COMPUBLISH: UM SISTEMA PARA A PUBLICAÇÃO, BUSCA E RECUPERAÇÃO DE COMPONENTES DE SOFTWARE NA INTERNET

Robson Pinheiro de Souza

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 MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovado por:

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

Profª. Cláudia Maria Lima Werner, D.Sc.

Profª. Maria Luiza Machado Campos, Ph.D.

Prof. Javam de Castro Machado, Dr.Sc.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 2002

SOUZA, ROBSON PINHEIRO DE

ComPublish: Um Sistema para a Publicação, Busca e Recuperação de Componentes de Software na Internet [Rio de Janeiro] 2002

X, 103 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2002)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Bancos de Dados Distribuídos e Heterogêneos

2. Mediadores

3. Publicação, Busca e Recuperação de Componentes de Software

4. Ambientes de Reutilização

I. COPPE/UFRJ II. Título (série)

AGRADECIMENTOS

À Deus, por me dar o direito a vida.

Aos meus pais, por terem me dado o apoio e as condições necessárias para estar hoje nesta universidade.

Em especial, às professoras Marta Mattoso e Cláudia Werner, pela orientação, pelo esforço, pelo incentivo e pela compreensão durante a longa caminhada deste trabalho.

À professora Maria Luiza e ao professor Javam Machado, por aceitarem participar desta banca.

A todos os professores da COPPE, em especial aos professores da linha de Banco de Dados e Engenharia de Software, por todos os ensinamentos passados em minha vida acadêmica.

A Regina Braga e Paulo Pires, pela amizade, pelos ensinamentos, pelo incentivo e pelos excelentes trabalhos de base dos quais se originou esta tese.

Aos amigos Nicolaas Ruberg, Marcelo Costa, Alessandreia de Oliveira, Maria Cláudia Cavalcanti e Márcio Victorino, pelos trabalhos conjuntos e pela troca de conhecimentos.

A todos os demais amigos da linha de Banco de Dados, Gabriela Ruberg, André Victor, Gustavo Pinto, Fernanda Baião, Humberto Vieira, Leonardo Guerreiro, Leonardo Cardoso, Rodrigo Reis, Carlete Ferreira, Fátima Cristina e a qualquer outro amigo que tenha esquecido de mencionar aqui, pela amizade, pela troca de conhecimentos e pelo convívio no dia a dia acadêmico.

Aos amigos da linha de Engenharia de Software e do Projeto Odyssey, Márcio Barros, Leonardo Murta, Alexandre Dantas, Gustavo Veronese, Hugo Vidal, Alexandre Correa, José Ricardo Xavier e Marcos Mangan, pela amizade, pelos momentos de descontração e por todas as contribuições e conhecimentos trocados.

Às secretárias Patrícia e Ana Paula, sempre simpáticas e dispostas a ajudar no que for possível.

À CAPES, pelo apoio financeiro.

À professora Marta Mattoso e ao professor Cláudio Amorim, por me concederem o privilégio de participar do Projeto Goa, junto aos colegas do Laboratório de Computação Paralela.

A Gabriela Ruberg, pelo meu futuro caminho profissional.

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

COMPUBLISH: UM SISTEMA PARA A PUBLICAÇÃO, BUSCA E RECUPERAÇÃO DE COMPONENTES DE SOFTWARE NA INTERNET

Robson Pinheiro de Souza

Dezembro/2002

Orientadores: Marta Lima de Queirós Mattoso
Cláudia Maria Lima Werner

Programa: Engenharia de Sistemas e Computação

Este trabalho apresenta o *ComPublish*, um sistema que visa auxiliar desenvolvedores de software a publicar variados artefatos de software na Internet, tal como modelos, diagramas, código fonte, documentos, programas ou qualquer outro tipo de artefato utilizado ou produzido nas diferentes etapas do processo de desenvolvimento de um software. Através de uma arquitetura baseada em mediadores, o *ComPublish* provê uma visão lógica de vários componentes publicados local ou remotamente dentro de um mesmo domínio de aplicação. Seus principais serviços são: (1) descrever e publicar componentes armazenados em um repositório qualquer da Internet; (2) integrar as informações dos componentes publicados de acordo com o seu domínio de aplicação; (3) prover serviços ontológicos capazes de relacionar informações de componentes publicados dentro de um mesmo ou em diferentes domínios de aplicação; (4) prover mecanismos de pesquisa e a recuperação dos componentes publicados.

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

COMPUBLISH: A SYSTEM FOR THE PUBLICATION, SEARCH AND RETRIEVAL OF SOFTWARE COMPONENTS ON THE INTERNET

Robson Pinheiro de Souza

December/2002

Advisors: Marta Lima de Queirós Mattoso
Cláudia Maria Lima Werner

Department: Systems and Computer Engineering

This work presents *ComPublish*, a system that aims to help software developers to publish diverse software artifacts on the Internet, such as models, diagrams, source code, documents, programs or any other artifact used or produced on different stages of the development process of a software. Through an architecture based on mediators, *ComPublish* provides a uniform view of local and remote components that belong to the same application domain. The main services of *ComPublish* are: (1) to describe and to publish components stored in a repository on the Internet; (2) to integrate the published component descriptions in the same domain application; (3) to provide ontology services to relate the component descriptions published in the same or different domains application; and (4) to provide search and retrieval mechanisms over the published components.

ÍNDICE ANALÍTICO

ÍNDICE DE FIGURAS

1 Introdução

1.1 - Motivações

Nos últimos anos, o grande avanço das redes de computadores fez crescer demasiadamente a quantidade de dados disponibilizados *on-line*. O surgimento de novas tecnologias, em especial a *World Wide Web* (WWW ou simplesmente Web), permite hoje que qualquer usuário de um computador ligado à Internet tenha acesso a informações eletrônicas dos mais variados tipos e tamanhos. Além disso, qualquer usuário é capaz de publicar sem muito esforço suas próprias informações, o que faz da Web atualmente a principal plataforma de distribuição de dados e aplicações. Em resumo, a Web é hoje um enorme banco de dados (BERNSTEIN et al., 1998).

Embora os grandes avanços tecnológicos (tais como redes de computadores mais rápidas e dispositivos de armazenamento de maior capacidade e velocidade) venham ajudando pesquisadores a construir mecanismos de busca e recuperação mais eficientes, a distribuição das fontes e a heterogeneidade dos dados ainda constituem uma grande barreira para o processo de localização e de integração de informações na Internet. Por isso, o acesso integrado a informações distribuídas e heterogêneas é um dos principais focos de pesquisa na área de banco de dados atualmente.

No início da década passada, os principais esforços estavam voltados para a investigação de técnicas capazes de integrar dados estruturados armazenados em sistemas de banco de dados autônomos, distribuídos e heterogêneos, surgindo os chamados Sistemas de Banco de Dados Distribuídos e Heterogêneos (SBDH) (THOMAS et al., 1990) (SHETH e LARSON, 1990) (DITTRICH e DOMENIG, 1999). Contudo, o grande crescimento da Internet proporcionou uma alta disseminação através da Web de dois novos tipos de informação: os dados não estruturados e os semi-estruturados. Dados não estruturados são aqueles que não possuem descrição alguma,

sendo formados essencialmente por texto. Um exemplo bastante comum na Web são os arquivos HTML (*Hypertext Markup Language*), acessados por qualquer *browser*. Já os dados semi-estruturados são aqueles cuja descrição e o valor da informação estão ambos presentes na sua representação. Nos últimos anos, XML (*Extensible Markup Language*) vem ganhando força como padrão de representação de dados semi-estruturados na Web, fornecendo mais recursos para melhor descrever diferentes tipos de informação. Atualmente, os *browsers* também já incluem a capacidade de visualizar arquivos XML.

No entanto, infelizmente, dados do tipo não estruturado, e até mesmo os semi-estruturados, contribuem bastante para tornar difícil a tarefa dos sistemas de busca de informação. O processamento de consultas realizado sobre páginas HTML, por exemplo, é executado sobre índices que apontam para algumas poucas palavras dentro do documento. XML, por sua vez, embora tenha embutido um esquema que descreve os dados, envolve ainda técnicas de linguagens de processamento de consultas de cerca de vinte e cinco anos atrás: a avaliação de consultas é feita, em geral, através de modelos de estruturas de árvore; a otimização, por sua vez, ou é ignorada ou é reduzida a poucas e simples heurísticas (BERNSTEIN et al., 1998).

Assim sendo, o processo de pesquisa de informações na Web, constituída em sua maior parte por dados não estruturados e semi-estruturados, é um processo bastante trabalhoso. Nos dias atuais, o estado da arte para o processamento e otimização de consultas conta ainda com as técnicas desenvolvidas para os modelos de dados estruturados. Por esse motivo, propostas como a do *XVerter* (VIEIRA, RUBERG, MATTOSO, 2002) e do sistema *Agora* (MANOLESCU, FLORESCU e KOSSMANN, 2000, 2001a e 2001b), que adotam uma solução híbrida para tratar da integração de informações na Web, apresentam-se como um caminho bastante viável. Estas propostas exploram a utilização de XML em todos os pontos de interface com usuário: dados de entrada, dados de saída e linguagem de consulta são todos descritos segundo o formato XML. No entanto, na abordagem do *XVerter*, os dados internos são todos convertidos para a forma de classes, enquanto na do *Agora*, para forma de tabelas, permitindo o uso de técnicas de processamento de consultas baseadas no modelo de dados orientado a objetos e relacional, respectivamente.

Um segundo ponto importante, cada vez mais evidenciado com o crescimento da Internet, está relacionado à distribuição e à quantidade de fontes de informação presentes na rede. Pesquisar informações na Web é um processo que conta com estruturas de índices que endereçam um vasto número de fontes de informação encontradas em variados sítios espalhados pelo mundo. No entanto, a efetividade destes índices é bastante dependente da forma com que estas fontes são descritas por seus provedores. Neste contexto, destaca-se a importância do papel desempenhado pelos metadados, dados que descrevem dados. A utilização de metadados permite prover uma melhor descrição das fontes de informação, enriquecendo o conteúdo dos dados e serviços publicados, ao mesmo tempo que ajuda os sistemas de busca a localizar, descrever e selecionar repositórios de dados na rede. Metadados são utilizados ainda para uma série de outras funcionalidades, como por exemplo, em catálogos para sistemas gerenciadores de bancos de dados e sistemas de integração de dados na Internet, em tarefas de administração de dados, no suporte a configuração de sistemas, dentre outras atividades (MOURA, CAMPOS e BARRETO, 1999). Atualmente, XML vem se popularizando como um padrão promissor de formato de escrita e armazenamento de metadados, com o apoio de vários grupos internacionais de desenvolvedores: *World Wide Web Consortium* (W3C, 2002a), *Object Management Group* (OMG), *Meta Data Coalition* (MDC), etc.

Uma outra questão importante, atualmente também foco de pesquisa em sistemas gerenciadores de banco de dados (SGBDs), diz respeito a necessidade destes sistemas em prover uma maior integração entre **dados e programas** (BERNSTEIN et al., 1998). Há algum tempo atrás, SGBDs eram construídos apenas para armazenar dados, deixando a parte lógica de programas para outros sub-sistemas. Mais recentemente, por uma questão de desempenho, SGBDs Relacionais adicionaram o conceito de *stored procedures* e *triggers*, permitindo a incorporação interna de programas desenvolvidos em uma linguagem proprietária. As tecnologias emergentes dos SGBDs Orientados a Objeto e Objeto-Relacional têm também como um dos principais objetivos facilitar a incorporação interna da lógica de programas, através da utilização de uma linguagem padrão tal como C ou *Java*. No entanto, embora SGBDs tratem com sucesso a integração de dados estruturados, a integração entre dados e programas ainda é um problema em aberto. Além disso, o grande crescimento da

Internet traz à tona um novo conjunto de problemas e necessidades. Atualmente, a Internet é composta por milhares de repositórios que armazenam tanto dados quanto programas. Este novo cenário, caracterizado sobretudo pela alta distribuição e heterogeneidade das informações, gera uma série de problemas de interoperabilidade entre dados e programas, o que exige dos sistemas de banco de dados novos requisitos capazes de tratar estes problemas de forma transparente para o usuário. Dentro deste contexto, a proposta do *Le Select* (XHUMARI e MOKRANE, 1999) se destaca por ser um sistema que trata da integração entre dados e programas distribuídos. No entanto, a utilização de recursos de metadados, no que diz respeito à busca de informações e à integração semântica de dados, é pouco trabalhada por este sistema.

Portanto, para prover uma integração mais geral e melhorar a qualidade da busca de informações na Internet, surge a necessidade de uma nova proposta de arquitetura de integração de dados capaz de: (1) prover um mecanismo de integração que contemple tanto fontes de dados estruturados (constituídas por sistemas gerenciadores de bancos de dados tradicionais), quanto fontes de dados semi-estruturados e não estruturados (constituídas por um sistema gerenciador de dados qualquer); e (2) prover mecanismos de pesquisa mais específicos para busca de informações heterogêneas e distribuídas.

Atualmente, as necessidades listadas acima são questões de pesquisa que não se restringem apenas aos interesses da área de Banco de Dados. Dentro da área de Engenharia de Software, por exemplo, tais necessidades podem ser mapeadas para o contexto de Reutilização de Software, onde processos típicos de reutilização envolvem o armazenamento e a recuperação de recursos bastantes heterogêneos, tais como documentos, modelos, diagramas, programas, dentre outros. Todos estes recursos, denominados de **artefatos de software**[1], são hoje em dia freqüentemente publicados na Internet. Portanto, a criação de um sistema de integração de informações voltado para Internet, de modo a facilitar a busca, identificação e seleção de diferentes artefatos de software, pode contribuir bastante para o sucesso de um processo de reutilização.

---

[1] Muitos autores associam a palavra componente a código. Por isso, o termo artefato de software foi criado para representar genericamente qualquer recurso produzido ou reutilizado em um processo de desenvolvimento de software, tais como documentos, modelos, diagramas, código, etc. Nesta dissertação, apenas por uma questão de simplicidade, será utilizado apenas a palavra componente, porém referindo-se sempre ao termo artefato de software.

Dentro deste contexto, pesquisadores em Reutilização de Software identificam diferentes aspectos técnicos que devem ser levados em conta para o sucesso da reutilização:

- Pesquisadores como ARANGO (1988), SAMETINGER (1997) e SEACORD (1998 e 1999) ressaltam a importância da utilização de repositórios de componentes como uma base preparada para o armazenamento, seleção e obtenção de componentes.

- KRUEGER (1992) e MILLI et al. (1995) apontam a abstração como um mecanismo importante para facilitar a busca por componentes. Eles defendem a idéia de que a organização de conjuntos de componentes reutilizáveis em bibliotecas de componentes divididas por domínio de aplicação (onde cada domínio representa um nível de abstração) reduz o universo de componentes a ser consultado, permitindo que as abstrações possam ser capturadas de forma mais precisa, o que torna a recuperação de componentes mais fácil.

- MILLI et al. (1995) ressalta também a importância de uma documentação relacionada a cada componente a ser reutilizado. Para isso, MILLI defende a idéia da utilização de um vocabulário familiar ao reutilizador e de uma estrutura de fácil entendimento para representar estas informações.

- BRAGA (2000b), em sua tese de doutorado, reforça ainda a importância do tratamento de questões como a distribuição e a interoperabilidade de repositórios na Internet. Além disso, BRAGA defende a idéia do uso de técnicas que levam em conta o comportamento do usuário em relação às suas buscas passadas e atuais (perfil do usuário), e de técnicas de Inteligência Artificial (agentes inteligentes) para melhorar a qualidade do processo de recuperação de componentes.

