegação e recuperação das informações do domínio utilizando o agente, todos consideraram bastante adequada, auxiliando na não apresentação de informações irrelevantes e guiando na descoberta de informações adequadas.

A ferramenta traz benefícios se comparada com outras ferramentas de busca gerais e ferramentas de busca de componentes?

- Este foi um ponto que não conseguimos avaliar com os resultados obtidos, uma vez que todos os avaliadores disseram não conhecer nenhuma ferramenta de busca em geral e nenhuma ferramenta de busca por componentes. Assim, a avaliação dos benefícios da Odyssey-Search se comparada com outras ferramentas de busca ficou prejudicada. As respostas dos avaliadores neste ponto nos surpreendeu, uma vez que com o uso da Internet, diversos mecanismos de busca são utilizados regularmente.

Assim, de acordo com as respostas dos avaliadores, podemos analisar as duas hipóteses levantadas.

1. **A utilização da ferramenta Odyssey-Search auxilia na compreensão do domínio;**

De acordo com os avaliadores, sim. Neste ponto, de acordo com o universo de avaliadores, a sua maioria na categoria de não conhecedores do domínio, o uso da ferramenta auxiliou uma maior compreensão do domínio em questão. Um avaliador com experiência no domínio sinalizou que este auxílio foi parcial, e o outro ressaltou que mesmo conhecendo o domínio as informações foram apresentadas em um formato

mais adequado, o que nos leva a inferir, em um primeiro momento, que a ferramenta é mais útil para não conhecedores do domínio.

2. **A utilização da ferramenta Odyssey-Search sugere componentes realmente úteis para o desenvolvimento de aplicações no domínio.**

Esta questão pôde ser avaliada parcialmente, uma vez que os participantes tiveram oportunidade apenas de observar as sugestões do agente, mas não de utilizar estas sugestões no desenvolvimento de aplicações no domínio. No que diz respeito somente à satisfação com as sugestões do agente, conforme pode ser observado na tabela 6.10 c, todos avaliaram como Ótimo e Bom.

Como avaliação final, podemos destacar que os avaliadores no geral consideraram as interações do agente adequadas (avaliações finais: Ótimo: 1, Bom: 4 e Regular: 3) mas um refinamento visual se faz necessário, principalmente com relação à forma de navegação e apresentação das informações do domínio.

6.3.4 Limitações

Uma limitação deste estudo de caso se refere ao curto espaço de tempo para sua realização, uma vez que durou cerca de 2 meses. Para uma avaliação mais precisa, seria necessário utilizar a ferramenta por mais tempo.

Outro problema é a não utilização da ferramenta com o real intuito de se desenvolver aplicações no domínio. Com isso, somente o aumento no grau de compreensão do domínio pôde ser avaliado de maneira adequada. O quanto a ferramenta é útil na reutilização de componentes do domínio não pôde ser avaliado de maneira adequada.

6.3.5 Evoluções necessárias

Com base nesta avaliação preliminar, podemos comprovar a necessidade de algumas melhorias na Odyssey-Search, principalmente com relação à interface. No entanto, as avaliações também mostraram que, para estes grupos de usuários específicos, uma ferramenta deste tipo melhora a busca por informações do domínio, antecipando informações que sequer o usuário supunha existir, conforme foi ressaltado nas respostas ao questionário de avaliação.

Algumas melhorias que devem ser realizadas na interface, detectadas através das avaliações são:

- Melhoria na apresentação visual do questionário sobre o perfil do usuário, disponibilizando menus de ajuda relacionados ao preenchimento do mesmo. Com este entendimento maior do questionário, a montagem do perfil do usuário poderá ser mais precisa e por conseqüência as interações do agente;
- Botões e menus de navegação mais intuitivos. Alguns avaliadores não conseguiram encontrar os botões de retorno às informações anteriores. Talvez o agente deva indicar estes botões realçando os mesmos durante a navegação;
- Interfaces devem ser disponibilizadas em português.

6.4 Conclusões

Neste capítulo, apresentamos dois estudos de caso com o objetivo de avaliar as propostas detalhadas no contexto desta tese.

De maneira geral, os estudos auxiliaram na melhoria das propostas, indicando onde estas poderiam sofrer mudanças. Em relação a verificar a adequabilidade das propostas, no contexto onde foram utilizadas, supomos que as mesmas já foram um primeiro passo para se ter uma avaliação e apontar a possibilidade de sucesso. No entanto, em um contexto mais geral, seriam necessários mais estudos de caso e experimentos para uma avaliação mais conclusiva.

Estas avaliações mais gerais serão realizadas através do projeto de transferência tecnológica entre a COPPE/UFRJ e a CMRJ. Como este projeto envolverá em um primeiro momento a CMRJ e depois outras casas legislativas, poderemos avaliar o Odyssey-DE e a Odyssey-Search em um contexto mais amplo.

Além disso, outros domínios devem ser avaliados. Esta possibilidade também está sendo concretizada, através de parcerias já firmadas, como por exemplo, com o Núcleo SofTex/JF, Embrapa e UFJF, e outras em andamento.

7 Conclusões e Perspectivas Futuras

Conforme ressaltamos no capítulo de introdução desta tese, a evolução na aplicação efetiva da reutilização de software foi pequena. Muitos trabalhos importantes foram realizados (FREEMAN, 1980), (NEIGHBOURS, 1981), (ARANGO, 1988), (PRIETO, 1991), (KANG *et al.*, 1990), (GRISS, 1998), mas ainda existem pesquisas a serem conduzidas.

Neste contexto, esta tese apresenta mais um passo na especificação de aspectos que promovem a reutilização, propondo uma abordagem integrada para a efetivação da reutilização, através do desenvolvimento dos seguintes aspectos:

- Especificação da arquitetura de uma infra-estrutura de apoio ao desenvolvimento destes componentes;
- Especificação de um processo para o desenvolvimento de componentes de software enfatizando características de reutilização. Esta especificação foi contemplada na implementação do suporte automatizado ao Odyssey-DE;
- Criação de mecanismos que facilitem a busca, recuperação e armazenamento de componentes, sendo estes mecanismos contemplados na implementação da Odyssey-Search.

Devemos destacar que o Odyssey-DE e a Odyssey-Search estão integrados, permitindo que em um mesmo ambiente, o componente possa ser desenvolvido, recuperado e selecionado através de busca local e remota.

Na literatura pesquisada (NEIGHBOURS, 1981), (ARANGO, 1988), (PRIETO, 1991), (MILLI et al., 1995), (GOMMA, 1995), (KANG et al., 1991), (SEACORD, 1998), (JACOBSON et al., 1997), entre outros, não encontramos nenhuma abordagem integrada que privilegie os aspectos apresentamos pela nossa proposta. Propostas pontuais relativas aos itens descritos acima são detalhadas na literatura, conforme descrevemos nas seções 7.1 e 7.2. No entanto, existem algumas abordagens que tratam de maneira integrada aspectos semelhantes à nossa proposta. Neste sentido, em (NEIGHBOURS, 1981) e (LEITE, 1994) é detalhada a abordagem DRACO e em (GOMMA, 1995), é detalhado o ambiente KBSEE.

O diferencial da nossa proposta em relação a proposta DRACO é a ênfase na recuperação de componentes distribuídos e desenvolvidos por terceiros. Atualmente, o tratamento adequado destes aspectos se mostra importante de modo a facilitar a recuperação destes componentes distribuídos de maneira adequada e eficiente.

A abordagem utilizada no ambiente KBSEE difere da nossa proposta principalmente em relação ao aspecto de integração das ferramentas utilizadas pelo ambiente. Podemos considerar que a integração entre as ferramentas é mínima, sendo necessárias diversas conversões para que as ferramentas possam se comunicar. Além disso, não são detalhados aspectos relativos a busca e recuperação de componentes especificados no ambiente e/ou desenvolvidos por terceiros. Em relação a possibilidade de uso de componentes desenvolvidos por terceiros, nenhum mecanismo específico é descrito.