Em resumo, processos de reutilização de software necessitam de mecanismos mais específicos para a busca e recuperação de componentes. Estes mecanismos devem levar em conta, dentre outras questões, o domínio de aplicação do componente, as

informações de documentação relacionadas ao componente, e questões de distribuição e heterogeneidade dos componentes publicados remotamente na Internet.

1.2 - Objetivos

Esta dissertação tem por objetivo a definição e implementação de um sistema capaz de: (1) integrar diversas informações de componentes de software reutilizáveis publicados em repositórios acessados pela Internet; e (2) prover funcionalidades para pesquisa, recuperação e armazenamento local destes componentes. Esta dissertação, diferente da maioria de outros trabalhos, considera que componentes reutilizáveis não são apenas componentes do tipo código, mas também estruturas de projetos, especificações, modelos, diagramas, documentos, ferramentas auxiliares, programas de teste, isto é, todo e qualquer produto (seja ele dado ou serviço) criado ou utilizado durante um processo de desenvolvimento de software (KRUEGER, 1992).

Para atingir os objetivos apresentados acima, este sistema, daqui em diante denominado *ComPublish*, foi projetado segundo uma arquitetura de camadas baseada na tecnologia de mediadores e tradutores (WIEDERHOLD, 1992, 1999). Esta tecnologia facilita que componentes possam ser agrupados segundo o conceito de domínio de aplicação, como sugerido por KRUEGER (1992) e MILLI et al. (1995), na medida em que cada mediador tem por finalidade prover uma visão lógica de dados agrupados por domínio. A fim de incentivar a atividade de documentação de componentes, ajudando o seu entendimento, e prover recursos para se desenvolver mecanismos de pesquisa de melhor qualidade, o *ComPublish* incorpora a idéia de utilização de metadados referentes aos componentes publicados remotamente. Para descrever estes metadados, foi utilizada a tecnologia XML, por ser uma estrutura de fácil representação, flexível, de fácil conversão para outras estruturas e bastante promissora para se tornar um padrão de documentação de componentes na Internet. Para armazenar e gerenciar estes metadados, ou até mesmo componentes armazenados localmente, cada mediador utiliza uma base de dados gerenciada por um SGBD Orientado a Objetos, que provê um modelo de representação poderoso capaz de gerenciar dados heterogêneos e, às vezes, bastante complexos, características dos componentes de software. O módulo tradutor, por sua

vez, é responsável por prover o acesso aos repositórios remotos. Cada mediador importa dos tradutores aos quais ele está ligado os metadados publicados, armazenando apenas os metadados referentes a componentes relativos a seu domínio de aplicação. Internamente a cada mediador, os metadados são convertidos de XML para o modelo orientado a objetos, permitindo, desta forma, uma melhoria na eficiência do processamento de consultas e na precisão dos componentes retornados, através da utilização de uma linguagem de consulta segundo padrão OQL.

O desenvolvimento do *ComPublish* é baseado no uso de outras arquiteturas e produtos em desenvolvimento: (1) a publicação de dados e serviços remotos é realizada pelo sistema *Le Select* (XHUMARI e MOKRANE, 1999); (2) a camada de integração, baseada no conceito de mediadores e tradutores, implementa uma evolução da arquitetura HIMPAR (PIRES, 1997); e (3) o armazenamento de metadados e componentes, e o processamento de consultas (segundo o modelo orientado a objetos) locais a um mediador são feitos pelo sistema GOA (MATTOSO et al., 1994, 2000 e 2002) (MAURO, 1997). A fim de integrar e adaptar estas diferentes ferramentas, de modo a possibilitar a construção de uma grande biblioteca digital distribuída de componentes de software reutilizáveis acessível através da Internet, um grande volume de implementação foi realizado sobretudo sobre os protótipos das arquiteturas HIMPAR e GOA, de modo a torná-los mais flexíveis, robustos e extensíveis, além de adaptá-los para trabalhar com a nova tecnologia XML (MATTOSO et al., 2002).

Os serviços oferecidos pelo *ComPublish* (SOUZA et al., 2000, 2001 e 2002) são atualmente utilizados pelo Ambiente *Odyssey* (WERNER et al., 1999, 2000 e 2002), que visa prover suporte ao desenvolvimento e reutilização de componentes em todas as fases de construção de um software. O *ComPublish* constitui uma extensão da *Odyssey Search Engine* (BRAGA, 2000b), visando prover soluções mais abrangentes para os problemas de publicação e busca de componentes na Internet. No entanto, é importante ressaltar que este é um sistema totalmente independente do *Odyssey*, permitindo que todos os seus serviços possam ser acessados, por exemplo, via um simples *browser*.

1.3 - Organização dos Capítulos

O restante desta dissertação encontra-se organizado como descrito a seguir:

⇒ O Capítulo 2 apresenta as principais tecnologias de integração de fontes de informação distribuídas e heterogêneas existentes hoje na área de Banco de Dados, descrevendo os principais conceitos, problemas e trabalhos desenvolvidos sobre este assunto;

⇒ O Capítulo 3 aborda o assunto DBC - Desenvolvimento Baseado em Componentes, apresentando uma lista de características, problemas e trabalhos relacionados ao desenvolvimento com reutilização. Ao fim, é feita uma análise crítica dos trabalhos encontrados na literatura, e é apresentado o diferencial do trabalho proposto nesta dissertação;

⇒ O Capítulo 4 apresenta os detalhes do projeto do sistema *ComPublish*, descrevendo ainda o seu papel dentro do contexto do projeto *Odyssey*;

⇒ O Capítulo 5 descreve os detalhes de implementação do sistema *ComPublish*, apresenta um exemplo de sua utilização, e faz uma análise final dos benefícios obtidos pela utilização do *ComPublish* na busca e recuperação de componentes, comparado aos tradicionais mecanismos de busca da Web;

⇒ Por fim, o Capítulo 6 encerra esta dissertação fazendo uma síntese e análise das principais contribuições obtidas, apontando suas limitações e identificando os trabalhos futuros.

2 Sistemas de Integração de Fontes de Dados Heterogêneas e Distribuídas

2.1 - Introdução

De acordo com SIMON e TOMASIC (1997), muitas atividades de domínios científicos são formadas por usuários provedores de dados, usuários consumidores de dados e usuários intermediários, que atuam em ambos papéis. Usuários provedores buscam tornar seus dados públicos, enquanto que usuários consumidores necessitam localizar estes conjuntos de dados, de acordo com um determinado critério, a fim de visualizá-los, consultá-los, e eventualmente extrair alguma informação de seu interesse.

No entanto, duas dificuldades principais são encontradas quando busca-se desenvolver uma ferramenta de suporte ao compartilhamento de dados necessário às atividades científicas: (i) Heterogeneidade dos dados - os dados a serem buscados podem estar guardados nas mais variadas formas de armazenamento (sistemas de arquivos, banco de dados, etc), além de se apresentarem em diferentes tipos e formatos (texto, tabelas, objetos, etc), e possuírem diferentes capacidades de consulta; (ii) Distribuição dos dados - os dados encontram-se armazenados em diferentes servidores espalhados pela rede, com diferentes protocolos de acesso necessários para enxergá-los.

A necessidade de integração de diversas e diferentes fontes de informação é atualmente uma crescente realidade dentro do ambiente computacional, sobretudo com a grande expansão da Internet. Este problema pode ser encontrado em uma larga variedade de domínios de aplicações, destacando-se, em particular, o domínio de desenvolvimento de aplicações científicas, que é por natureza multi-disciplinar, além de envolver muitas organizações e instituições autônomas (VALDURIEZ et al., 2000 e 2001). Neste contexto, a utilização de metadados que capturam a semântica de tais recursos científicos é um fator essencial para garantir a qualidade do processo de

integração de informações altamente heterogêneas e distribuídas (CAVALCANTI et al., 2002a e 2002b).

Sendo assim, este capítulo descreve, de uma forma geral, variados problemas envolvidos na construção de sistemas de bancos de dados heterogêneos e distribuídos, e diferentes propostas encontradas na literatura para tentar solucioná-los. No capítulo 3, por sua vez, é descrita a solução que melhor se adapta ao contexto dos problemas relacionados à integração de repositórios de componentes de software reutilizáveis, e como ela foi aplicada na arquitetura proposta por esta dissertação.

O restante do capítulo é organizado como a seguir: a seção 2.2 apresenta uma classificação bastante abrangente, proposta por DITTRICH e DOMENIG (1999), que descreve diferentes abordagens para tratar do problema de integração de fontes de informações heterogêneas e distribuídas. A seção 2.3 descreve uma série de trabalhos encontrados na literatura relacionados ao assunto. Por fim, a seção 2.4 conclui o capítulo apresentando um resumo e as observações finais do exposto nas seções anteriores.

2.2 - Classificação

Existem atualmente na literatura variadas soluções para tratar do problema da integração de dados heterogêneos e distribuídos. Diferentes grupos de pesquisadores classificam estas soluções segundo um determinado conjunto de aspectos. Em (SHETH e LARSON, 1990), por exemplo, é apresentada uma classificação que leva em consideração o grau de autonomia das fontes locais (alto ou baixo) e o nível de acoplamento existente entre os componentes da camada de integração do sistema (forte ou fraco). Mais recentemente, DITTRICH e DOMENIG (1999) apresentaram uma classificação que leva em consideração a forma com que os dados são integrados (materializados ou virtuais), a origem dos dados (nativos ou derivados) e a estrutura dos dados integrados (estruturados, semi-estruturados ou não estruturados). Por ser esta uma classificação bastante abrangente e intuitiva, esta dissertação segue as idéias propostas por ela, como detalhadas a seguir.

Primeiramente, a classificação proposta por DITTRICH e DOMENIG (1999) (figura 2.1) divide os sistemas de integração em dois grandes grupos: (1) os que materializam os dados; (2) os que integram os dados virtualmente. Na primeira abordagem, os dados das fontes locais são integrados dentro de um novo banco de dados, ao contrário da segunda, onde os dados permanecem nas fontes de origem.

Em um nível mais baixo, a classificação divide os sistemas que materializam dados em dois subgrupos: Banco de Dados Universais e Data Warehouses. Já os sistemas que integram dados virtualmente são divididos em três diferentes subgrupos: Máquinas de Busca, Bancos de Dados Federados e Sistemas de Consulta Mediados. Neste nível, são levados em conta a origem (nativos ou derivados)[2] e a natureza estrutural dos dados integrados (estruturados, semi-estruturados ou não estruturados).

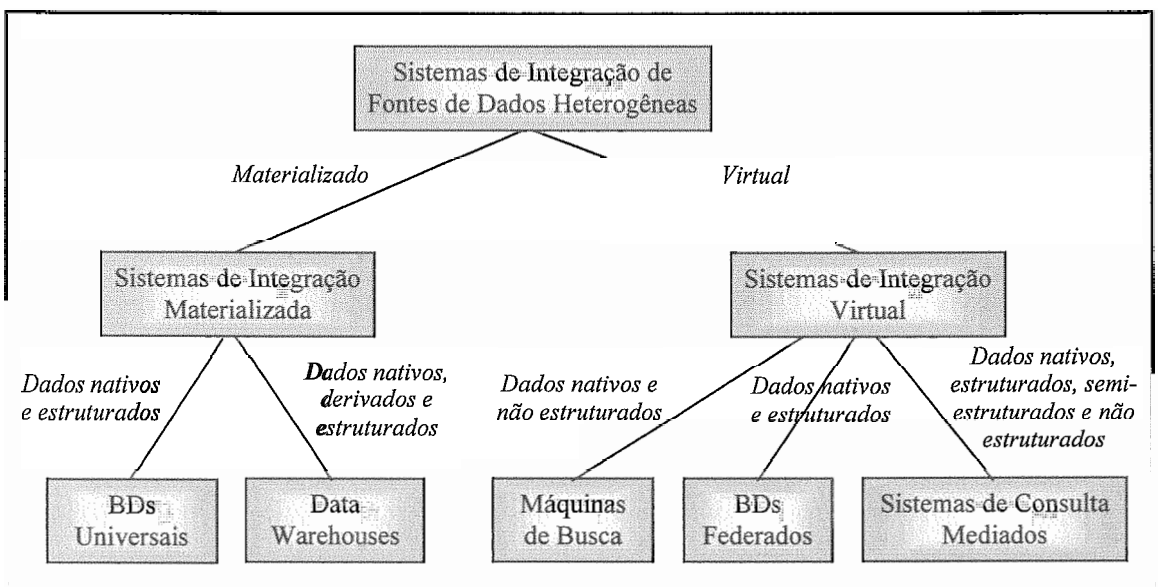


Figura 2.1 - Classificação de Sistemas de Integração de Fontes Dados Heterogêneas

É importante ressaltar que esta classificação é ainda bastante flexível, podendo-se encontrar na literatura sistemas que misturam aspectos de diferentes abordagens. Por exemplo, existem sistemas que podem ser classificados como Bancos de Dados Federados pelo fato de integrarem apenas fontes de dados estruturadas, mas que, no entanto, utilizam a tecnologia de mediadores para a tarefa de integração de dados. Além

[2] Dados são classificados como nativos se eles são recuperados utilizando o mesmo conteúdo dos dados armazenados na fonte local. Já dados derivados são aqueles que são recuperados a partir de alguma transformação aplicada sobre os dados originais da fonte local.

disso, é possível também se encontrar na literatura pequenas variações com outros nomes para cada uma das cinco abordagens citadas. Por exemplo, o conceito de *Middlewares* se confunde bastante com o conceito de Mediador. Não existe uma definição clara entre os pesquisadores para distinguir ambas abordagens. De uma forma geral, um *middleware* pode ser visto como um sistema mais voltado para tratar as questões de interoperabilidade de acesso a fontes de dados heterogêneas, sem se preocupar com a integração semântica dos dados. O *middleware* facilita a publicação de dados e serviços em um ambiente distribuído e heterogêneo, sem definir um esquema global e sem prover mecanismos para garantir transparência quanto à distribuição física das fontes. Como, por exemplo, podemos citar o *Le Select*, descrito na seção 2.3.3.

Nas seções a seguir, são descritas as principais caraterísticas de cada uma das cinco categorias de sistemas mencionadas na figura 2.1. Dentre elas, um enfoque maior é dado aos Sistemas de Consulta Mediados, abordagem esta adotada para o sistema proposto nesta dissertação.

2.2.1 – Bancos de Dados Universais

Nesta proposta de integração, dados são migrados das fontes locais para um Banco de Dados "Universal" (BDU), que deverá ser capaz de gerenciar uma grande variedade de tipos de informações. Assim, o modelo de dados suportado pelo BDU deverá ser semanticamente rico, sendo os SGBDs Relacionais Objeto e Orientado a Objetos os mais indicados para isso. Uma vez que os dados das fontes locais sejam extraídos, integrados e armazenados em um SGBD central, os sistemas locais são desativados.

Esta abordagem pode ser uma boa solução para usuários ou aplicações que necessitam de todas as funcionalidades de um SGBD. No entanto, as aplicações produzidas para os sistemas locais precisam ser rescritas para o novo SGBD. Além disso, a migração dos dados pode se tornar um processo bastante caro, pois os dados locais precisam ser transformados e freqüentemente enriquecidos para o modelo canônico suportado pelo BDU.

2.2.2 – Data Warehouses

Semelhante a proposta anterior, esta abordagem consiste em migrar os dados das fontes locais para um BD central, o *Data Warehouse* (DW). No entanto, diferentemente dos DBUs, as fontes locais permanecem em operação. Logo, os dados são efetivamente replicados.

No DW, os dados importados nem sempre são do mesmo volume e conteúdo dos dados locais. Estes passam por um processo de extração, "limpeza", transformação e integração, sendo normalmente preparados para uso de uma aplicação específica (CAMPOS e BORGES, 2002). Isto ajuda bastante os desenvolvedores de aplicações, que passam a ter um conhecimento prévio da estrutura dos dados que irão trabalhar. No entanto, a atividade de migração dos dados é normalmente um processo bastante custoso, consumindo em média 60% do tempo do projeto de um DW (BECKER e PEREIRA, 1999). Além disso, os DWs apresentam o problema de nem sempre oferecerem os dados mais recentemente avaliados, na medida que um DW não é imediatamente atualizado após alguma mudança numa fonte local. Apesar disso, uma vez atualizados, os DWs são capazes de armazenar um histórico de atualização dos dados, que são úteis, por exemplo, em aplicações de mineração de dados (*data mining*).

2.2.3 – Máquinas de (Meta)Busca

Graças ao grande crescimento da WWW, as Máquinas de Busca são hoje os sistemas de consulta mais populares da Internet. Estas permitem consultar fontes de dados distribuídas e homogêneas, constituídas essencialmente de arquivos HTML (dados não estruturados). Devido a ausência de um esquema para os dados publicados, as consultas formuladas são pouco precisas, sendo executadas a partir da busca de palavras chave dentro dos documentos HTML.

A principal desvantagem desta abordagem está ligada ao volume e à imprecisão dos dados recuperados por uma consulta. Muitas informações irrelevantes são retornadas ao usuário, o que dificulta a aquisição do conhecimento procurado. A fim de aumentar a eficiência de recuperação, uma nova abordagem conhecida como Máquina

de Metabusca tem sido proposta. Nestes sistemas, consultas são enviadas para diferentes máquinas de busca, e os resultados retornados são apresentados ao usuário através de uma lista ordenada a partir da relevância das informações encontradas (técnicas conhecidas como *Ranked List* e *Relevance Feedback* são utilizadas para tal propósito). Como exemplos de Máquinas de Metabusca pode-se citar o *SavvySearch* (DREILINGER e HOWE, 1997), o *MetaCrawler* (SELBERG e ETZIONI, 1997) e o *Informia* (BARJA et al., 1998).

2.2.4 – Bancos de Dados Federados

São sistemas que visam a integração de vários SGBDs individuais, distribuídos e heterogêneos. O Banco de Dados Federado (BDF) fornece ao usuário a noção de estar trabalhando em um SGBD local. Para isso, o sistema provê uma visão uniforme dos dados, através de um modelo global de dados e de uma linguagem de consulta global. Os BDFs buscam tornar transparente ao usuário os problemas de semântica e heterogeneidade entre os modelos de dados das bases locais, além de todo o processo de interação com cada SGBD participante da federação. Além disso, os BDFs buscam prover todas as funcionalidades típicas de um SGBD tradicional, dando suporte a consultas precisas, a operações de inserção e atualização, a mecanismos de transações, etc.

De forma diferente das duas primeiras abordagens, os BDFs não materializam os dados das fontes locais. Estes permanecem gerenciados por cada SGBD local, responsável por processar as consultas provenientes do sistema central. Isto traz a vantagem de que os dados recuperados por uma consulta global serão sempre os mais atuais, no entanto, o processamento global de consultas pode ter o desempenho bastante reduzido.

Os BDFs são sistemas que já vêm sendo pesquisados há vários anos. Atualmente, é possível encontrar no mercado alguns produtos comerciais, tais como o *DataJoiner* da IBM (DATAJOINER, 1997) e o *Miracle* da *Oracle* (HUGHES, 1996), dentre outros.

2.2.5 – Sistemas de Consulta Mediados

São sistemas que permitem consultar fontes de dados distribuídas e heterogêneas. Diferentemente dos BDFs, um Sistema de Consulta Mediado (SCM) não provê as típicas funcionalidades de um SGBD, ou seja, este tipo de sistema, semelhante a uma Máquina de Metabusca, é projetado para suportar essencialmente consultas. Além disso, são capazes de gerenciar os três tipos estruturais de dados, ou seja, dados estruturados, semi-estruturados e não estruturados. Sendo assim, os SCMs permitem ao usuário executar tanto consultas precisas (formuladas com base nas informações dos esquemas das bases de dados estruturadas), quanto consultas imprecisas (formuladas através do uso de palavras chave, devido à ausência de esquema das bases de dados não estruturadas).

Nos tópicos a seguir, são descritos, em maiores detalhes, os principais aspectos envolvidos na construção desta proposta de sistema.

Arquitetura

Seguindo proposta apresentada em (WIEDERHOLD, 1992 e 1999), os SCMs são projetados como uma arquitetura de três camadas (figura 2.2):

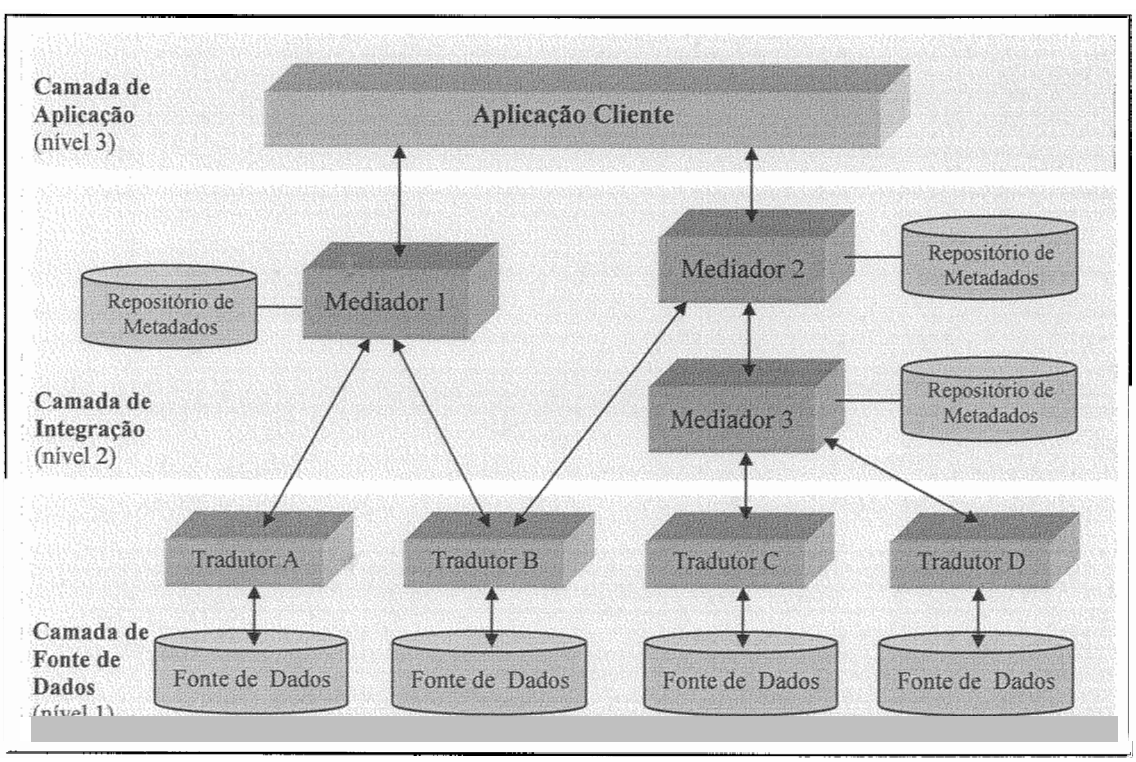


Figura 2.2 - Arquitetura de Sistemas de Consulta Mediados.

- Camada de Fonte de Dados – corresponde ao nível mais baixo da arquitetura (nível 1), sendo constituída pelas fontes de dados e pelos tradutores. O tradutor é o componente do sistema que se conecta à fonte de dados para exportar suas funcionalidades e dados. Este é responsável por todo o trabalho de conversão de informações do modelo de dados da fonte local para o modelo de dados suportado pelo mediador na camada superior. Além disso, se a fonte de dados local não possui capacidade própria para o processamento de consultas, cabe ao tradutor prover também esta funcionalidade.

- Camada de Integração – corresponde ao nível intermediário da arquitetura (nível 2), sendo constituída pelos mediadores. O mediador é um componente de software que explora o conhecimento representado em um conjunto de dados, a fim de gerar informações para aplicações residentes em uma camada superior. Cada mediador encapsula a representação de múltiplas bases de dados, tratando os problemas de conflitos semânticos gerados por diferentes modelos de dados e diferentes esquemas, e provendo um acesso uniforme aos dados por ele integrados. Para isso, cada mediador possui um repositório de metadados, que descreve as informações necessárias à integração dos dados envolvidos no seu domínio de aplicação. O usuário global é capaz de submeter consultas ao sistema via mediador, utilizando para isso uma linguagem de consulta global. Quando uma consulta é submetida ao mediador, esta é decomposta automaticamente em subconsultas a serem enviadas aos repositórios corretos; os resultados parciais, uma vez retornados ao mediador, são empacotados e devolvidos ao usuário global, para o qual estes passos são totalmente transparentes.

- Camada de Aplicação – corresponde ao topo da arquitetura (nível 3), que é constituída pelas aplicações clientes. Para executar uma pesquisa, o usuário global utiliza uma aplicação cliente para se conectar a um mediador e submeter sua consulta. Os dados recuperados em diferentes fontes são então retornados ao usuário de maneira uniforme, segundo o modelo de dados de exportação do mediador.

Extensibilidade

Uma característica importante dos SCMs é a sua capacidade de extensão, visto que fontes de dados podem ser adicionadas bem como removidas a qualquer instante do sistema. Por isso, é desejável que o sistema forneça flexibilidade para se adicionar novos metadados à camada de integração, bem como tente facilitar, na medida do possível, o trabalho de publicadores na escrita de tradutores para novas fontes de dados. No entanto, este é um aspecto que depende bastante da forma que o sistema é projetado. Neste sentido, duas principais questões devem ser levadas em conta:

- Número de mediadores – a camada de integração pode ser projetada contendo um (forma centralizada) ou múltiplos mediadores (forma descentralizada). Na proposta descentralizada, grupos de mediadores cooperam entre si, comunicando-se uns com os outros, a fim de processar diferentes partes de uma consulta submetida por um usuário. Os mediadores podem estar sendo executados numa mesma máquina ou em diferentes máquinas conectadas em rede. Cada mediador integra virtualmente um conjunto de fontes de dados voltados para um determinado domínio de aplicação. Este tipo de abordagem permite que mediadores (domínios de aplicação) sejam adicionados ou removidos do sistema com um pequeno esforço, o que garante uma boa extensibilidade. Já na proposta centralizada, um único mediador é utilizado para integrar todas as fontes e processar as consultas. Nesta abordagem, não há comunicação entre diferentes mediadores, o que garante um processamento de consulta mais rápido. No entanto, o sistema deixa de ser facilmente estendido como na abordagem anterior.

- Funcionalidades de mediadores e tradutores – duas abordagens diferentes podem ser utilizadas na divisão das funcionalidades entre mediadores e tradutores. A primeira delas consiste em construir tradutores "gordos". Nesta proposta, cada tradutor é responsável por executar a seguinte seqüência de passos: (1) receber as consultas expressas na linguagem de consulta global; (2) convertê-las para o modelo da linguagem de consulta local; (3) recuperar os dados locais; e (4) devolvê-los para o mediador no modelo de dados global. Em resumo, tradutores "gordos" desempenham todas as adaptações necessárias entre

o modelo de dados local da fonte e modelo de dados global do mediador. A principal vantagem desta abordagem é que o processamento de consultas no mediador se torna mais simples e rápido, consistindo basicamente em: (1) quebrar a consulta global em subconsultas (ainda expressas na linguagem de consulta global) a serem enviadas para cada tradutor; e (2) realizar as traduções semânticas necessárias (conversão de nomes e tipos) baseado nas informações (metadados) armazenadas de cada fonte de dados. No entanto, a extensibilidade neste caso fica comprometida, pois sempre que uma nova fonte é adicionada, muitas funcionalidades terão que ser implementadas no novo tradutor. Já a segunda proposta consiste em construir tradutores "magros". Neste caso, as funcionalidades de traduções passam a ser executadas no mediador, o que aumenta a extensibilidade do sistema, visto que a tarefa de escrever novos tradutores para novas fontes de dados se torna mais simples. No entanto, isto implica em um mediador bem mais "gordo", pois este passa a ter que cobrir uma ampla variedade de modelos e linguagens, além de dificultar um processamento de consulta eficiente na camada de integração.

Representação Interna dos Dados

O modelo de dados interno da camada de integração deverá ser definido de acordo com a estrutura de dados que o SCM irá manipular. Em geral, como estes sistemas são desenvolvidos para integrar dados heterogêneos, um modelo de dados interno mais complexo (por exemplo, o relacional-objeto ou o orientado a objetos) será o mais indicado para isso, a fim de que este consiga representar internamente qualquer tipo de informação.

Metadados

Essenciais ao funcionamento de todo o sistema, principalmente ao processamento de consultas, diversos metadados são armazenados em repositórios internos do sistema a fim de descrever: (1) informações variadas, colhidas das fontes de dados, cujo conteúdo contém o conhecimento, por exemplo, de localização, esquema local (caso exista), capacidades, custos, etc; (2) informações relacionadas aos esquemas

globais dos mediadores; (3) informações de mapeamentos entre os esquemas globais e locais; (4) informações de conhecimento ontológicos[3]; etc.

Capacidade e Processamento de Consultas

Devido à ampla variedade de dados integrados, os SCMs devem ser capazes de suportar tanto consultas precisas quanto consultas imprecisas. Neste sentido, os seguintes aspectos devem ser levados em conta:

- Dependência de esquema – as consultas submetidas ao SCM podem ou não ser dependentes de esquema global. No caso de dependência, o usuário necessariamente tem que expressar sua consulta em termos do esquema global. No entanto, se o esquema global não é necessariamente exigido, o SCM deve permitir ao usuário executar consultas imprecisas. Neste caso, o sistema deve oferecer, por exemplo, a possibilidade de pesquisar os valores de campos de um banco de dados relacional sem, no entanto, ter que especificar o nome de campo.

- Linguagem de consulta – os SCMs devem combinar as técnicas provenientes de dois tipos de linguagens: (1) uma linguagem de consulta semelhante a de banco de dados, onde as informações são recuperadas a partir de nomes de atributos, tipos e suas operações; (2) uma linguagem de consulta semelhante a de máquinas de busca, onde as consultas são expressas pela combinação de palavras chave e operadores booleanos (*or*, *and* e *not*).

- Apresentação dos resultados – dependendo da forma que o SCM é consultado (consultas precisas ou imprecisas), um suporte extra para apresentação dos resultados deve ser provido. No caso de consultas precisas, uma simples listagem dos resultados é suficiente. No entanto, se a consulta executada for imprecisa, muitos resultados podem ser produzidos. Neste caso, os resultados retornados devem ser apresentados ao usuário através de uma lista ordenada a partir da relevância das informações encontradas, utilizando para tal fim técnicas conhecidas como *Ranked List* e *Relevance Feedback* (YATES e NETO, 1999).

---

[3] Uma ontologia corresponde a uma descrição de um determinado conceito. É utilizada para associar informações similares representadas por diferentes vocabulários (por exemplo, "cão" e "cachorro" representam conceitos similares).