Acreditamos que uma infra-estrutura especificada e implementada que envolva a integração de um processo de desenvolvimento de componentes e técnicas de armazenamento, busca e recuperação, como é o caso da infra-estrutura Odyssey, é fundamental para a aplicação efetiva da reutilização de software. A ausência de tal infra-estrutura integrada pode provocar incompatibilidades, gerando a necessidade de conversões, o que faz com que os benefícios esperados não sejam satisfatoriamente alcançados.

7.1 Desenvolvimento de Componentes Reutilizáveis

Nossa abordagem para a especificação de componentes reutilizáveis é sintetizada no processo Odyssey-DE, definido no capítulo 4, que é uma proposta de especificação de um processo de engenharia de domínio com ênfase no desenvolvimento de componentes reutilizáveis. Este processo pode ser visto como uma abordagem que evoluiu da engenharia de domínio pura para uma proposta mais voltada para o que denominamos de *engenharia de componentes*. Neste sentido, o processo possui uma ênfase na especificação de componentes reutilizáveis em todas as etapas, desde a fase de análise do domínio até a fase de implementação.

Algumas propostas de processos e métodos de engenharia de domínio já foram propostas na literatura, conforme descrevemos no capítulo2. Trabalhos como FODA (KANG *et al.*, 1990), FORM (KANG *et al.*, 1998), RBSE (JACOBSON *et al.*, 1997),

EDLC (GOMMA *et al.*, 1996), Catalysis (D'SOUZA *et al.*, 1998), OODE (CHAN *et al.*, 1998), entre outros foram analisados e chegamos à conclusão que estas abordagens tratam de um ou outro aspecto importante relacionado ao desenvolvimento de componentes de software reutilizáveis, mas não tratam de maneira integrada e detalhada estes aspectos, como é realizado no Odyssey-DE.

O método FODA (KANG *et al.*, 1990) foi um dos métodos de análise de domínio mais populares da década de 90. Podemos considerá-lo, em relação ao suporte ao desenvolvimento de componentes, como um método para organizar as informações do domínio, de forma a facilitar o entendimento deste domínio por parte dos desenvolvedores de software. No entanto, não é um método para o desenvolvimento de componentes reutilizáveis. Além disso, o FODA não tem preocupação com a especificação de mecanismos que permitam, em um primeiro momento, a criação de componentes reutilizáveis e facilitem, posteriormente, a recuperação destes componentes. Assim, o diferencial do Odyssey-DE em relação ao FODA é a preocupação com a identificação de componentes reutilizáveis em todas as etapas do processo. Além disso, o Odyssey-DE conta com o suporte automatizado para o armazenamento, busca e recuperação destes componentes reutilizáveis, baseado nas ligações entre as abstrações do domínio, especificadas através do modelo de características, e os componentes reutilizáveis. Este suporte adiciona um aspecto a mais ao Odyssey-DE em relação à promoção da aplicação efetiva da reutilização no desenvolvimento de software.

O método FORM é uma evolução do método FODA, com uma abordagem mais voltada para o desenvolvimento de componentes. O diferencial do Odyssey-DE em relação a este método é a ênfase não só na captura das funcionalidades do domínio mas também na captura dos conceitos do domínio e no relacionamento destes com os componentes, o que facilita a reutilização dos mesmos, uma vez que provê mais semântica na recuperação dos componentes. Neste sentido, o agente de armazenamento, busca e recuperação de componentes provê o uso de ontologias do domínio (modelo de características), conjugado com o uso da tecnologia de mediadores.

Os métodos Catalysis e RBSE seguem uma abordagem mais voltada para DBC. No entanto, a maior ênfase é na criação de aplicações que utilizam componentes e não na especificação de componentes para serem reutilizados. Em relação ao método ODM, este é

visto como um método com uma ênfase mais gerencial, com o objetivo de organizar as etapas de um processo de engenharia de domínio. Neste sentido, o Odyssey-DE poderia ser visto como uma possível instanciação do ODM, em relação a sua característica gerencial.

Analisando o Odyssey-DE em relação aos métodos EDLC e OODE, estes são similares em relação à utilização do paradigma orientado a objetos. No entanto, o Odyssey-DE possui uma maior preocupação no detalhamento do que seria um componente reutilizável, sendo que o EDLC e o OODE apenas utilizam a representação orientada a objetos em seus modelos.

Desta forma, podemos sintetizar como principais contribuições do Odyssey-DE, os seguintes pontos:

- Especificação e implementação de um modelo de abstrações do domínio e a posterior ligação destas abstrações com componentes do domínio em diversos níveis de abstração. O grande apelo deste modelo de abstrações é a posterior recuperação de componentes do domínio baseado em conceitos semânticos e em alto nível de abstração;
- Detalhamento e implementação de componentes reutilizáveis do domínio desde o nível conceitual até sua implementação;
- Especificação e implementação de uma abordagem de desenvolvimento de componentes, onde diferentes estilos arquiteturais podem ser utilizados, não se prendendo a um estilo específico, como é o caso do OODE.
- Especificação de uma arquitetura de apoio à utilização do processo, i.e., a arquitetura Odyssey.

Para observar a aplicação do processo Odyssey-DE em um domínio específico, foi realizado um estudo de caso que serviu para que pudéssemos melhorar o processo, através da identificação de pontos onde o mesmo precisa sofrer modificações e/ou evoluções, principalmente em relação à fase de projeto. Consideramos este estudo de caso um primeiro passo para o aperfeiçoamento do processo, contribuindo para ressaltar aspectos pontuais onde foram necessárias evoluções. No entanto, nenhuma afirmação pode ser feita em relação a eficácia do processo em qualquer situação de aplicação. A utilização efetiva do mesmo em outros domínios de aplicação se faz necessária, neste sentido. Para abordar este

aspecto específico, novos estudos de casos, envolvendo outros domínios de aplicação já estão em andamento.

7.2 Criação de mecanismos que facilitem o armazenamento, a busca e recuperação de componentes

Um dos requisitos para o sucesso da reutilização é a localização, compreensão e recuperação dos componentes necessários a um dado contexto. Para que isto seja possível, é necessária a disponibilidade de mecanismos específicos para o armazenamento, indexação e recuperação de informações sobre estes componentes.

A ferramenta Odyssey-Search é uma proposta neste sentido, sendo um sistema multi-agente voltado para o armazenamento, organização das abstrações do domínio, ligação destas abstrações com os componentes, busca e recuperação de componentes, como descrito no capítulo 5. Para isso, a Odyssey-Search utiliza os conceitos de agente de interface, de filtragem, e, de armazenamento e recuperação, conjugados. Nenhuma abordagem voltada especificamente para reutilização conta com este suporte abrangente da infra-estrutura Odyssey, utilizando tecnologias avançadas em suas diversas ferramentas. De acordo com SAMETINGER (1997) e YEE *et al.* (2000), a utilização deste tipo de tecnologia se faz cada vez mais necessária. Sametinger ressalta a importância dos repositórios de componentes serem divididos por domínio e o uso de um mecanismo de referência a estes repositórios, de maneira integrada. Ye *et al.* descrevem a importância de mecanismos mas ativos e mais direcionados para as necessidades específicas dos usuários (reutilizadores). A Odyssey-Search aborda ambos os aspectos através do uso da camada de mediação, onde cada mediador acessa componentes relacionados a um domínio específico e o gerente de serviços (GS) se encarrega de permitir o acesso integrado e/ou direcionado a estes domínios. Além disso, os agentes de filtragem e interface provêem os mecanismos ativos e focados nas necessidades do usuário.