Uma vez definidas as características de consulta, a próxima etapa consiste em definir o plano de seu processamento. De uma forma geral, este segue os seguintes passos:

- Decomposição da consulta – consiste em decompor a consulta global expressa segundo o esquema de dados do mediador em uma lista de consultas baseadas nos esquemas de dados das fontes locais.

- Conversão de atributos – consiste em conveter os nomes dos atributos das subconsultas geradas na etapa de decomposição do escopo do mediador para o escopo das fontes locais.

- Otimização da consulta – visa encontrar possíveis dependências entre as subconsultas geradas, de modo a formular um plano de processamento paralelo que maximize na medida do possível a cooperação entre as fontes locais.

- Tradução e execução das subconsultas – consiste em traduzir as subconsultas do modelo global do mediador para o modelo das respectivas fontes locais, permitindo assim o processamento local.

- Empacotamento dos resultados – os resultados retornados por cada fonte local ao mediador devem ser convertidos para o modelo global de dados, agrupados e então retornados ao usuário.

2.3 – Trabalhos Relacionados

Uma grande variedade de trabalhos de pesquisa pode ser encontrada na literatura com enfoque na integração de fontes de dados heterogêneas e distribuídas. Na seção anterior, foram mencionados alguns deles. Nesta seção, são apresentadas as características mais relevantes de cinco projetos importantes: DISCO, HIMPAR, *Le Select, Agora* e MIX.

2.3.1 – DISCO

O DISCO – "*Distributed Information Search Component*" (TOMASIC, RASCHID e VALDURIEZ, 1998) (DISCO) é um sistema de integração de dados que tem por objetivo conectar um grande número de fontes de dados heterogêneas e distribuídas. Sua arquitetura de integração é baseada no conceito de Mediadores e Tradutores (WIEDERHOLD, 1992 e 1999). O modelo de dados interno à camada de mediação segue o padrão ODMG (CATTEL, 1993), utilizando extensões da linguagem ODL para a definição de dados e a linguagem OQL para a consulta aos dados.

A figura 2.3 ilustra a arquitetura DISCO, cujos componentes são detalhados a seguir.

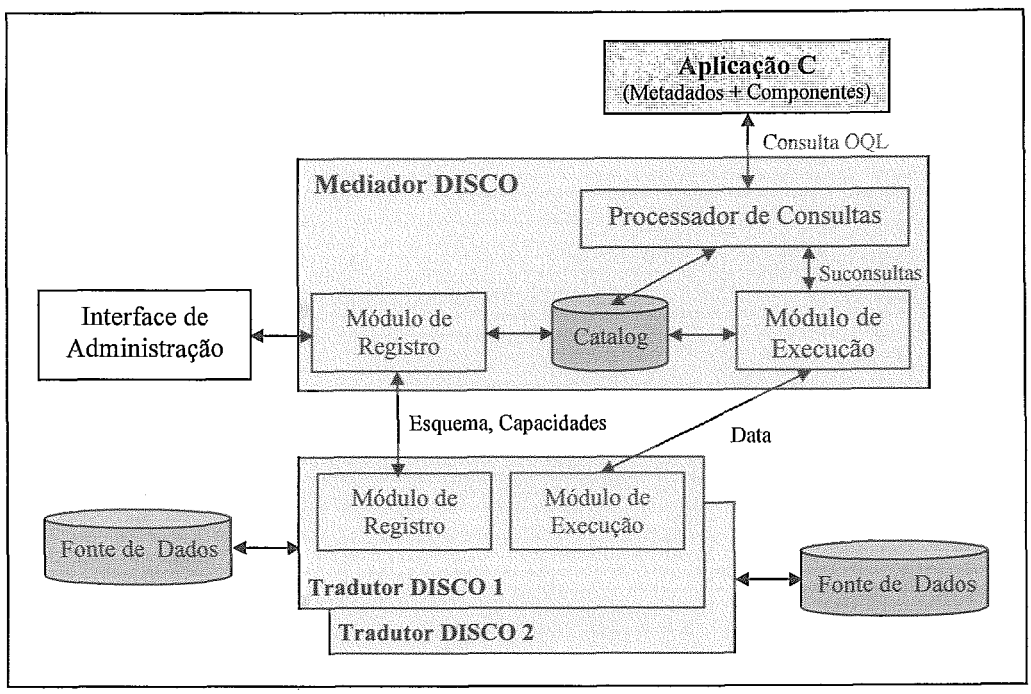


Figura 2.3 - Arquitetura DISCO.

Mediadores

Os mediadores são os componentes da arquitetura responsáveis por prover uma visão uniforme das múltiplas fontes de dados que eles integram. Independente do modelo de dados e da linguagem de consulta da fonte publicada, todas as informações são acessadas pelo usuário via mediador de forma homogênea, através de um modelo de dados e uma linguagem de consulta orientada a objetos.

Cada mediador possui internamente um processador de consultas globais, cuja função é gerar para cada consulta recebida um plano de consulta distribuído, que é esquematizado na forma de uma árvore constituída de operadores algébricos (*select*, *project*, *join*, *scan*, etc). Uma vez montada esta árvore, o processador identifica as melhores subárvores que deverão ser submetidas para processamento em cada tradutor. A fim de montar planos de consultas distribuídos mais eficientes, o mediador implementa um modelo de custo genérico aplicado sobre todos os tradutores. Informações como tempo de acesso a fonte, tamanho de coleções, dentre outras, são exportadas pelos tradutores de modo a especializar o modelo de custo a ser utilizado para uma determinada fonte. Outro aspecto importante do processador de consultas é o tratamento especial dado ao processamento de consultas para fontes não avaliadas. Quando uma consulta envolve várias fontes de dados, parte delas ou até mesmo todas podem não estar disponíveis no momento. Para esta situação, o processador de consultas retorna um resultado especial: a primeira parte do resultado contém as subconsultas que foram enviadas às fontes de dados não avaliadas, enquanto que a segunda parte contém os dados retornados pelas fontes de dados avaliadas. Isto traz as seguintes vantagens para o usuário: (1) saber que algumas fontes de dados não foram consultadas por problemas de comunicação; (2) executar novamente apenas as subconsultas que falharam assim que as fontes de dados não avaliadas se tornarem disponíveis novamente.

Cada mediador possui ainda internamente um dicionário de dados denominado *Catalog*. Este banco guarda informações indispensáveis ao processamento de consultas no mediador, como por exemplo, o esquema global do mediador, os mapeamentos entre o esquema global e os esquemas locais de cada fonte de dados, os custos de acesso às fontes, os tradutores existentes, dentre outras. Estas informações são fornecidas pelo Módulo de Registro, que as importa via tradutor durante o registro de uma nova fonte no sistema.

Tradutores

Os tradutores, por sua vez, são responsáveis por tratar os problemas de heterogeneidade de cada banco de dados integrado. Eles recebem as subconsultas provenientes do mediador e as traduzem para a linguagem apropriada do banco de

dados. O banco então processa a consulta e retorna os resultados para o tradutor, que os converte para o modelo de dados do mediador.

Os tradutores são responsáveis também por exportar para o mediador diversas informações (metadados) da fonte de dados integrada, tais como o esquema local, operadores algébricos suportados, informações de custos, etc.

Interface de Administração

É uma interface utilizada pelo administrador do sistema para acessar os serviços do Módulo de Registro do mediador, permitindo o registro de novos tradutores e a importação dos metadados relacionados as fontes publicadas. Todas estas informações são armazenados no *Catalog*.

Em seu protótipo inicial, o DISCO foi projetado apenas para integrar bancos de dados estruturados com capacidade própria para processamento de consultas. Mais recentemente, porém, o sistema se tornou um produto comercial desenvolvido pela empresa *Bull*, que passou a desenvolver novos pacotes de tradutores com capacidade para suportar dados semi-estruturados e não estruturados (adaptadores para *Notes*, *AltaVista*, *JDBC*, etc).

2.3.2 - HIMPAR

A HIMPAR – "*Heterogeneous Interoperable Mediators and Parallel Architecture*" (PIRES e MATTOSO, 1996) (PIRES, 1997), desenvolvido pela COPPE na Universidade Federal do Rio de Janeiro, é uma arquitetura baseada em padrões que visa a integração de diversos repositórios de dados distribuídos e heterogêneos. A fim de prover uma visão uniforme de diferentes repositórios de dados, a HIMPAR implementa uma camada de integração, estruturada hierarquicamente, baseada no conceito de Mediadores e Tradutores (WIEDERHOLD, 1992 e 1999). O modelo de dados interno e a linguagem de consulta global seguem os conceitos de orientação a objetos, baseado no padrão ODMG-93 (CATTEL, 1993). O principal diferencial desta arquitetura quando foi projetada foi o tratamento da interoperabilidade em relação à

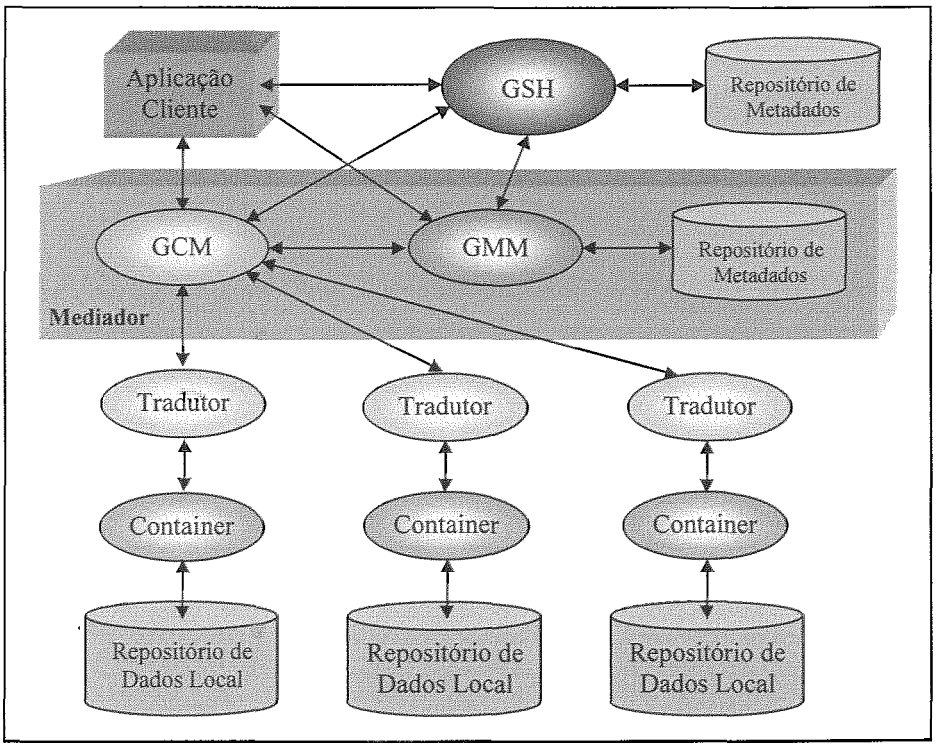


Figura 2.4 - Arquitetura HIMPAR.

comunicação entre os seus diferentes componentes, realizada através do padrão CORBA (CORBA, 1997).

A seguir, são descritos, em maiores detalhes, os principais componentes da arquitetura, como ilustrado na figura 2.4.

Mediadores

Formam a camada global responsável pelas funcionalidades de integração dos dados e de processamento de consultas globais. Cada mediador fornece uma visão integrada dos dados associados a um determinado domínio de aplicação. Um domínio pode ser representado por um ou vários mediadores organizados numa forma hierárquica, ou seja, um mediador pode estar ligado a outros submediadores que representam especializações dentro do domínio. No entanto, cada mediador pode ser construído e mantido de forma independente dos demais, o que permite uma arquitetura de alta extensibilidade.

Na HIMPAR, cada mediador é implementado por dois objetos CORBA: o Gerente de Metadados do Mediador (GMM) e o Gerente de Consultas do Mediador (GCM). O GMM é objeto responsável pela criação, manutenção e gerência das

informações contidas no esquema de integração do Mediador e que são armazenadas no Repositório de Metadados do Mediador. Já o GCM é objeto responsável pela análise e decomposição das consultas de acordo com as informações contidas no Repositório de Metadados do Mediador fornecidas pelo GMM. Após a decomposição e otimização da consulta global, as subconsultas são enviadas para os Tradutores envolvidos naquela consulta.

Tradutores

São responsáveis pelas funcionalidades de conversão de subconsultas da linguagem global (OQL) para a linguagem de consulta local de cada repositório. Após a conversão, o Tradutor envia a subconsulta para o *Container* que encapsula o sistema que gerencia o repositório de dados local.

Containers

São responsáveis por encapsular o sistema de gerenciamento de cada repositório de dados local, tornando-o compatível com o padrão CORBA. A implementação de um *Container* é diretamente dependente do sistema a ser encapsulado. Logo, cada tipo de sistema possuirá um *Container* específico. *Containers* para sistemas que não sejam orientados a objetos, como por exemplo SGBDs Relacionais ou repositórios não estruturados, devem implementar o mapeamento entre o modelo utilizado pelo repositório e modelo de dados global. Além disso, no caso dos repositórios de dados semi-estruturados e não estruturados que apresentam facilidades restritas de acessos aos dados armazenados, deverão ser implementadas funcionalidades de consultas e recuperação dos dados locais. No entanto, sistemas que ofereçam adaptadores CORBA poderão ser automaticamente acoplados a HIMPAR.

Gerente de Serviços HIMPAR (GSH)

Fornece um conjunto de serviços relacionados a gerência e manutenção dos metadados do sistema. Estes serviços compreendem o registro de novos objetos GCM, GMM, Tradutores e *Containers* (armazenados em um repositório interno de metadados), e a consulta aos objetos já cadastrados. O GSH fornece ao cliente HIMPAR recursos para descobrir quais são os Mediadores disponíveis no sistema.

Repositórios de Dados Local

Representam as fontes de dados distribuídas e heterogêneas integradas a arquitetura. A HIMPAR é capaz de integrar variadas fontes: SGBDs Orientados a Objeto, SGBDs Relacionais, repositórios de dados semi-estruturados e não estruturados.

Aplicação Cliente

Para que uma aplicação cliente acesse os serviços da HIMPAR, é necessário que esta encapsule um objeto CORBA que conheça as definições das interfaces do GSH, do GCM e do GMM. Através do GSH, o cliente consegue obter informações sobre os mediadores disponíveis no sistema. Uma vez obtida a referência para um determinado mediador, o cliente pode solicitar ao GMM os serviços de obtenção do esquema global do mediador e assim conseguir formular consultas a serem enviadas para GCM.

2.3.3 – Le Select

O *Le Select* (XHUMARI e MOKRANE, 1999), originalmente desenvolvido pelo grupo INRIA na França, é um *middleware* que implementa uma arquitetura para facilitar a publicação de dados de natureza heterogênea, provendo mecanismos para a consulta dos dados publicados, e a publicação de programas, provendo mecanismos para acessar seus serviços.

Além de publicador, outro papel importante do *Le Select*, na realidade seu principal objetivo, é a capacidade de integrar dados e programas distribuídos. Através do *Le Select*, um cliente situado em um site A é capaz de executar uma consulta em site B e repassar a resposta a um programa publicado em um site C, que recebe e processa os dados, gerando como saída um conjunto de resultados que é devolvido ao cliente.

Em contraste a muitos outros sistemas de integração existentes, o *Le Select* é um sistema completamente distribuído, não existindo um repositório global de dados publicados, nem ao menos um esquema global de informações integradas. Além disso, deixa de lado a característica de múltiplas camadas de integração, para se tornar essencialmente um sistema Cliente/Servidor.