Na literatura, podemos encontrar diversos sistemas de agentes tratando um ou outro aspecto da Odyssey-Search. No entanto, nenhum abordando todos os aspectos. Em relação à adaptação da interface, temos sistemas como o AVANTI (FINK *et al.*, 1996), (HyperContext (STAFF, 1997), PUSH (ESPINOZA, HOOK, 1996). Comparando estes sistemas com o agente de interface utilizado no Odyssey-Search, podemos dizer que o

sistema PUSH é o que mais se parece com os mecanismos de adaptação da Odyssey-Search, uma vez que na maior parte do tempo, o agente de interface que faz parte do Odyssey-Search é um agente autônomo, mas quando alguma situação de conflito se apresenta, este requisita o auxílio do usuário. Neste contexto, o Odyssey-Search está em conformidade com as tecnologias atualmente consideradas adequadas para o desenvolvimento de agentes de interface, ou seja, utiliza conceitos de autonomia e colaboração (com o usuário e com agentes de filtragem). Os outros sistemas apresentados utilizam uma abordagem ou outra.

No contexto de filtragem de informação, também existem algumas abordagens, descritas em detalhe no capítulo 3. Comparando estas com a abordagem utilizada na Odyssey-Search, em um contexto mais amplo, onde se inclui a busca e recuperação de informação, podemos notar que, no que tange os agentes de filtragem da Odyssey-Search, os mecanismos utilizados são bem similares aos utilizados pelas ferramentas descritas no capítulo 3. No entanto, quando analisamos a recuperação de informação, nenhum destes mecanismos utiliza uma abordagem onde a busca é baseada em ontologias especificadas e armazenadas por domínios, podendo esta busca ser estendida para outros domínios de aplicação, utilizando os relacionamentos entre termos ontológicos de diferentes domínios, como descrito no capítulo 5. Esta busca pode ser realizada tanto sobre dados estruturados (armazenados em bases de dados) quanto em dados não estruturados (armazenados em documentos). As ferramentas descritas acima se restringem à busca em documentos, e mais ainda, quando utilizam os mecanismos de busca convencionais como AltaVista e Yahoo, se restringem a mecanismos baseados em palavras chave ou índice, o que é bastante limitado. Assim, sob este ponto de vista, a Odyssey-Search agrega dois mecanismos poderosos na busca por informação, ou seja, o uso de modelos de usuário e técnicas inteligentes na filtragem de informação e uso de ontologias para realizar a busca por informações.

Em relação à recuperação de informação, abordagens utilizando a tecnologia de mediação conjugada com o uso ou não de ontologias já foram descritas na literatura. Dentre estas podemos destacar os sistemas SIMS, InfoSleuth e OBSERVER, descritos no capítulo 3. A diferença do sistema SIMS para a abordagem do Odyssey-Search é que no SIMS os agentes são organizados de forma hierárquica, não existindo uma comunicação entre

agentes no mesmo nível de abstração, ou seja, entre dois agentes que descrevem domínios complementares. Assim, os agentes (mediadores) no SIMS nunca se comunicam com outros agentes para obter dados relevantes em domínios similares, como é o caso da Odyssey-Search. Neste ponto específico, a Odyssey-Search preserva a autonomia dos domínios ao mesmo tempo que exibe os relacionamentos entre eles. Além disso, não é utilizado nenhum tipo de mecanismo de filtragem de informação direcionada para o usuário em questão. Atualmente, com o volume de informações que podem ser retornadas por uma simples consulta, esta é uma característica bem interessante.

Na abordagem adotada pelo Odyssey-Search, as preferências do usuário são levadas em consideração e inclusive o sistema faz sugestões, baseado nestas preferências, de informações que o usuário sequer supunha a existência. A interface de consulta é ligada a uma rede semântica que é utilizada na formulação da consulta para as fontes de dados. A Odyssey-Search também permite a consulta envolvendo múltiplas ontologias. Uma vez que uma dada consulta não é satisfeita utilizando-se uma dada ontologia, esta consulta pode ser especificada utilizando-se termos de outra ontologia, através dos relacionamentos ontológicos mapeados junto aos mediadores, e consultando-se outros tipos de bases de dados. Outro ponto interessante da abordagem do OBSERVER, que também está presente no Odyssey-Search, é que na formulação de consultas utilizando-se outras ontologias, é possível monitorar o nível de perdas semânticas da consulta. Assim, o usuário tem um controle melhor do que exatamente foi retornado.

Em relação ao armazenamento dos componentes do domínio, a infra-estrutura Odyssey também provê uma proposta interessante comparado-a com outras abordagens baseadas em reutilização. Dentre os aspectos tratados pela Odyssey-Search no armazenamento de componentes, vale a pena destacar:

- O armazenamento dos componentes reutilizáveis, divididos por domínios de aplicação, onde cada domínio possui uma ou mais bases de componentes, distribuídas ou não;
- O acesso às bases de componentes através de um modelo integrador do domínio, i.e. a ontologia do domínio, que no contexto da infra-estrutura Odyssey é o modelo de características.

Além de ser uma abordagem integrada para o armazenamento, a busca, compreensão e recuperação de componentes, podemos ressaltar os seguintes pontos de contribuição da proposta:

- Proposição de uma abordagem de recuperação de componentes baseada nas abstrações capturadas por especialistas e engenheiros do domínio. Esta abordagem permite a recuperação dos componentes a nível semântico e não a partir da sintaxe de sua interface, como realizado em outras abordagens (SEACORD, 1998);
- Acesso de forma facilitada a componentes distribuídos, através do uso da camada de mediação, onde repositórios de componentes remotos e heterogêneos podem ser agregados à estrutura de mediação, através da especificação de tradutores;
- Busca por componentes reutilizáveis em outros domínios de aplicação, se a busca realizada pelo usuário não é satisfeita no contexto do domínio de foco da aplicação a ser desenvolvida;
- Busca baseada nas preferências do usuário, disponibilizando componentes que o usuário sequer supunha que existiam;
- Uso de uma abordagem de recuperação de componentes integrada ao processo de reutilização da infra-estrutura Odyssey como um todo, uma vez que a Odyssey-Search está integrada ao Odyssey-DE.

Com isso, acreditamos que a Odyssey-Search é uma proposta inovadora para a resolução dos seguintes problemas levantados em (YE *et al.*, 2000):

- Esforço necessário para o entendimento dos componentes e conhecimento da existência dos mesmos;
- Custo associado com as mudanças do ambiente de desenvolvimento da aplicação para o ambiente de recuperação de componentes e vice-versa, no caso de se usar ferramentas distintas para o desenvolvimento da aplicação e para a recuperação de componentes, o que não é o caso da ferramenta Odyssey-Search, que é totalmente integrada à infra-estrutura Odyssey. Estas mudanças podem acarretar na interrupção do fluxo de trabalho e da seqüência de desenvolvimento da aplicação;

- Esforço de especificação de consultas utilizando linguagens de consulta proprietárias dos repositórios, quando estas linguagens estão disponíveis, ou então uso de mecanismos de consultas imprecisos, baseados em palavras-chave.

As abordagens contidas na Odyssey-Search, assim como em outras abordagens, possuem pontos a serem analisados com maior cuidado. Alguns deles são:

- O uso de uma estrutura de mediação é criticado por alguns especialistas (KANTORSKI et al, 2000) por ser uma estrutura onde existe uma troca de informações intensiva entre diversos módulos e este aspecto pode comprometer o desempenho da ferramenta como um todo;
- A especificação de tradutores para repositórios heterogêneos não foi abordada no contexto desta tese. Este é um ponto importante da ferramenta como um todo e trabalhos futuros neste sentido já estão sendo realizados.

7.3 Desdobramentos dos Resultados

A infra-estrutura Odyssey como um todo, o processo Odyssey-DE e a ferramenta Odyssey-Search foram publicados e apresentados em diversos foros internacionais e nacionais de pesquisa relacionados à área, como detalhado no capítulo de introdução desta tese (BRAGA, WERNER, 1999), (BRAGA *et al.*, 1998a), (BRAGA *et al.*, 1998b), (BRAGA *et al.*, 2000a).

Além disso, a partir de apresentações do projeto Odyssey em foros nacionais, como o Simpósio Brasileiro de Engenharia de Software, algumas parcerias estão sendo discutidas no sentido de se utilizar o projeto Odyssey em domínios de aplicação distintos.

Neste sentido, vale a pena destacar a utilização do Odyssey-DE e da infra-estrutura Odyssey no domínio de processos legislativos. Um fator preponderante é a existência, em âmbito nacional, de um programa denominado InterLegis, que visa a cooperação entre as casas legislativas. O programa tem como objetivo principal o fomento a projetos de casas legislativas, consideradas de referência, que auxiliem a melhoria do funcionamento de casas legislativas de menor porte. Neste contexto, o projeto de cooperação entre o grupo de reutilização da COPPE/UFRJ e a CMRJ, para o desenvolvimento de componentes reutilizáveis no domínio de processos legislativos, realizado nos últimos dois anos, foi

considerado de grande importância para a comunidade InterLegis, tendo sido indicado pelo ENIAL 2000 (Encontro Nacional de Informática Aplicada ao Legislativo) (WERNER *et al.*, 2000a), em junho deste ano, como prioritário para o programa InterLegis, ganhando o prêmio de melhor trabalho livre para o legislativo.

As propostas apresentadas nesta tese deram origem a diversos trabalhos de pesquisa que estão em andamento. Dentre esses trabalhos podemos destacar pesquisas na área de agentes inteligentes (COSTA *et al.*, 2000), Mediadores e publicação de componentes (PINHEIRO *et al.*, 2000), Ontologias (OLIVEIRA, 2000), Arquiteturas de Software (XAVIER, 2000), interfaces de componentes, mecanismos para conexão de componentes e processamento de consultas distribuídas (RUBERG, 2000).

7.4 Trabalhos Futuros

Com base nos estudos de caso realizados, além de constatações relacionadas à literatura existente, algumas melhorias necessárias tanto ao processo Odyssey-DE quanto à ferramenta Odyssey-Search já foram detectadas. Algumas destas melhorias já estão sendo realizadas em trabalhos de doutorado, mestrado e projetos de final de curso de graduação.

Em relação ao processo Odyssey-DE e o suporte automatizado ao mesmo, os seguintes pontos devem sofrer evolução:

- Detalhamento maior das fases de projeto e implementação do Odyssey-DE. Atualmente, estas etapas estão descritas no processo mas não foram realmente experimentadas em um caso real. O que foi feito e descrito no capítulo 6, foi a especificação de alguns componentes de projeto e de implementação mas não de todos os componentes do domínio. Com certeza, com um estudo de caso mais detalhado relativo a estas etapas, modificação das mesmas poderão surgir.
- Devido à complexidade da aplicação de um processo de ED, um suporte automatizado a todas as etapas se faz necessário. Atualmente, a infra-estrutura Odyssey provê suporte para a fase de análise e para algumas etapas da implementação (i.e., geração de código por classe). Neste sentido, é necessária a evolução da infra-estrutura de forma a atender as demais etapas do processo. Trabalhos neste sentido já estão sendo realizados.

- Necessidade de checagem de consistência entre os modelos utilizados, no sentido de evitar inconsistências nas diversas etapas de especificação dos componentes;
- Especificação detalhada das interfaces dos componentes arquitetural e implementacional, realizando-se um estudo detalhado das técnicas de conexão que poderiam ser aplicadas;
- Necessidade de uma ferramenta para acompanhamento do processo por parte da infra-estrutura Odyssey, onde um controle mais rigoroso de cada etapa e os produtos disponibilizados seria realizado.

Em relação a Odyssey-Search, as seguintes características podem ser melhoradas ou adicionadas à ferramenta:

- A apresentação das sugestões do agente pode levar a um efeito conhecido por *positive feedback* (WOODRIDGE, 2000), que é quando o usuário é levado a escolher sempre a sugestão dada pelo agente. Como o agente coloca a opção em vermelho, com bastante destaque, o usuário pode instintivamente sempre escolher esta opção. Como trabalho futuro relacionado a esta tese, está sendo desenvolvido um estudo no sentido de selecionar melhores técnicas a serem utilizadas nesta apresentação de forma a tentar minimizar este efeito;
- O algoritmo atualmente utilizado para o batimento de palavras é bastante simples, considerando somente palavras inteiras e não radicais de palavras. Além disso, o algoritmo não tem o desempenho desejado quando uma grande quantidade de informações precisa ser analisada. Assim, estudos estão sendo realizados no sentido de aperfeiçoar este algoritmo de forma a obter melhores resultados.
- Atualmente, o protótipo da Odyssey-Search não conta com o acesso a informações disponibilizadas na Internet. No entanto, o módulo que irá prover esta funcionalidade já está em desenvolvimento, como dissertação de mestrado (COSTA, 2000).
- O agente de recuperação da Odyssey-Search, atualmente, não possui acesso a nenhum tipo de base heterogênea e/ou base de terceiros. Apesar de utilizar o conceito de múltiplas ontologias e acesso a componentes de domínios heterogêneos, este acesso é feito apenas para componentes criados a partir da infra-estrutura Odyssey. O

acoplamento de bases de terceiros ao agente de recuperação demanda o desenvolvimento de um tradutor desta base para o modelo da Odyssey-Search. O desenvolvimento deste tradutor, dependendo de como está estruturada esta base de terceiros, nem sempre é trivial de ser realizado. Neste sentido, a arquitetura *Le Select* (BOUGAMIN *et al.*, 1998) está sendo acoplada ao agente de recuperação da Odyssey-Search. Este acoplamento se mostra vantajoso a partir do momento que esta arquitetura possui um mecanismo facilitado para a criação de novos tradutores a partir de seu contexto. Com isso, o problema atual de acoplar bases de terceiros à infra-estrutura Odyssey poderia ser minimizado. Este trabalho já está sendo realizado como parte de uma dissertação de mestrado (PINHEIRO, 2000);

- O processamento de consultas atual da Odyssey-Search não utiliza nenhuma espécie de otimização para consultas distribuídas. Está sendo desenvolvido uma dissertação de mestrado (RUBERG, 2000) que prevê esta otimização, utilizando dados estatísticos para prover o melhor caminho para o processamento das consultas em múltiplas bases distribuídas;
- No que tange a busca por componentes reutilizáveis em outros domínios de aplicação, atualmente utilizamos um conjunto de relacionamentos básicos entre termos de diferentes ontologias. No entanto, novos relacionamentos podem ser definidos, além de se prover um suporte mais automatizado para a "descoberta" destes relacionamentos. Neste sentido, está sendo desenvolvido um trabalho (OLIVEIRA, 2000) que prevê este suporte no contexto da Odyssey-Search, em particular, e do SGO GOA++ (MAURO *et al.*, 1997), em um contexto mais geral, a partir da especificação de um servidor de relacionamentos ontológicos.
- O detalhamento e descrição das interfaces dos componentes ainda precisa ser aprimorado no contexto da infra-estrutura Odyssey de forma geral e da Odyssey-Search especificamente. Neste sentido, pesquisas envolvendo o uso de XML para a representação da interface dos componentes estão sendo realizadas;
- Outro ponto específico onde o uso de XML está sendo empregado é na publicação dos componentes desenvolvidos no contexto da infra-estrutura Odyssey, sendo a descrição do componente publicada no formato XML (PINHEIRO *et al.*, 2000).

Conforme podemos observar nos diversos relatos descritos na literatura, a reutilização ainda não é uma prática comum entre os desenvolvedores de software. Neste sentido, esta tese teve como objetivo apresentar novos caminhos nesta direção, propondo abordagens e ferramental específico para esta efetivação. Acreditamos que o desenvolvimento baseado em componentes, e o uso de mecanismos mais eficientes para a busca destes componentes sejam um caminho promissor neste sentido.

Bibliografia

ANGELE, J., FENSEL, D., STUDER, R., 1996, Domain and Task Modeling in MIKE. In: A. Sutcliffe, D. Benyon, F. van Assche (Eds.): Domain Knowledge for Interactive System Design, *Proceedings of IFIP 8.1/13.2 Joint Working Conference*, Genebra, Maio, Chapman & Hall.