A seguir, são descritas a arquitetura geral do *Le Select* e suas principais funcionalidades.

Arquitetura Geral

A figura 2.5 ilustra o desenvolvimento de aplicações utilizando o *Le Select*. Duas entidades importantes são bem distinguidas: os sites publicados e os clientes. Para publicar dados e programas, o publicador precisa primeiramente instalar e executar em sua máquina local um Servidor *Le Select*. Além disso, o publicador precisa informar ao *Le Select* como acessar os dados. Para isso, o publicador precisará escrever um programa tradutor ou tentar configurar, se possível, um tradutor já existente fornecido pela Fábrica de Tradutores *Le Select*. Uma vez convertidos, todos os dados publicados passam a ser enxergados pelos clientes na forma relacional (tabelas contendo linhas e colunas). Os clientes podem acessar esses dados através de uma aplicação que inclua a biblioteca cliente *Le Select* ou diretamente via visualizador *Web*.

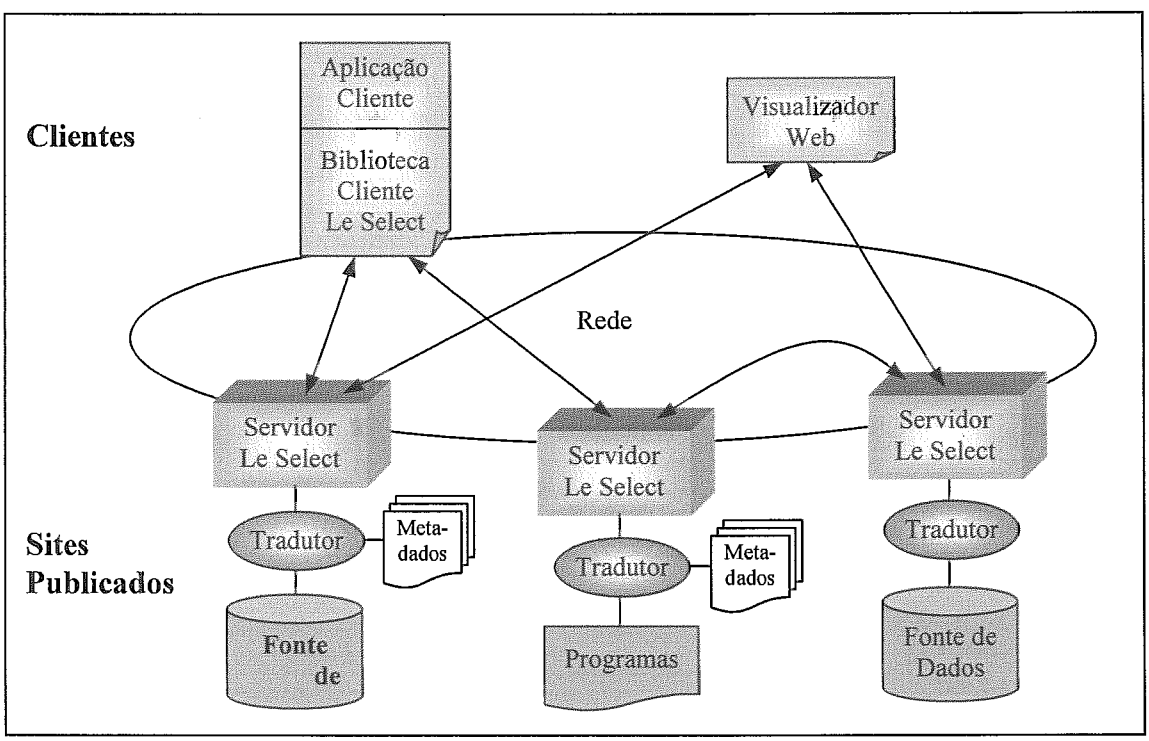


Figura 2.5 - Arquitetura *Le Select*.

Acesso a Dados e Programas

O Servidor *Le Select* implementa uma API que provê as funcionalidades para o processamento de consultas e para a execução dos programas publicados. O processador

de consultas constitui uma leve extensão do padrão JDBC, considerando apenas um subconjunto dos comandos da linguagem de consulta do padrão SQL92. Este subconjunto inclui basicamente as cláusulas SELECT-FROM-WHERE. A cláusula SELECT permite a seleção de todos os campos de uma tabela (*) ou apenas dos campos especificados diretamente pelo nome. A cláusula FROM deve ser seguida da localização completa da tabela a ser consultada, incluindo o endereço da máquina, nome do tradutor e nome da tabela. Apenas uma tabela pode ser consultada por vez, não sendo possível ainda realizar junções. Já a cláusula WHERE suporta operadores de comparação (=, >, <, <=, >=, <>), funções aritméticas (+, -, *, /) e operadores booleanos (AND, OR, NOT). Funções de agregações (COUNT, SUM, AVG, MAX, MIN) e as cláusulas ORDER BY, GROUP BY e HAVING não são ainda suportadas. A seguir, são ilustrados alguns exemplos de consultas que podem ser enviadas ao *Le Select*:

```
select *
from //www.ics.forth.gr/Bathymetry-Crete/Heraklion;

select x,y,z
from //www.ics.forth.gr/Bathymetry-Crete/Heraklion
where (X > 10) and (Y < 20);
```

Além disso, a API também provê uma outra linguagem específica para solicitação assíncrona de serviços. Esta linguagem é formada essencialmente por um conjunto de três comandos:

- JOB EXECUTE *<nome_programa>*
  parameter *<nome_parametro>* = '*valor_parametro*'
  input dataset is *<consulta_SQL>*
- JOB QUERY *<id_serviço>* *<timeout>*
- JOB DELETE < *id_serviço* >

O comando JOB EXECUTE permite disparar a execução de um programa, podendo passar como argumentos de entrada uma lista de parâmetros simples (formado por tipos inteiros, reais, booleanos, etc) e uma lista de tabelas resultados de consultas enviadas a Servidores *Le Select*. Como a execução de um programa pode demorar de alguns segundos a várias horas, a solicitação do comando JOB EXECUTE devolve ao

cliente apenas um identificador para o serviço em execução. Através do comando JOB QUERY <*id_serviço*>, o cliente pode descobrir se a execução foi encerrada ou não. Um segundo parâmetro <*timeout*> indica quanto tempo em milisegundos o cliente deve esperar para receber a resposta. Se esse parâmetro não é especificado, o comando retorna imediatamente. Uma vez encerrada a execução, o resultado de saída gerado pelo programa fica armazenado no próprio site do programa, em um diretório interno de trabalho do Servidor *Le Select*. Esse resultado fica disponível para o cliente até que um comando JOB DELETE <*id_serviço*> seja solicitado para sua remoção.

Para ilustrar a funcionalidade de execução de programas via *Le Select*, seja um programa hipotético de exemplo denominado `Converte`, que possui a seguinte interface de solicitação:

```
Converte(
  table Point(x: float, y: float, z: float),
  heigth: float,
  length: float):
  table points(id: int, x: float, y: float, z: float);
```

A seguir, é ilustrado o comando de chamada para este programa a partir do *Le Select*:

```
job execute //www.cos.ufrj.br/LeSelect/Converte
  input dataset is
    select x, y, z from //www.nce.ufrj.br/LeSelect/PointTable
  parameter height = 20
  parameter length = 50
```

Para acessar o servidor *LeSelect*, diferentes módulos de comunicação são avaliados, cada um deles baseado num protocolo de comunicação particular e exportando algumas funcionalidades da API. Os protocolos de rede, atualmente suportados pelo *Le Select*, são: CORBA, sockets TCP/IP, FTP e HTTP.

Publicação de Dados e Programas

Para publicar seus dados, o publicador deve informar ao Servidor *Le Select* previamente instalado em sua máquina como acessar os dados. Para isso, ele precisa

escrever um programa tradutor de dados. Este é responsável por converter os dados da sua forma original para o modelo de dados interno do *Le Select*, ou seja, o modelo relacional. Fábricas de Tradutores de Dados podem ser utilizadas pelos publicadores para criar seus próprios tradutores de dados através da configuração de parâmetros. Atualmente, o *Le Select* provê tradutores para banco de dados JDBC e para arquivos textos (ASCII) estruturados na forma tabular.

De forma semelhante a publicação de dados, a publicação de programas exige que o publicador escreva um tradutor de programas. Este é responsável por: (1) converter os dados de entrada da forma de tabela para o formato exigido pelo programa; (2) executar o programa; e (3) gerar os resultados de saída que serão acessados pelos tradutores de dados. Os dados de entrada são especificados através de consultas (embutidas dentro do próprio comando de solicitação de programas) enviadas a um Servidor *Le Select*. De forma similar às Fábricas de Tradutores de Dados, Fábricas de Tradutores de Programas são utilizadas pelos publicadores para criar tradutores de programas através da configuração de parâmetros.

Para utilizar um tradutor já existente, basta ao publicador escrever um arquivo de definição para o tradutor (*wrapper definition file*), onde são configurados seus parâmetros. Este arquivo utiliza uma sintaxe particular no formato XML, devendo ser salvo com a extensão ".wd" e armazenado no diretório particular de configurações do sistema de arquivos do Servidor *Le Select*. No entanto, se o publicador necessitar escrever um novo tradutor, ele deverá antes implementá-lo herdando um conjunto de classes *Java* abstratas (que buscam facilitar ao máximo o trabalho de implementação) fornecidas pela API *Le Select*. A seguir, é ilustrado um exemplo de arquivo de definição para um tradutor de dados do tipo texto:

```
<Source wrapperType="TextWrapper" name="Paris Stock Exchange">
  <Parameters>
    <Table name="year95" file="/data/stocks/Paris/1995.data">
      <Column name="Date" type="char" size="8"/>
      <Column name="Open" type="double" size="9"/>
      <Column name="Higher" type="double" size="9"/>
    </Table>
  </Parameters>
</Source>
```

Publicação de Visões

A fim de dar suporte à integração e transformação de dados, o *Le Select* oferece o mecanismo de visões. Uma visão é constituída pela declaração de uma lista de consultas SQL que permite a integração de dados publicados em diferentes sites. Esta lista pode ser constituída por consultas simples ou por consultas que contenham o cálculo de expressões na cláusula SELECT, permitindo a aplicação de transformações sobre os dados publicados antes do processo de integração.

Uma vez que a visão tenha sido definida, esta passa a ser utilizada por clientes *Le Select* como sendo uma tabela comum. No entanto, o sistema internamente não executa a visão imediatamente após sua definição para materialização do resultado. Ao invés disso, quando um cliente envia uma consulta ao Servidor *Le Select* referenciando esta tabela, a definição da visão é recuperada e incorporada à consulta cliente para otimização, sendo então executada e materializada. Para o cliente *Le Select*, todos estes passos são totalmente transparentes.

Semelhante à publicação de tradutores, para se publicar uma visão, o publicador deve escrever a definição da visão em um arquivo de sintaxe particular no formato XML, salvá-lo com extensão ".wd" e armazená-lo no diretório particular de configurações do sistema de arquivos do Servidor *Le Select*. A seguir, é ilustrado um exemplo de arquivo para a definição de uma visão:

```
<View>
  <Definition query="select 10*x, 10*y, 10*z from
    //www.nce.ufrj.br/LeSelect/PointTable"/>
  <Documents>
    <ViewDocument>
      <attribute name="x" value="escala x"/>
      ...
    </ViewDocument>
  </Documents>
</View>
```

Publicação de Metadados

Uma vez que o *Le Select* organiza os dados dentro de tabelas, o usuário passa a ter um bom conhecimento da sintaxe dos dados através dos nomes e tipos de cada coluna, porém ele não tem nenhuma informação sobre a semântica dos dados, ou seja, o que cada valor representa, para que serve, etc. Além disso, informações auxiliares podem contribuir bastante para definir uma semântica geral a cada conjunto de dados publicados, tal como autor dos dados, datas de criação e publicação, referências para outros documentos, etc.

Por isso, para enriquecer o conteúdo dos dados publicados, o *Le Select* oferece um conjunto de serviços para publicação de metadados, permitindo que sejam anexados documentos no formato XML tanto para os dados publicados quanto para os tradutores produzidos. As especificações destes documentos podem ser definidas dentro do arquivo de definição do tradutor (*wrapper definition file*) de duas formas diferentes: (1) o conteúdo do documento pode ser embutido diretamente no arquivo de definição do tradutor; ou (2) o conteúdo do documento pode ser especificado em outro arquivo cujo nome é definido no arquivo de definição do tradutor. Veja um exemplo a seguir:

```
<Documents>
  <WrapperDocument>
    <attribute name="TextWrapper" value="Convert tabular text
      tables..."/>
    ...
  </WrapperDocument>
  <TableDocument tableName="Year95">
    <attribute name="Date" value="Fabrication date"/>
    ...
  </TableDocument>
  <TableDocument tableName="Year96" XMLFileName="Tmp/Year96.xml" />
<Documents>
```

Estes documentos podem ser consultados quando o site publicado é visualizado, podendo também servir como base para a busca de dados publicados.

2.3.4 - Agora

O *Agora* (MANOLESCU, FLORESCU e KOSSMANN, 2000, 2001a e 2001b), desenvolvido pelo grupo INRIA na França, é um *middleware* voltado para integração de dados distribuídos e heterogêneos, caracterizado por associar XML como formato padrão de interface com o usuário e processamento de consultas relacional interno à camada de integração. Tanto a linguagem de consulta quanto os documentos de resposta gerados como saída estão baseados no padrão XML, fazendo todo o processamento relacional interno transparente para o usuário.

Este trabalho investiga o uso de diferentes técnicas para integrar as tecnologias relacional e XML, de modo a obter uma alternativa de consulta mais eficiente que um puro sistema de mediação baseado em XML. Neste sentido, são propostas soluções para: (1) mapear documentos XML em um esquema genérico de dados relacional; (2) traduzir consultas escritas em uma linguagem de consulta XML para SQL e rescrever os resultados de consultas relacionais na forma de documentos XML, baseado neste esquema genérico relacional; (3) aumentar o desempenho de recuperação de estruturas de documentos XML armazenadas neste modelo relacional genérico a partir de palavras chave, utilizando para isto um conjunto de poucas tabelas que armazenam índices para textos.

O *Agora* (figura 2.6) é implementado como uma camada de integração acima do sistema *Le Select* (descrito na seção 2.3.3). Desta forma, todas as funcionalidades para a publicação de dados e mais o processamento de consultas relacionais são providos pelo *Le Select*. O que o *Agora* faz é estender as funcionalidades do *Le Select*, desenvolvendo uma camada superior que oferece uma visão XML de todas as fontes de dados publicadas. No entanto, estas visões XML são armazenadas internamente na forma de tuplas relacionais, sendo que, para isto, o trabalho propõe um conjunto de tabelas padrões para representar de uma forma genérica qualquer instância de um documento XML. Além da visão XML, o sistema oferece também uma linguagem de consulta baseada no padrão XML, o *XQuery*. Internamente, uma consulta em *XQuery* submetida ao sistema é convertida para linguagem de consulta relacional SQL e repassada ao processador de consultas do *Le Select*. O *Le Select* retorna, então, um conjunto de tuplas relacionais de resposta, que são convertidas para documentos XML e repassados ao

usuário do sistema. Por fim, o sistema oferece ainda suporte a busca com palavras chave, que permite ao usuário recuperar a estrutura exata de um documento XML armazenado no repositório de visões, de tal forma a ajudá-lo a formular consultas mais eficientes.