ARANGO, G., 1988, *Domain Engineering for Software Reuse*, Technical Report UCI-ICS 88-27, Universidade da Califórnia.

ARC, 1996, Army Reuse Center – Domain Selection Report at http://sort.ivv.nasa.gov/related.htm.

ARMSTRONG, R. et alli, 1995, WebWatcher: A Learning apprentice for WWW, *Proceedings of AAAI Spring Symposium on Information Gathering form Heterogeneous Distributed Resources*, Stanford, Califórnia, Estados Unidos.

ATKINSON, C., et alli, 2000; Component-Based Software Engineering: The KobrA Approach, International Workshop on Component-Based Software Engineering, Limerick, Irlanda, Junho.

ATKINSON, S., 1998, Modelling Formal Integrated Component Retrieval Proceedings of the Fifth International Conference on Software Reuse, pp. 337-347, Junho, Vitória, Canadá.

BAESA-YATES, R., RIBEIRO-NETO, B., 1999, *Modern Information Retrieval.* Addison-Wesley.

BALABANOVIC, M., SHOHAM, Y., 1995, Learning Information Retrieval Agents: Experiments with Automated Web Browsing, *Proceedings of AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Resources*, Stanford, Califórnia, Estados Unidos.

BARROS, M., 1995 , Recuperação de Componentes em Bibliotecas de Software: Uma Abordagem Conexionista, Tese de Mestrado, COPPE/UFRJ, Setembro.

BARROS, M., 2000, *Reutilização de Conhecimento no Gerenciamento de Projetos Baseado em Cenários*, : Engenharia de Domínio e Desenvolvimento Baseado em Componentes, Relatório Técnico do Projeto Odyssey 10/2000, COPPE/UFRJ, Rio de Janeiro, Brasil.

BAYARDO,R., et alli, 1997, InfoSleuth: Agent-based semantic integration of informationin open and dynamic environments, *Proocedings of 1997 ACM International Conference on Management of Data (SIGMOD),* Tucson, Arizona, Maio.

BENAKI, E., et alli, 1997; User Modeling in WWW: the UMIE Prototype *Proceedings of the workshop "Adaptive Systems and User Modeling on the World Wide Web" Sixth International Conference on User Modeling*, Chia Laguna, Sardenha, Itália, Junho.

BIGUS, J., BIGUS, J., 1998, Constructing Intelligent Agents with Java, Wiley.

BOOCH, G., RUMBAUGH, J., JACOBSON, I., 1998, *The Unified Modeling Language User Guide*, 1 ed., Nova York, Addison-Wesley Longman.

BOUGANIN, L.,KAPISKAIA, O., VALDURIEZ, P., 1998, Memory-adaptive Scheduling for Large Query Execution. Intl. Conf. On Information and Knowledge Management (CIKM), Bethesda, Maryland, Estados Unidos.

BRAGA, R., 1995, *Modelagem e Implementação de Um Ambiente de Sistemas de Informação de Escritório Usando um Banco de Dados Orientados a Objetos*, Tese de Mestrado, COPPE/UFRJ.

BRAGA, R., WERNER, C., 1999a, Odyssey-DE: Um Processo para Desenvolvimento de Componentes Reutilizáveis, *X Conferência Internacional em Tecnologia de Software* (X CITS), Curitiba, Brasil, Maio.

BRAGA, R., WERNER, C., MATTOSO, M. 1998b, A Reuse Infrastructure based on Domain Models, Proc. 9th Int.Conf. on Computing and Information (ICCI'98)-pp.227-234, Winnipeg, Canada, Junho.

BRAGA, R., WERNER, C., MATTOSO, M., 1998a, A Reuse Infrastructure based on Domain Models, Panel at ICSR5, Vitória, Canadá.

BRAGA, R., WERNER, C., MATTOSO, M., 1999b, Odyssey: A Reuse Environment Based on Domain Models, *2nd IEEE Symposium on Application-Specific System and Software Engineering Technology (ASSET'99)*, Richardson, Estados Unidos.

BRAGA, R., WERNER, C., MATTOSO, M., 2000a, A Reuse Infrastructure based on Domain Models, Special Issue of the Journal of Computing and Information, (vol.3/ICCI98/2/3, 2000), pp.227-234, ISSN 1201-8511.

BRAGA, R., WERNER, C., MATTOSO, M., 2000b, Using Ontologies for Domain Retrieval, *11th International Conference and Workshop on Database and Expert Systems Applications (DEXA'00) - Workshop on Domain Engineering*, Greenwich, Inglaterra, Setembro.

BRAGA, R., WERNER, C., MATTOSO, M., 2000c, A Multi-Agent System for Domain Information Discovery and Filtering, *Proceedings of XV Simpósio Brasileiro de Engenharia de Sofware*, João Pessoa, Outubro.

BRAGA, R.; MATTOSO, M.; WERNER, C., 1999c; The Use of Mediators for Component Retrieval in a Reuse Environment, : *Proc. Technology of Object-Oriented Languages and Systems* (TOOLS-30 USA'99) Conference, Workshop on Component-Based Software Engineering Process, IEEE CS Press, Santa Barbara, Califórnia, Agosto, 1999, pp.542-546.

BRAGA,R., WERNER, C.; 1999, Processo de Engenharia de Domínio do Ambiente Odyssey, Relatório Técnico do Projeto Odyssey 4/99, COPPE/UFRJ.

BRERETON, P., 1999, Evolution of Component-Based Systems, International Workshop on Component-Based Software Engineering, Los Angeles, Estados Unidos, Maio.

BRIGHT, M.;HURSON, A.; PAKZAD, S., 1992, *A taxonomy and current issues in multidatabase systems*. Computer, Nova York, v.25, n.3, pp.50-62.

BROWN, A., WALLNAU, K., 1998, The Current State of CBSE, *IEEE Software*, Setembro/Outubro, pp. 37-46.

BROWNE, D., TOTTERDELL, P., NORMAN, M., 1990, *Adaptative User Interface*, ed1, Academic Press, Califórnia, Estados Unidos.

BUSCHMANN, F., et alli, 1996, *Pattern-Oriented Software Architecture - A system of patterns* , John Wiley Publishing.

CAMPOS, F., COELHO, F., BRAGA, R., 2000, Software Agropecuário: um Processo de Avaliação em Quatro Níveis, *WorkShop de Qualidade de Software, XIV Simpósio Brasileiro de Engenharia de Software*, Outubro, João Pessoa.

CBSE, 1998, 1999, 2000, Component Based Software Engineering Workshops Series at http://www.sei.cmu.edu/cbs/.

CHAN, S., LAMMERS, T., 1998, OODE, *5th International Conference on Software Reuse (ICSR-5)*, Vitória, Canadá, Junho.

CHAWATHE, S.; MOLINA, H.G.; HAMMER, J., 1994, The TSIMMIS Project: Integration of Heterogeneous Information Sources. : Proceedings of ANNIVERSARY MEETING OF THE INFORMATION PROCESSING SOCIETY OF JAPAN, Tokyo, Japão. pp.7-18.

CHEN, L., SYCARA, K., 1998, WebMate: A Personal Agent for Browsing and Searching, *Autonomous Agents'98 Conference*, Minneapolis, Estados Unidos.

CHUNG, C., 1990, DATAPLEX: An Access to Heterogeneous Distributed Databases. *Communications of the ACM*, Nova York, v.33, n.1, pp.70-80, Janeiro.

CIMA, A., 1996, *Desenvolvimento de Software com Reutilização Baseada em "Frameworks" Orientados a Domínio*, Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil.

CMRJ, Documento Interno de Análise Essencial da tramitação no legislativo, 1988.

COCKBURN, Alistar, 1999, Use Case Templates, at http://members.aol.com/acockburn

COHEN, S, 1994, Feature-Oriented Domain Analysis: Domain Modeling, Tutorial Notes; *3rd Int. Conference on Software Reuse*, Rio de Janeiro, Novembro.