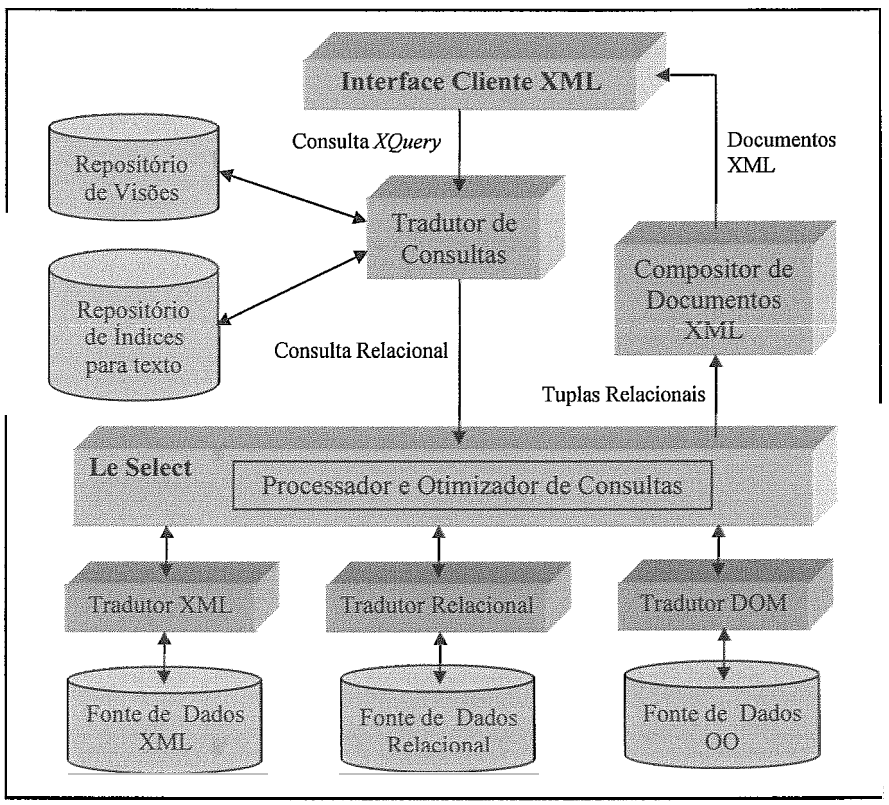


Figura 2.6 - Arquitetura *Agora*.

2.3.5 - MIX

O MIX – "*Mediation of Information using XML*" (MIX), original da Universidade da Califórnia em *San Diego*, é um projeto que tem por objetivo estudar e desenvolver sistemas de integração de fontes de informações distribuídas e heterogêneas baseado nas tecnologias de mediação e XML. A filosofia do MIX é enxergar a Web como um enorme banco de dados distribuído cujo modelo de dados segue o emergente padrão XML. Devido a grande variedade de informações, sistemas de mediadores são construídos sobre este enorme banco de dados, a fim de prover ao usuário uma visão integrada dos dados organizados por diferentes assuntos de seu interesse.

Um dos principais sistemas desenvolvido como parte deste projeto é o MIXm – "*MIX Mediator System*" (BARU et al., 1999). Neste sistema, a troca e a integração de dados é realizada somente através de XML, ou seja, informações de instâncias e esquema são representadas através de documentos XML e DTDs XML, respectivamente. Para consultar o sistema, o usuário utiliza o XMAS, uma linguagem de consulta que é também baseada no padrão XML. Como opção, o sistema oferece ainda uma interface gráfica denominada BBQ – "*Blended Browsing and Querying*", que permite a construção de consultas complexas de forma bastante intuitiva (semelhante ao QBE) sobre o DTD de visão provido pelo mediador.

A figura 2.7 ilustra a arquitetura do MIXm. Para o sistema, todas as fontes de informação devem prover: (1) uma visão XML de seus dados; e (2) um conjunto de consultas XMAS que elas são capazes de responder. Caso a fonte de dados não tenha esta capacidade própria, um tradutor (*wrapper*) deverá ser construído para prover estes serviços.

Conceitualmente, o MIXm exporta um DTD correspondente à visão de cada fonte de dados. O repositório de definição de visões armazena os metadados relacionados aos mapeamentos entre o DTD de visão do mediador e os DTDs de visões de cada fonte integrada. No entanto, os dados são previamente materializados.

Em tempo de execução, aplicações (tal como o BBQ GUI) remetem ao mediador consultas formuladas em XMAS baseadas na sua visão integrada. Internamente, o mediador decompõe a consulta cliente dentro de subconsultas a serem enviadas para cada fonte remota. As fontes geram e retornam ao mediador os resultados das consultas na forma de documentos XML. O mediador integra então os resultados dentro de um único documento e o devolve à aplicação cliente. Esta acessa o resultado da consulta através de uma extensão própria da API DOM, denominada DOM-VXD – "*DOM for Virtual XML Documents*". Esta API permite que os resultados de uma consulta não precisem ser materializados por completo da fonte remota, na medida que ela oferece mecanismos de pedidos sobre demanda, conforme o usuário vai navegando por dentro do documento XML de resposta.

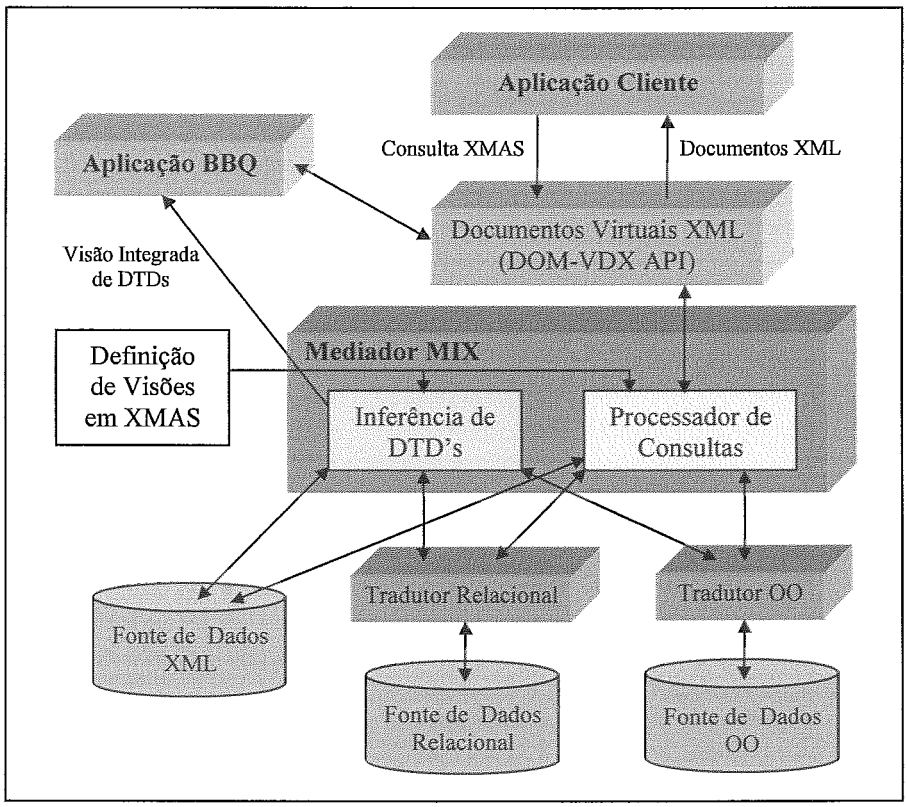


Figura 2.7 - Arquitetura *MIXm*.

2.4 - Considerações Finais

Neste capítulo foi abordado um dos principais assuntos de pesquisa na área de banco de dados atualmente: o problema da integração de fontes de dados heterogêneas e distribuídas. Este é um problema que vem sendo pesquisado há cerca de duas décadas. Inicialmente, as pesquisas estavam voltadas para solucionar o problema da integração de SGBDs heterogêneos e distribuídos, que manipulavam dados estruturados e que possuíam capacidade própria para processamento de consultas. No entanto, mais recentemente, a Web trouxe à tona uma nova realidade para a área de banco de dados: fontes de dados semi-estruturados e não estruturados, publicando tanto dados quanto programas. Isto tem direcionado as pesquisas para atender a necessidade de se integrar não somente SGBDs tradicionais, mas também, de uma forma mais abrangente, fontes de dados quaisquer em sites espalhados pela *Web*.

Exposto o problema, foi apresentado um resumo da classificação proposta por DITTRICH e DOMENIG (1999), evidenciando diferentes abordagens encontradas na

literatura para tratar do problema em questão. O quadro a seguir resume esta classificação, listando cada uma das cinco abordagens propostas com suas respectivas vantagens e desvantagens.

	Vantagens	Desvantagens
Banco de Dados Universais	• Completa funcionalidade de SGBD • Processamento de consultas com alto desempenho	• Custosa migração dos dados • Migração de aplicações • Não suporta consultas imprecisas
Data Warehouses	• Completa funcionalidade de SGBD • Processamento de consultas com alto desempenho • Projeto do banco otimizado para aplicações específicas • Armazenamento do histórico de atualizações	• Custosa migração dos dados • Nem todos os dados são avaliados • Os dados mantidos podem estar desatualizados • Não suporta consultas imprecisas
Banco de Dados Federados	• Completa funcionalidade de SGBD • Integração virtual dos dados, o que permite dados sempre atualizados	• Processamento de consultas com baixo desempenho • Não suporta consultas imprecisas
Máquinas de Busca	• Voltadas para consultas a dados não estruturados • Útil para consultas imprecisas	• Não útil para consultas precisas • Somente consulta
Sistemas de Consultas Mediados	• Suporte a todos os tipos de dados • Suporte a consultas precisas e imprecisas • Suporte dinâmico a conjunto de fontes de dados	• Somente consulta

Figura 2.8 – Resumo da classificação de DITTRICH e DOMENIG (1999).

Por fim, foi apresentada uma série de trabalhos relacionados que exemplificam as soluções adotadas por diferentes projetos de pesquisas voltados para a integração de informações na Internet. Ao analisar estes trabalhos, foi possível verificar realmente que a *Web* vem se tornando o centro das atenções de pesquisas, e que por isso, a preocupação para o suporte a dados semi-estruturados como XML (em larga expansão na *Web*) vem crescendo bastante nos projetos mais recentes.

No próximo capítulo, são apresentados diferentes aspectos relacionados à construção de repositórios voltados para o armazenamento e a recuperação de componentes de software, levando-se em conta, sobretudo, questões como a distribuição

e heterogeneidade de repositórios espalhados pela Internet. Além disso, são apresentados também outros trabalhos voltados mais especificamente para busca e recuperação de componentes de software, e que são baseados em diferentes técnicas de integração apresentadas neste capítulo. Ao final, é feito uma análise crítica geral dos trabalhos descritos, sendo apresentado, então, o diferencial do trabalho proposto por esta dissertação.

3 Repositórios de Componentes de Software

3.1 – Introdução

O mercado cada vez mais competitivo em que as empresas desenvolvedoras de software se encontram atualmente exige que estas reformulem e aperfeiçoem dia após dia sua metodologia de produção. Cada vez mais, aumenta a demanda por sistemas maiores e mais complexos. Cada vez mais, aumentam as exigências dos usuários por produtos de software mais modernos, rápidos e confiáveis. Todas estas necessidades exigem mudanças na abordagem atual de desenvolvimento de software (COHEN, 1998). Segundo Cohen, o desenvolvimento de software requer, dentre diversos aspectos, (1) um maior grau de automação de suas atividades e controle do processo de desenvolvimento; (2) a utilização de novas tecnologias, como distribuição e componentização; (3) a utilização de padrões adotados pelo mercado; dentre outros aspectos. Adaptar-se a todos estes aspectos requer tempo e esforço, porém, são essenciais para que objetivos como (1) aumento da produtividade e da qualidade do processo de desenvolvimento; (2) diminuição dos custos; (3) redução dos prazos de entrega; (4) redução de defeitos e riscos dos produtos finais; (5) facilidade de adaptação à novas tecnologias; possam ser alcançados pelas empresas.

Neste contexto, o Desenvolvimento Baseado em Componentes (DBC) vem ganhando força nos últimos anos como a mais promissora tendência para o processo de desenvolvimento de software capaz de atender aos requisitos das aplicações modernas (SAMETINGER, 1997). O DBC tem como principal proposta a construção de aplicações pela composição de produtos de software preexistentes, através da identificação e reutilização de diferentes componentes construídos e utilizados em sistemas anteriores. Este reaproveitamento pode proporcionar, entre outras coisas, a redução da complexidade, dos custos e do tempo de desenvolvimento, além de aumentar

a confiabilidade dos sistemas produzidos, pois a cada uso de um determinado requisito, sua eficiência e validade são novamente verificadas, garantindo, desta forma, maior qualidade e redução de erros nos produtos finais.

Assim sendo, a **reutilização** vem ganhando cada vez mais espaço dentro das empresas como uma atividade essencial para o sucesso do processo de desenvolvimento de software. No entanto, para que a reutilização seja efetiva, é necessário que exista dentro da organização um poderoso mecanismo de compartilhamento de informações. Infelizmente, a maioria das empresas não consegue lidar com a grande quantidade de dados heterogêneos e distribuídos produzidos pelos seus diversos setores. Isto dificulta bastante a troca de componentes entre os desenvolvedores, que, na maioria das vezes, nem sequer sabem que eles existem. Dentro deste contexto, a construção de um grande repositório, capaz de integrar informações de outros repositórios heterogêneos e distribuídos, surge como uma proposta de solução para organizar as informações de variados componentes, tornando-os públicos e acessíveis a todos dentro da empresa.

Além disso, a Internet, que é atualmente uma enorme fonte de distribuição de informações, incluindo componentes de software, pode ser também um fator de grande contribuição para o processo de reutilização, pois esta é capaz de prover uma grande variedade de componentes produzidos por vários desenvolvedores em todo o mundo. Portanto, um repositório integrado, capaz de prover mecanismos de suporte à busca e recuperação de componentes em bases de repositórios distribuídas, considerando tanto a Intranet de uma empresa quanto a Internet como um todo, representa uma ferramenta em potencial para a prática da reutilização. Isto pode ajudar bastante os desenvolvedores de software a encontrar e selecionar mais rapidamente para seus projetos componentes já prontos e testados por outros desenvolvedores, aumentando, consequentemente, a produtividade e a qualidade final do processo de desenvolvimento.

No restante deste capítulo, são abordadas diferentes questões relacionadas à construção de repositórios de componentes de software para apoio a reutilização. Primeiramente, a seção 3.2 apresenta os benefícios do uso de SGBDs para o armazenamento e a recuperação de componentes de software. A seguir, a seção 3.3 aborda os problemas da distribuição e heterogeneidade de repositórios de software na Internet. A seção 3.4, por sua vez, apresenta alguns trabalhos relacionados ao assunto

em questão. Por fim, a seção 3.5 conclui o capítulo e apresenta as perspectivas futuras de trabalhos nesta área.

3.2 – O Uso de SGBDs no Suporte ao Armazenamento e à Recuperação de Componentes de Software

Segundo Sametinger (SAMETINGER, 1997), um repositório de componentes é uma base preparada para o armazenamento, seleção e obtenção de componentes. Embora o papel desempenhado pelo repositório seja uma característica tipicamente operacional, o sucesso de um processo de desenvolvimento de software baseado na reutilização, como o DBC, está diretamente ligado aos mecanismos de armazenamento e recuperação dos componentes reutilizáveis. Portanto, a utilização de um repositório capaz de prover estes mecanismos com qualidade se torna uma condição importante para se atingir tal sucesso.