COHEN, S, 1998 Object-Oriented Technology and Domain Analysis, *5th International Conference on Software Reuse (ICSR-5)*, Vitória, Canadá, Junho.

COSTA, M, 2000, Um Sistema Multi-Agente para Busca e Filtragem de Informações de Domínio na Internet, *XIV Simpósio Brasileiro de Engenharia de Software*, Workshop de Teses, João Pessoa, outubro.

D'SOUZA, D., WILLS, A., 1998, *Objects, Components and Frameworks with UML: The Catalysis Approach*, 1 ed., Massachusetts, Addison-Wesley Longman..

DANTAS, A., 2000, *Agentes para Verificação de Consistência e Críticas de Modelos OO*, Projeto Final de Curso, COPPE/IM/UFRJ, Rio de Janeiro, Brasil (em andamento).

DAVIES, N.J.; WEEKS, R, 1998, Jasper: Communicating Information Agents for WWW ,at http://www.labs.bt.com/projects/knowledge/jaspaper.htm.

DIGRE, T., 1998, Business Object Component Architecture, *IEEE Software*, Setembro/Outubro, pp. 60-69.

ESPINOZA, F., HOOK, K., 1996, A WWW Interface to an Adaptative Hypermedia System, PAAM'96, Londres, Inglaterra.

EUZEMAT, J., 1996, Corporate Memory through cooperative creation of knowledge base and hyper-documents, *Proceedings of KAW96*.

FAYAD, S, HAMU, D., BRUGALI, D., 2000, *Enterprise frameworks characteristics, criteria, and challanges*, CACM, Vol. 43, No 10, Outubro.

FININ, T., LABROU, Y., 1994, A semantics approach for KQML -- a general purpose communication language for software agents, *Third International Conference on Information and Knowledge Management* (CIKM'94).

FININ, T., NICHOLAS, C., MAYFIELD, J., 1998, Software Agents for Information Retrieval, apresentação disponível em www.cs.umbc.edu/abir.

FINK, J., KOBSA, A, NILL, A., 1996, User-Oriented Adaptativity and Adaptability in the AVANTI Project, *Proceedings of the Design for the Web Conference.*

FLEMING, F.,COHEN,R.,1999, User Modeling in the Design of Interactive Interface Agents, *Proceedings of International Conference in User Modeling 1999* (UM'99).

FORSELL, M., HALTUNEN, V., AHONEN, J., 2000; Use and Identification of components in component-Based Software Development Methods, *6th International Conference on Software Reuse (ICSR-6)*, Viena, Áustria, Junho.

FOWLER, Martin, 1997, *Analysis Patterns - Reusable Object Models* - Addison Wesley Publications.

FRAGOUDIS, D., LIKOTHANASSIS, S., 1999, User Modeling in Information Discovery: An Overview, *Proocedings of International Conference in User Modeling 1999* (UM'99).

FRAKES, W.,1994, Systematic Software Reuse: a paradigm shift; 4th *International conference on software reuse*, Rio de Janeiro, Brasil, Novembro.

FREEMAN, P., 1980, *Reusable Software Engineering: A statement of long-range research objectives*, Technical Report TR-159, Universidade da Califórnia.

FREEMAN, P., 1983, Reusable Software Enginnering: concepts and Research Directions. *Proocedings of the Workshop on Reusability in Programming*, pp. 129-137. ITT, Setembro.

FREEMAN, P., 1984, A conceptual Analysis of the Draco Approach to Constructing Software Systems, *Trans. On Software Enginnering*, SE-13(7): 830-844, Julho.

GAMMA, E., HELM, R., JOHNSON, R., *et alli.*, 1994, *Design Patterns: Reuse of Object Oriented Design*, 1 ed., Nova York, Addison-Wesley Longman.

GARLAN, D., ALLEN, R., MELTON, R.,1994, Architectural Mismatch: or Why It's Hard to Build Systems Out of Existing Parts, *Proceedings of the International Conference on Software Engineering*, Seattle.

GOMAA, H., 1995, Reusable Software Requirements and Architectures for Families of Systems, *Journal of System Software*, Novembro, pp. 189-202.

GOMMA, H., 2000; Object Oriented Analysis and Modeling for Families of Systems with UML, *6th International Conference on Software Reuse (ICSR-6)*, Viena, Áustria, Junho.

GRISS, M., 2000; Implementing Product-Line Features with Component Reuse, *6th International Conference on Software Reuse* (ICSR-6), Viena, Áustria, Junho.

GRISS, M., FAVARO, J., D'ALESSANDRO, M., 1998, Integrating Feature Modeling with RSEB, *5th International Conference on Software Reuse (ICSR-5)*, ACM/IEEE, Vitória, Canadá, Junho.

GUARINO, N., 1998, "Formal Ontology and Information Systems", *1st International Conference on Formal Ontology in Information Systems*, IOS Press, Trento, Itália, Junho.

HADJAMI, H., GHEZALA, B., KAMOUN, F., 1995, A Reuse Approach Based on Object Orientation. Its Contributions in the Development of CASE Tools, *Proceedings of the Symposium on Software Reusability*, pp. 233-237, Seattle.

HALL, P.,2000, "Educational Case Study–What is the Model of an Ideal Component? Must it be an Object?", *International Workshop on Component-Based Software Engineering*, Limerick, Irlanda, Junho.

HALLSTEINSEN, S., SKYLSTAD, G., 1999, The Magma approach to CBSE, *International Workshop on Component-Based Software Engineering*, Los Angeles, Estados Unidos, Maio.

HAN, J, 1999, An Approach to Software Component Specification, *International Workshop on Component-Based Software Engineering*, Los Angeles, Estados Unidos, Maio.

HAN, S., et alli, 1998, WebACE: A Web Agent for Document Categorization and Exploration, *Autonomous Agents'98 Conference*, Minneapolis, Estados Unidos.

HEINEMAN, 1999, An Evaluation of Component Adaptation Techniques, *International Workshop on Component-Based Software Engineering*, Los Angeles, Estados Unidos, Maio.

HERBERT, J., PRICE, A.,1999, Utilizando TestFlows no teste cooperativo de software OO, *XI CITS*, Curitiba.

HUHNS,M., SINGH, M., 1997, *Managing Heterogeneous Transaction Workflows with Co-operating Agents*, Agent Techonoly Jennings & Woodridge (eds), Springer.

IBM, 2000, www.ibm.com.

INPRISE, 2000, www.inprise.com.

JACOBSON, I., BOOCH, G., RUMBAUGH, J., 1999, *The Unified Software Development Process*, 1 ed., Massachusetts, Addison-Wesley Longman.

JACOBSON, I., CHRISTERSON, M., JONSSON, P., *et alli*, 1992, *Object-Oriented Software Engineering – A Use-case Driven Approach*, 1 ed., Massachusetts, Addison-Wesley Longman.

JACOBSON, I., GRISS, M., JONSSON, P., 1997, *Software Reuse. Architecture, Process and Organization for Business Success*, 1 ed., Nova York, Addison-Wesley Longman.

JANCA, P., GILBERT, D., 1997, *Practical Design of Intelligent Agent System*, Agent Technology Jennings & Woodridge (eds), Springer.

JAVASUN, 2000, Página de referência do Java, http://www.java.sun.com.

JAVAWEB, 2000, www.java.sun.com.

JENNINGS, N., et alli, 1998, *A Roadmap of Agent Research and Development*, Autonomous Agents and Multi-Agent Systems, 1 7-38, Kluwer Academic Publisher.

JENNINGS, N., WOOLDRIDGE, M., 1997, *Applications of Intelligent Agents*, Agent Techonoly Jennings & Woodridge (eds), Springer.

JOHNSON, R., 1992, Documenting Frameworks Using Patterns, *Proceedings of the Conference on Object-Oriented Programming System, Languages and Interfaces (OOPSLA'92)*, pp. 63-76, Vancouver, Canadá, Outubro.