Para prover um serviço de armazenamento de componentes de software com qualidade, um repositório deve ser capaz de manipular dados de alta heterogeneidade. Além disso, componentes de software são, em geral, estruturas de dados que apresentam uma grande complexidade de representação, caracterizado sobretudo pela grande quantidade de relacionamentos internos. Aliado à necessidade de um modelo capaz de representar e armazenar componentes, acrescenta-se ainda a manipulação de grandes coleções de componentes e a navegação entre instâncias de componentes. Tais requisitos praticamente inviabilizam o uso de sistemas de arquivo para a persistência, devido à falta de flexibilidade do armazenamento e manipulação, além de impor uma gerência desses componentes em memória para manipular grandes coleções. Desta forma, analisando o estado da arte em gerência de dados, a utilização da tecnologia dos SGBDs associada à tecnologia de orientação a objetos (como os SGBDs Orientados a Objetos e os SGBDs Relacionais-Objeto) pode ser apontada como uma proposta apropriada para a manipulação de componentes, pois esta alinha um modelo de representação poderoso, representado pelo paradigma orientado a objetos, com a disponibilização de mecanismos de acesso e gerência de dados adequados às estruturas de dados inerentes dos componentes de software.

A qualidade dos serviços de recuperação providos por um repositório, por sua vez, também está diretamente ligada a capacidade que o repositório tem de estruturar suas informações internamente. Mais uma vez, o estado da arte em gerência de dados aponta os SGBDs Orientados a Objetos e os SGBDs Relacionais-Objeto como os mais eficientes em técnicas de processamento e otimização de consultas. No entanto, para que esta recuperação seja efetiva, SAMETINGER (1997) ressalta a importância do armazenamento de informações adicionais (metadados) associadas ao componente. Isto permite que as consultas possam ser executadas sobre um conjunto de dados semanticamente mais ricos, aumentando assim a precisão dos componentes retornados.

Além das funcionalidades básicas de armazenamento, seleção e recuperação de componentes, outras funcionalidades, não necessariamente obrigatórias, porém presentes na maioria dos SGBDs, podem ajudar a melhorar bastante a qualidade do processo de disponibilização de componentes pelo repositório, incentivando ainda mais a prática da reutilização (EZRAN et al., 1999). Dentre estas funcionalidades, destacam-se:

- Pesquisa – além de prover serviços para simples exploração dos componentes armazenados, é importante que o repositório também forneça recursos para pesquisas mais refinadas sobre a base de metadados dos componentes. Isto permitirá que o usuário aumente a eficácia da busca pelos componentes desejados, facilitando assim a descoberta e a reutilização dos mesmos.

- Histórico – é importante para o gerenciamento do repositório que ele armazene informações do histórico de cada componente, tais como inclusão, exclusão e alterações realizadas ao longo do tempo. Estas informações reunidas permitem que os desenvolvedores possam consultar uma ficha de evolução de cada componente disponível, colaborando na análise e reutilização dos mesmos;

- Controle de versões – é importante que o repositório seja capaz de armazenar e gerenciar diferentes versões de um componente, estabelecendo um relacionamento entre elas e criando um histórico de alterações.

- Estatísticas – a coleta de estatísticas como quantidade de componentes disponíveis, freqüência de acessos a cada componentes, percentagem de

pesquisas bem sucedidas, tempo médio de processamento de consultas, taxa de recuperação, dentre outras, pode ajudar no gerenciamento do repositório.

- Notificação de mudanças – é importante que o repositório notifique a seus usuários alterações realizadas em componentes ou em suas principais funcionalidades, tais como: atualização de versão de um componente, inclusão ou exclusão de componentes, alterações na política de controle de acesso, etc.

- Acesso pela rede – com o crescimento cada vez maior das redes de computadores e do números de usuários distribuídos geograficamente dentro de uma organização, é importante que um usuário possa acessar o repositório através de um ponto qualquer da rede.

- Controle de acesso – um repositório pode adotar uma política de segurança que restrinja o acesso a determinadas funcionalidades apenas a grupo de usuários autorizados. Por exemplo, apenas usuários administradores devem possuir permissão para inserir ou excluir componentes do repositório.

3.3 – Distribuição e Heterogeneidade

O grande crescimento das redes de computadores nos últimos anos traz à tona dois novos problemas que devem ser levados em conta na montagem de qualquer repositório de dados: a distribuição e a heterogeneidade. Organizações pequenas, com poucas divisões departamentais, normalmente possuem apenas um único repositório centralizando todas as informações, e estes problemas deixam de ser evidentes. No entanto, organizações maiores ou geograficamente distribuídas são formadas, por sua vez, por diversos setores isolados, cada qual relacionado a uma área de atuação da empresa e contendo seu próprio repositório. Neste caso, faz-se necessário a montagem de um novo repositório capaz de integrar as informações de cada um dos repositórios distribuídos e homogeneizar os problemas de heterogeneidade existentes, como, por exemplo, diferentes dispositivos de armazenamento (sistemas de arquivos, banco de dados, etc), diferentes capacidades de consulta, dados de diferentes tipos e formatos (texto, tabelas, objetos, etc), diferentes protocolos de acesso via rede, etc. Este é, portanto, um problema típico de montagem de sistemas de bancos de dados distribuídos

e heterogêneos, para o qual diferentes soluções podem ser aplicadas, como já abordado no capítulo 2. A seguir, são apresentadas propostas que aplicam algumas destas soluções no contexto da montagem de repositórios de componentes de software distribuídos.

Levando-se em conta a característica de distribuição, SAMETINGER (1997) classifica os repositórios de componentes de software em três categorias: (1) Repositórios locais e centralizados – armazenam componentes de todos os tipos e aplicações; (2) Repositórios específicos a um domínio – armazenam componentes relativos a um determinado domínio de aplicação; e (3) Repositórios de referências – armazenam referências a componentes armazenados em outros repositórios remotos, funcionando como uma espécie de catálogo.

Segundo SEACORD (1999), repositórios locais e centralizados, armazenando vários componentes de forma genérica, tendem com o passar do tempo a falhar, pois a cada inserção de um novo componente estes se tornam cada vez mais inchados e consequentemente mais pesados durante um processo de pesquisa. Assim, uma solução proposta para minimizar este problema é a divisão e organização destes repositórios locais genéricos por domínio de aplicação. Desta forma, cada repositório restringe-se a armazenar uma faixa de componentes bem específica, o que faz diminuir bastante o escopo de componentes a serem consultados e melhorar, portanto, a eficiência dos resultados de pesquisas.

No entanto, o advento da Internet nos últimos anos permite que qualquer usuário de computador ligado em rede tenha acesso, através da Web, a um enorme mundo de informações distribuídas e heterogêneas, incluindo componentes de software. Desta forma, a Web pode ser considerada também um enorme repositório de componentes de software distribuídos, devendo, por isso, fazer parte dos mecanismos de busca por componentes no apoio a reutilização. Neste caso, em se tratando de repositórios externos à organização, SAMETINGER (1997) sugere a utilização do mecanismo conhecido como repositório de referência. No entanto, embora esta abordagem possa fazer aumentar em grande quantidade o número de componentes disponíveis, a falta de informações sobre os componentes publicados restringem bastante a qualidade dos mecanismos de pesquisa disponíveis atualmente, baseados, em sua maioria, no uso de

poucas palavras chave. Isto gera, como conseqüência, resultados muitos volumosos e de baixa precisão, dificultando a descoberta de componentes.

No contexto da reutilização, a necessidade por mecanismos de pesquisas mais eficientes para a busca e recuperação de componentes de software pode ser considerado um fator decisivo para o sucesso do processo de reutilização como um todo. As chances de um desenvolvedor buscar e reutilizar um componente já pronto, ao invés de construir um novo, se tornam cada vez maiores à medida que aumentam as facilidades e a qualidade dos mecanismos de pesquisa disponíveis. Um mecanismo de busca ruim pode desestimular o desenvolvedor a buscar e reaproveitar componentes, gerando, portanto, uma grande barreira à reutilização.

Na seção a seguir, é identificada a importância do uso de metadados para descrever os componentes publicados por um repositório de software e melhorar os mecanismos de busca dentro de um processo de pesquisa por componentes.

3.4 – A Utilização de Metadados na Organização de Repositórios de Componentes de Software

Metadados são informações que descrevem a estrutura e o significado de dados e aplicações. No contexto da Internet, metadados vêm sendo cada vez mais utilizados para descrever fontes de dados heterogêneas e distribuídas, provendo informações sobre a identificação, localização e conteúdo dos dados armazenados. Recentemente, metadados têm se tornado um importante trunfo para a construção de algoritmos de busca e recuperação de informações mais eficientes, pois, uma vez que as pesquisas são realizadas sobre dados semanticamente mais ricos, os resultados gerados passam a ser mais precisos e de melhor qualidade (MOURA, CAMPOS e BARRETO, 1999).

No contexto da reutilização, metadados não só ajudam a melhorar a qualidade do processo de pesquisa por componentes, como também ajudam a melhorar o processo de entendimento dos componentes, descrevendo suas características básicas que permitem ao desenvolvedor selecionar mais rapidamente, para uma análise posterior mais detalhada, os componentes que poderão atender as suas necessidades.

Conforme dito anteriormente, XML vem se popularizando como um padrão de formato de escrita e armazenamento de metadados, com o apoio de vários grupos internacionais de desenvolvedores: *World Wide Web Consortium* (W3C), *Object Management Group* (OMG), *Meta Data Coalition* (MDC), etc. No contexto da Engenharia de Software, pesquisadores como GUERRIERI (1998) descrevem as vantagens do uso de XML para a documentação de várias etapas do processo de desenvolvimento de software. No contexto da reutilização, diferentes propostas baseadas em XML para descrever componentes de software são encontradas na literatura, como, por exemplo, o *Open Information Model* (OIM), proposto pela *Meta Data Coalition* (MDC); o 3CML, proposto em (ALONSO e FRAKES, 2000); o *ComponetSource Reusable Component Specification* (COMPONENTSOURCE), proposto pela *Component Source*; dentre outras.

3.5 – Trabalhos Relacionados

Com a crescente motivação para a utilização da técnica de reutilização no processo de desenvolvimento de software, diversas empresas estão investindo na construção de repositórios que suportem o armazenamento e a recuperação de componentes de software. Algumas delas desenvolvem seus repositórios para uso interno, enquanto outras para propósitos comercias de publicação e venda de componentes na Internet. A seguir, são apresentadas algumas propostas encontradas na literatura e na Internet.

3.5.1 – ProReuso

O *ProReuso* (MELO, SOUZA e HOLANDA, 2001) é um repositório que visa o armazenamento, a análise, a classificação e a recuperação de componentes de software reutilizáveis e seus documentos associados. Um componente, denominado pelos autores de ativo de software, pode ser representado por um *JavaBeans*, *Enterprise JavaBeans*, documentos que representam casos de uso, diagramas de interação, diagrama de classes,

JavaDoc, manuais do usuário, etc. Esta ferramenta, que consiste de uma aplicação Web, é disponibilizada aos funcionários da *Oracle* do Brasil, através da Intranet da empresa.

Para alimentar o repositório com novos ativos, o *ProReuso* oferece funcionalidades de submissão de componentes e documentos relacionados. Estes documentos são estruturados no formato XML e indexados por uma ferramenta disponibilizada pelo banco de dados *Oracle*, a *Oracle Intermedia*. As informações armazenadas nestes documentos são utilizadas como base para as rotinas de pesquisa e recuperação de ativos.

Para gerenciar a qualidade dos ativos e documentos submetidos, o *ProReuso* implementa um processo de aprovação de componentes, baseado em um *workflow*. Este processo é inspecionado por um conjunto de gerentes de projeto e especialistas de qualidade, que devem validar e aprovar o ativo, para só então ser disponibilizado.

A seguir, são listadas as principais funcionalidades oferecidas pelo *ProReuso*:

- Submissão de Componentes e Documentos – é responsável por iniciar o processo de submissão de ativos. Durante a submissão, o usuário preenche um documento com as principais características do componente, as quais irão compor o documento XML;
- Acompanhamento de Processos – permite a um usuário acompanhar graficamente o processo de aprovação de ativos submetidos;
- Pesquisa – permite a procura por ativos e seus respectivos documentos armazenados no repositório. Essa pesquisa pode ser contextualizada, ou seja, é permitido ao usuário indicar quais campos do documento XML devem ser utilizados na pesquisa;
- Exploração de Catálogo – exibe ao usuário uma listagem de todos os ativos e suas descrições armazenados no repositório;
- Inserção de Comentários – permite a um desenvolvedor que tenha utilizado um ativo inserir comentários sobre ele, de modo a compartilhar suas experiências com outros desenvolvedores.

3.5.2 – ComponentSource

O *ComponentSource* (COMPONENTSOURCE) é um mercado de compra e venda de componentes de software na Web. Através de uma política própria, o site www.componentsource.com está aberto a qualquer desenvolvedor que queira publicar para venda seus componentes. Para isso, o site disponibiliza um bom suporte técnico que provê avaliação gratuita de versões, suporte a vários idiomas, centro de soluções para a construção de componentes e para comércio eletrônico, dentre outros serviços.

A fim de pesquisar componentes para a compra, o site fornece ao usuário um catálogo dividido inicialmente por tecnologias (COM, C++, *Java*, VCL, etc). Para cada um destes grupos, componentes podem ser localizados de acordo com a sua categoria (gráficos, editores, Internet, multimídia, etc), autor ou ordem alfabética do nome. O site permite ainda a execução de pesquisas mediante o uso de palavras chave.

Todo componente, quando "embalado" para venda, é descrito por um conjunto de quatro documentos que compõe a *ComponetSource Reusable Component Specification*. Estes documentos descrevem as funcionalidades (interface, operações, eventos, etc), a tecnologia (*Java*, COM, C++, etc), o pacote de distribuição (*zip*, *jar*, *dll*, etc) e as informações comercias (preço, licença, etc) relacionadas ao componente.

3.5.3 – Agora System

O *Agora System*[4] (SEACORD, HISSAN e WALLNAU, 1998) é uma máquina de busca que visa a recuperação de componentes de software reutilizáveis do tipo código, tal como componentes CORBA e *JavaBeans*. Este sistema mantém uma grande base de índices para repositórios de componentes de software na Web, classificados segundo o tipo de componente que armazenam. O *Agora System* combina um mecanismo de introspecção[5] à abordagem de máquinas de busca para registrar componentes do tipo código, através de informações sobre sua interface. Isto permite

---

[4] Este é um outro sistema diferente daquele proposto na seção 2.3.4, apesar da semelhança de nomes.
[5] Termo comum a *JavaBeans*. Descreve a habilidade do *JavaBeans* em prover informações de sua interface.

reduzir o custo de buscar e trazer o componente de software durante o processamento de consultas.

O *Agora System* suporta basicamente dois tipos de operações:

- Localização e indexação de componentes – tarefa executada automaticamente em *background* por agentes desenvolvidos em *JavaBeans* e CORBA. Uma vez que um componente tenha sido localizado, informações de sua interface (atributos, métodos, eventos, etc) são decompostas em um conjunto de termos que são utilizados para construir um documento indexado dentro da base de índices do sistema. Este documento inclui também a localização (URL) do componente na Web.

- Busca e recuperação de componentes – tarefa executada manualmente por um usuário. A consulta deve ser baseada no uso de palavras chave, a partir de termos encontrados nas interfaces dos componentes, tal como nomes e valores de atributos, métodos e eventos. Operadores lógicos (*and*, *or*, *not* e *near*) e operadores de concatenação (+ e -) são também avaliados.