KANG, K., COHEN, S., HESS, J. *et alli*, 1990, *Feature-Oriented Domain Analysis (FODA) - Feasibility Study*, SEI Technical Report CMU/SEI-90-TR-21.

KANG, K., KIM, S., LEE, J., *et al.*, 1998, *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*, SEI Technical Report.

KANTORSHI, G., RIBEIRO, C., 2000, Interoperabilidade de Bancos de Dados Heterogêneos através da WWW, *XIV Simpósio Brasileiro de Banco de Dados*, João Pessoa, Outubro.

KAZAM, 2000, http://www.sei.cmu.edu/cbs/papers/eval_bib.html

KITCHENHAN, B., PICKARD, L., PFLEEGER, S., 1995, Case Studies for Method and Tool Evaluation, *IEEE Software*, Julho, pp. 52-62.

KOZACZYNSKI, W., 1999, Composite Nature of Component, *International Workshop on Component-Based Software Engineering*, Los Angeles, EUA, Maio.

KRUEGER, C., 1992, Software Reuse, : *ACM Computing Surveys*, v. 24, n. 2, pp. 131-183.

LARSEN, G., 2000, *Component Based Enterprise Frameworks*, CACM, Vol. 43, No 10, Outubro.

LEITE, J., 1994, SANT'ANNA, M. et alli, Draco-Puc : A Technology Assembly for Domain Oriented Software Development., Proceedings of Third International Conference on Software Reuse, IEEE Computer Society Press, Rio de Janeiro.

LIEBERMAN, H., 1995, Letizia: An Agent that Assists Web Browsing, *International Joint Conference on Artificial Intelligence*, Montreal, Agosto.

LIM, W.,1998 , *Managing Software Reuse*, Prentice Hall.

LINO, S., 1999, Sistemas de Consultas Visuais para SGBD-OOs com Interfaces Inteligentes: Arquitetura e Implementação, Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil.

LINTON, F., JOY, D., SCHAEFER,H.,1999, Building User and Expert Models by Long-Term Observation of Application Usage, *Proocedings of International Conference in User Modeling 1999* (UM'99).

LIU, L., PU, C., 1996, " Metadata in the Interoperation of Heterogeneous Data Sources ", *the proceedings of the First IEEE Metadata Conference*, Abril.

LUKE, S., et al, 1997.; "Ontology-based Web Agents"; Proceedings of First International conference on Autonomous Agents.

MAES, P., 2000, http://pattie.www.media.mit.edu/people/pattie.

MATTOSO, M.; WERNER, C.; BRAGA, R.; PINHEIRO, R.; MURTA, L.; ALMEIDA, V.; COSTA, M.; BEZERRA, E. SOARES, J.; RUBERG, N.; 2000, Persistência de Componentes num Ambiente de Reuso, XIV Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, João Pessoa, outubro, pp.351-354.

MATTOSO, M.L.Q. SOUZA, J.M., 1994, GOA: Um Servidor de Objetos Persistentes para Sistemas de Banco de Dados Orientado a Objetos, *XX Conferência Latinoamericana de Informática*, Cidade do México, México, setembro.

MAURO, R., ZIMBRÃO, Z., BRÜGGER, T., *et al.*, 1997, GOA++: Tecnologia, implementação e extensões aos serviços de gerência de objetos, *Anais do XII Simpósio Brasileiro de Banco de Dados (XII SBBD)*, pp. 272-286, Fortaleza, Brasil, Outubro.

McILROY, M., 1976, "Mass produced Software Components, from 1969 NATO Conference in Software Engineering. *Sofware Enginneering Concepts and Techniques*, pp. 88-98. Petrocelli/Cahrter, Bruxelas, 1976.

MENA, E., 1998: OBSERVER: An Aproach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies, Doctoral Thesis, Universidade de Zaragoza, Novembro.

MENCZER, F., BELEW, R.K., 1998, Adaptative Information Agents in Distributed Textual Environments, *Autonomous Agents'98 Conference*, Minneapolis, Estados Unidos.

MENZIES, T. et alli, 1998, Evaluating a Temporal Causal Ontology *Proocedings of the International Conference on Formal Ontology in Information Systems* (FOIS'98).

MERAD, S., 1999, Solution Concepts for the Optimal Selection of Software Components, *International Workshop on Component-Based Software Engineering*, Los Angeles, Estados Unidos, Maio.

MICROSOFT, 2000, www.microsoft.com.

MILLER, N., 2000, A Engenharia de Aplicações no Contexto da Reutilização Baseada em Modelos de Domínio, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

MILLI, H., MILLI, F., MILLI, A., 1995, Reusing Software: Issues and Research Directions. *IEEE Transactions on Software Engineering*, 21 (6): 528-562.

MLADENIC, D.; 1998, *Machine Learning on non-homogeneous, distributed text data*, Tese de Doutorado, University of Ljubljana.

MOTTA, C., 1999, *Um Ambiente de Recomendação e Filtragem Cooperativas para Apoio a Equipes de Trabalho*, Tese de Doutorado, COPPE/UFRJ, Janeiro.

MOUKAS, A, 1997; *Amathea: Information Filtering and discovery Using a Multiagent Evolving System*, Master Thesis, MIT.

MURTA, L., 1999, FRAMEDOC: Um FrameWork para a Documentação de componentes Reutilizáveis, Projeto Final de Curso, DCC/IM/UFRJ, Novembro.

MURTA, L., 2000, Uma Máquina de Processos de Desenvolvimento de Software Baseada em Agentes Inteligentes, *XIV Simpósio Brasileiro de Engenharia de Software, Workshop de Teses*, João Pessoa, outubro.

NEIGHBORS, J., 1981, *Software Construction Using Components*, Tese de D.Sc., Universidade da Califórnia, Irvine, Estados Unidos.

NWANA, H., NDUMU, D., 1997, *A Brief Introduction to Software Agent Technology*, Agent Technology Jennings & Woodridge (eds), Springer.

ODMG, 2000, www.odmg.org.

OLIVEIRA, A., 2000, Servidor de relacionamentos ontológicos, Tese de Mestrado, COPPE/UFRJ (em andamento).

OMG, 2000, www.omg.org.

ONTOLIGUA, 2000, http://ksl-web.stanford.edu/knowledge-sharing/ontolingua/

ÖSZU, M., VALDURIEZ, P.,1999, Principles of Distributed Database Systems, Nova Jersey, Prentice-Hall.

PAEPCKE, A., GARCIA-MOLINA, H., RODRIGUEZ-MULA, G., CHO, J., 1999, Beyond Document Similarity: Understanding Value-Based Search and Browsing Technologies, documento de trabalho disponível em http://www-diglib.stanford.edu/.

PALAZZO, L., 1996, Agentes de Informação Inteligentes, Relatório Técnico, UFRGS.

PARRISH, A., DIXON, C., HALE, D.,1999, Component Based Software Engineering: A Broad Based Model is Needed, *International Workshop on Component-Based Software Engineering*, Los Angeles, Estados Unidos, Maio.

PEREIRA, R.C.G., 1999, Uma abordagem para gerenciamento de consistência em um ambiente de banco de dados heterogêneos, Tese de Mestrado, UFPE, Fevereiro.

PINHEIRO, R., 2000, Serviços para Publicação e Recuperação de Componentes de Software através da Internet em Ambientes de Reuso, *XIV Simpósio Brasileiro de Engenharia de Software, Workshop de Teses,* João Pessoa, Outubro.

PIRES, P., MATTOSO, M., 1997, A CORBA based architecture for heterogeneous information source interoperability, *Proceedings of Technology of Object-Oriented Languages and Systems - TOOLS'25*, IEEE CS Press, Melbourne Austrália, Novembro, pp.33-49.

POHL,W., NICK, A., 1997, Machine Learning and Knowledge Representation in the LaboUr Approach to User Modeling, *Proocedings of International Conference in User Modeling 1997* (UM'97), Viena.

POULIN, J.,1995, Populating software repositories: Incentives and domain specific software, *The Journal of Systems and Software*, 30 (3): 187-199, Setembro.

POYNTON, B., LANERGAN, R., 1979, Reusable Code: The Application Development Technique of the Future, Proocedings of the IBM SHARE/GUIDE Software symposium, IBM, Monterey, California.

PRESSMAN, R., 1997, *Software Engineering – A practitioner's approach*, 3 ed., Cingapura, McGraw-Hill Editions.

PRIETO-DIAZ, R., 1987, A software Classification Scheme for Reusability. *IEEE Software*, 4(1): 6-16, Janeiro.

PRIETO-DIAZ, R., ARANGO, G., 1991, *Domain Analysis and Software System Modeling*, IEEE Computer Society Press Tutorial.

RATIONAl, 2000, www.rational.com.

RIG, 1999, Reusability Library Interoperability Group, http://www.asset.com/rig.

ROCHA, A., WERNER, C., TRAVASSOS, G., *et alli*, 1996, Processo de Desenvolvimento de Software Baseado em Reutilização, *VII Conferência Internacional de Tecnologia de Software (VII CITS)*, Curitiba, Brasil, Junho.

RUBERG, N., 2000, Otimização de consultas em ambientes distribuídos, Tese de Mestrado, COPPE/UFRJ (em andamento).

RUMBAUGH, R., et alli, 1991, Object-Oriented Modeling and Design, Prentice Hall.

SALTON, G., 1989, Automatic Text Processing, Addinson-Wesley.

SAMETINGER, J., 1997, *Software Engineering with Reusable Components*, Springer.

SANT'ANNA , M., LEITE, J., PRADO, A, 1998, A Generative Approach to Componentware, *International Workshop On Component-Based Software Engineering*, Kyoto, Japão.

SEACORD, R., 1999, Software Engineering Component Repositories, *International Workshop on Component-Based Software Engineering*, Los Angeles, Estados Unidos, Maio.

SEACORD, R., HISSAN, S., WALLNAU, K., 1998, *Agora: A Search Engine for Software Components*, SEI Technical Report CMU/SEI-98-TR-011.

SHAW, M.; GARLAN, D., *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996

SHETH, A.; LARSON, J., 1990, Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. *ACM Computing Surveys*, Nova York, v.22, n.3, Setembro.

SILVA, M., WERNER, C., 1996, "Packing Reusable Components Using Patterns and Hypermedia", *4th International Conference on Software Reuse*, Orlando, Estados Unidos, Maio.

SIMOS, M., 1996, Organization Domain Modeling (ODM): Domain Engineering as a Co-Methodology to Object-Oriented Techniques, : *Fusion Newsletter*, v. 4, Hewlett-Packard Laboratories, pp. 13-16.

SIMOS, M., ANTHONY, J., 1998, Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering, : *Proceedings of the 5th International Conference on Software Reuse (ICSR-5)*, ACM/IEEE, pp. 94-102, Vitória, Canadá, Junho.

STAFF, C., 1997, HyperContext: A Model for Adaptative Hypertext, *Proocedings of the Sixth International Conference in User Modeling, UM'97*, Viena.

STRAUCH, J. M., 1998, Integração de Bases de Dados Geográficas Heterogênea e Distribuídas, Tese de Doutorado, COPPE/UFRJ, Dezembro.

SUBRAHMANIAN, V. et al., 1999, HERMES: A Heterogeneous Reasoning and Mediator System. Disponível por www em: http://www.cs.umd.edu/ projects/hermes/overview/paper/index.html.

SZYPERSKIi, C., 1998, Component Software: Beyond Object-Oriented Programming, Addison-Wesley.

TRANNIN, M.; SANTOS, F.; WERNER, C.M.L., BORGES, M.; 1999, Uma Infra-estrutura de apoio à Aquisição Cooperativa de Conhecimento em Engenharia de Domínio, *XIII Simpósio Brasileiro de Engenharia de Software*, Florianópolis, outubro.

UCHÔA, E. M. A., MELO, R., 1999,. HEROSfw: a Framework for Heterogeneous Database Systems Integration, *DEXA Conference and Workshop Programme*, Florença, Itália.

VIEIRA, M., 1998, Abordagem para Apoio ao Teste Baseado no Comportamento de Sistemas Orientados a Objetos, Tese de Mestrado, COPPE/UFRJ, Maio.

WERNECK, V., 1995, Ambiente para Desenvolvimento de Software Baseado em Conhecimento, Tese de Doutorado, COPPE/UFRJ, Junho.

WERNER, C., BRAGA, R., MATTOSO, M., *et alli.*, 2000, Odyssey: Estágio Atual, *Caderno de Ferramentas do XV Simpósio Brasileiro de Engenharia de Software (XIV SBES)*, João Pessoa, Brasil, Outubro.

WERNER, C., MATTOSO, M., BRAGA, R., *et al.*, 1999, Odyssey: Infra-estrutura de Reutilização baseada em Modelos de Domínio, *Caderno de Ferramentas do XIII Simpósio Brasileiro de Engenharia de Software (XIII SBES)*, pp.17-20, Florianópolis, Brasil, Outubro.

WERNER, C.; BRAGA, R.; ZOPELARI, M.; et alli. 2000 a, Odyssey-LE: Uma Infra-Estrutura de Reutilização para o Domínio de Processamento Legislativo,V ENIAL - Encontro Nacional de Informática Aplicada ao Legislativo, Agosto, Vitória, ES (em CD-ROM).

WIEDERHOLD, G., 1999, *Future Trends in Integration of Information*, Journal of Intelligent Information Systems, Kluwer pubs., disponível em http://www-db.stanford.edu/pub/gio/paperlist.html.

WIEDERHOLD, G., GENESERETH, M., 1995, Basis for Mediation, *Proc. COOPIS'95 Conference*, Vienna Áustria, Maio, available from {coopis@cs.toronto.edu}, pp.138-155.

WIEDERHOLD, G., GENESERETH, M., 1997: The Conceptual Basis for Mediation Services; *IEEE Expert*, Vol.12 No.5, Sep.-Oct., pp. 38-47.

WIEDERHOLD, G.., 1994, An Algebra for Ontology Composition, *Proceedings of 1994 Monterey Workshop on Formal Methods*, Setembro.

WOOLDRIDGE, M, et alli, 2000, The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent* Systems. 3(3):285-312.

WOOLDRIDGE, M, JENNINGS, R., 1998, editors: Intelligent Agents IV Springer-Verlag Lecture Notes AI Volume 1365, Fevereiro.

WOOLDRIDGE, M., 1997, Agent –based Software Engineering, *IEEE Transactions on Software Engineering*, vol 144(1), pp.2-37, Fevereiro.

WOOLDRIDGE, M., 1998, Agent-based computing. *Interoperable Communication Networks*. 1(1), pp. 71-97. Janeiro.

XAVIER, J., 2000, *Uma Ferramenta de Apoio à Definição e Instanciação de Arquiteturas Específicas de Domínio*, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil (em andamento).

XHUMARI, F, AMZAL, M., SIMON, E.,1999, "Le Select: a Middleware System for Publishing Autonomous and Heterogeneous Information Sources", Inria, Grupo Caravel, França, disponível em http://www-caravel.inria.fr/~xhumari/LeSelect.

YACOUB, S., AMMAR, H., MILLI, A.,1999a., A Model for Classifying Component interfaces, *International Workshop on Component-Based Software Engineering*, Los Angeles, Estados Unidos, Maio.

YACOUB, S., AMMAR, H., MILLI, A.,1999b, Characterizing a Software Component, *International Workshop on Component-Based Software Engineering*, Los Angeles, Estados Unidos, Maio.

YE, Y.; FISCHER, G. , 2000, Promoting Reuse with Active Reuse Repository Systems, *IEEE ICSR 2000*, Viena, Junho.

ZOPELARI, M., 1998, *Uma Proposta de Sistemática para Aquisição de Conhecimento no contexto de Análise de Domínio*, Tese de M.Sc., COPPE/UFRJ, Novembro, Rio de Janeiro, Brasil.