3.5.4 – Odyssey Search

A *Odyssey Search* (BRAGA, 2000b) é uma ferramenta acoplada ao ambiente de reutilização *Odyssey* (WERNER et al., 1999, 2000 e 2002) e que visa oferecer serviços para a busca e recuperação de componentes de software através de duas abordagens diferentes: (1) a abordagem Web, onde informações de componentes podem ser pesquisadas em grandes máquinas de busca tais como *Google*, *AltaVista*, etc; e (2) a abordagem de bibliotecas digitais, que provê recursos para a recuperação de variados componentes (diagramas, modelos, documentos, etc) armazenados em repositórios heterogêneos e distribuídos pela Internet.

A arquitetura *Odyssey Search* (figura 3.1) é baseada em múltiplos agentes, que utilizam técnicas como mediadores, perfil de usuário e hipermídia adaptativa para o armazenamento, busca e recuperação de informações relacionadas a componentes de um determinado domínio de aplicação. O sistema é dividido em três tipos de agentes:

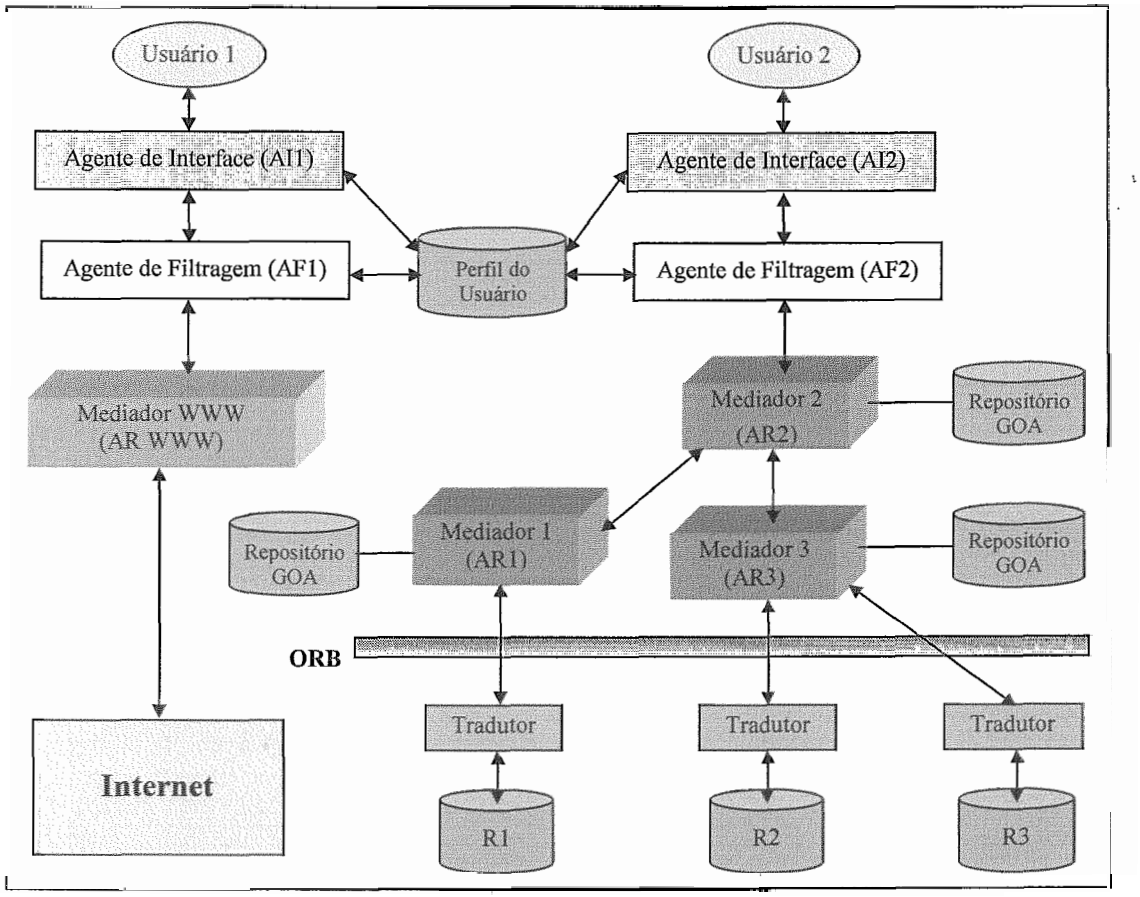


Figura 3.1 - Arquitetura *Odyssey-Search*.

- Agente de Interface (AI) - responsável pela adaptação da interface, de acordo com o perfil do usuário;
- Agente de Filtragem (AF) - responsável pela filtragem dos resultados das pesquisas por componentes, de acordo com a necessidade de informações por parte do usuário;
- Agente de Armazenamento e Recuperação (AR) - responsável pelo armazenamento e pela recuperação de componentes locais, e pela busca e recuperação de informações de componentes armazenados em repositórios distribuídos na Internet ou publicados na Web.

Através do ambiente *Odyssey*, um usuário é capaz de construir novos componentes (diagramas, modelos, documentos, etc) ou pesquisar por componentes já

prontos através da *Odyssey-Search*. A partir das necessidades iniciais do usuário em relação à aplicação a ser desenvolvida e às escolhas e informações que ele disponibilize, o Agente de Interface (AI) constrói uma base de informações que constitui o perfil do usuário. Esta base é alterada dinamicamente pelo AI a medida que o usuário vai navegando pelo ambiente, de modo a acomodar as necessidades e os requisitos do usuário. O perfil do usuário é utilizado pelo Agente de Filtragem (AF) a cada pesquisa, a fim de filtrar as informações de componentes retornadas ao usuário pelo Agente de Armazenamento e Recuperação (AR).

Todos os componentes construídos dentro do ambiente *Odyssey* são estruturados segundo o padrão orientado a objetos (OO) e armazenados pelo AR. A arquitetura dos ARs se baseia no conceito de mediadores, cada qual armazenando seus componentes, segundo um determinado domínio de aplicação, dentro de uma base de dados própria no Servidor GOA (MATTOSO et al., 2000). O uso de ontologias ajuda a estabelecer relacionamentos entre componentes de um mesmo mediador ou de mediadores diferentes. Um componente externo, publicado em um repositório remoto qualquer, pode também ser adicionado à base de dados de um mediador mediante à construção de um tradutor, capaz de converter o componente do seu formato original para o formato interno OO. Um mediador especial (mediador WWW) provê acesso a componentes publicados na Web, para os quais as referências são armazenadas.

3.6 – Considerações Finais

Muitas empresas desenvolvedoras de software têm buscado nos últimos anos a modernização do seu processo de produção, de modo a se tornarem mais competitivas e atenderem com maior eficiência as novas exigências do mercado (sistemas mais modernos, rápidos e confiáveis). A fim de se adaptarem a estas novas necessidades, as empresas estão buscando na técnica de reutilização de software a solução prática para atingir tal modernidade.

Dentro deste contexto, um repositório de componentes desempenha um papel chave no processo de armazenamento, organização e disponibilização de componentes de software que podem ser reutilizados em vários projetos de sistemas. Por isso, muitas

empresas estão buscando soluções para desenvolver seus próprios repositórios. Com isso, informações de diversos componentes, produzidos por vários desenvolvedores em diferentes setores da empresa, poderão ser integradas em uma grande biblioteca digital de informações sobre componentes. Desta forma, todos os desenvolvedores poderão ter acesso a qualquer componente de software produzido dentro da organização de uma forma mais rápida e fácil, incentivando a reutilização.

Somado a isto, a Internet tem se confirmado nos últimos anos como o maior e o mais popular meio para o compartilhamento de informações entre pessoas no mundo. Por isso, é esperado que esta se torne também uma grande fonte de reutilização para organizações e desenvolvedores em geral. Por este motivo, repositórios construídos dentro de uma organização devem considerar também a possibilidade de busca por componentes em repositórios publicados na Internet, aumentando, desta forma, a possibilidade de descoberta de novos componentes.

No entanto, não adianta de nada aumentar em grande proporção a quantidade de componentes acessíveis se não existir um mecanismo eficiente de organização e procura por estes componentes (SEACORD, 1999). Organizar componentes em diferentes repositórios divididos de acordo com o seu domínio de aplicação, utilizar metadados para descrever com maior semântica as informações referentes a um componente, desenvolver mecanismos de pesquisas eficientes para buscas de componentes dentro do repositório, dentre outras propostas, são alguns exemplos de técnicas que podem ser utilizadas na montagem de uma biblioteca digital para aumentar a eficiência geral, tanto em velocidade quanto em qualidade, da busca por componentes de software na Internet.

Ao analisar cada um dos trabalhos descritos neste e no capítulo anterior, é possível verificar que todos os sistemas descritos apresentam contribuições importantes relacionadas à integração de dados heterogêneos e distribuídos, problema típico a ser tratado em um sistema de publicação e recuperação de componentes de software na Internet. No entanto, dentre todos os sistemas apresentados, o *Le Select* (seção 2.3.3) se destaca por: (1) publicar e integrar de dados e programas; e (2) publicar metadados associados a qualquer dado ou programa por ele publicado. Estas duas características são bastante úteis a um sistema de disponibilização de componentes de software, que deve ser capaz de publicar informações bastante variadas, tais como projetos,

especificações, modelos, códigos fonte, ferramentas auxiliares, programas de teste, etc, além de utilizar os recursos de metadados para auxiliar no processo de documentação e busca de componentes. No entanto, o *Le Select* é um sistema que se restringe apenas a questões de interoperabilidade, não tratando aspectos semânticos de integração de metadados, o que consequentemente dificulta a busca de informações distribuídas de forma transparente para o usuário.

Assim, como é descrito no próximo capítulo, a *ComPublish* é um sistema que reúne as idéias e funcionalidades presentes tanto no *Le Select* quanto na *Odyssey Search*, integrando ambos sistemas a fim de oferecer serviços mais abrangentes para publicação, integração de informações e busca de uma grande variedade de componentes de software armazenados em um repositório qualquer da Internet.

4 Projeto do Sistema ComPublish

4.1 – Introdução

O crescimento da atividade de reutilização dentro das empresas desenvolvedoras de software, como parte do processo de modernização de seus sistemas de produção, tem feito crescer, em larga escala, a produção e, sobretudo, o reaproveitamento de componentes de software no desenvolvimento de sistemas. Conforme dito anteriormente, a reutilização pode proporcionar, entre outras vantagens, a gerência da complexidade, a redução dos custos e do tempo de desenvolvimento, além de aumentar a confiabilidade dos sistemas produzidos através do uso de componentes previamente testados. No entanto, não é possível reutilizar componentes prontos sem saber como e onde encontrá-los. Dentro deste contexto, o repositório surge como uma solução para o armazenamento de componentes, tornando-os públicos e acessíveis a todos. Tal repositório deve ser capaz de gerenciar estruturas de dados dos mais variados tipos, tamanhos e complexidade (atributos típicos dos componentes de software). Atualmente, muitos repositórios podem ser encontrados na Internet. Localizá-los, no entanto, não é trivial. Por isso, a construção de uma infra-estrutura capaz de integrar informações de componentes armazenados em repositórios distribuídos torna-se uma questão importante para um processo de reutilização, sobretudo com o crescimento da Internet. Além disso, tal infra-estrutura deve prover também recursos avançados para pesquisa de componentes. As estratégias de busca atualmente disponíveis na Web, baseadas em sua maioria no uso de palavras chave, permitem a execução de pesquisas de forma limitada, originando respostas muito volumosas e genéricas. Aliado a isto, a falta de informações descritivas (metadados) relacionadas aos componentes publicados, além dos problemas associados a questões semânticas que envolvem a existência de diferentes vocabulários descrevendo informações semelhantes dentro de um mesmo contexto, diminuem ainda mais a precisão e qualidade dos mecanismos de busca existentes.

Dentro deste contexto, nossa pesquisa teve como objetivo explorar novas características para o processo de publicação, busca e recuperação de componentes de software na Internet, buscando propor soluções para problemas como: (1) a grande heterogeneidade de componentes disponibilizados na Internet (diferentes modelos de dados); (2) a distribuição e a interoperabilidade entre os vários repositórios de componentes publicados na rede (protocolos variados de acesso via rede, diferentes sistemas operacionais, restrições de acesso, etc); (3) a falta de informações capazes de prover uma maior riqueza semântica para a maioria dos componentes publicados, o que dificulta a execução de um mecanismo de pesquisa de maior precisão e a compreensão por parte do desenvolvedor dos componentes encontrados; (4) a necessidade de uma abrangência maior dos mecanismos de publicação com relação à possibilidade de se publicar uma variedade maior de componentes (não só código, como tratado pela maioria dos trabalhos, mas também modelos, diagramas, programas, etc); e (5) a necessidade de recursos mais eficientes para pesquisa de componentes, diferente das estratégias de busca atualmente disponíveis na Web, baseadas em sua maioria no uso de palavras chave.

Baseado nestes problemas, esta dissertação propõe o sistema *ComPublish*, uma ferramenta que constitui uma extensão do trabalho de armazenamento e recuperação de componentes já iniciado por BRAGA (2000b), e que visa oferecer uma nova solução para a publicação, integração de metadados, localização e recuperação de componentes de software em repositórios locais e na Internet. Assim, este capítulo destaca inicialmente, na seção 4.2, as funcionalidades oferecidas pelo *ComPublish*, tais como o processo de publicação de componentes, a pesquisa e recuperação de componentes, dentre outras. A seguir, a seção 4.3 aborda as principais características do projeto *ComPublish*. Por fim, a seção 4.4 aborda o uso do *ComPublish* no contexto do ambiente de reuso *Odyssey*.

4.2 – Funcionalidades

ComPublish é um sistema que tem por objetivo auxiliar desenvolvedores de software a publicar, pesquisar e recuperar variados artefatos de software na Internet, tal

como modelos, diagramas, código fonte, documentos, programas ou qualquer outro tipo de artefato utilizado ou produzido nas diferentes etapas do processo de desenvolvimento de um software. Para fins de organização, o *ComPublish* provê uma visão lógica de vários componentes publicados local ou remotamente, divididos segundo seu domínio de aplicação. Seus principais serviços são: (1) descrever e publicar componentes armazenados em um repositório qualquer da Internet; (2) integrar as informações dos componentes publicados de acordo com o seu domínio de aplicação; (3) prover o gerenciamento de ontologias capazes de integrar termos de diferentes domínios; (4) prover mecanismos de pesquisa e a recuperação dos componentes publicados.

Nas sub-seções a seguir, são descritos em maiores detalhes cada um dos serviços oferecidos pelo *ComPublish*.

4.2.1 – Serviços para Publicação de Componentes

A fim de facilitar o processo de publicação de componentes por parte do desenvolvedor, o *ComPublish* oferece um módulo completo para publicação de informações através do servidor *Le Select* (seção 2.3.3). O *Le Select* oferece muitas facilidades para publicação de informações na Internet, dentre as quais destacamos: fácil instalação, pode ser executado em diferentes sistemas operacionais (desenvolvido segundo a tecnologia Java), variados protocolos de acesso (HTTP, FTP, *Socket*, CORBA), facilidades para a construção de tradutores que permitem o acesso a diferentes fontes de dados, suporte a publicação de metadados, dentre outros recursos.

4.2.2 – Serviços para Integração de Informações de Componentes

A fim de solucionar o problema de distribuição de componentes armazenados em repositórios espalhados por toda a Internet e publicados via servidor *Le Select*, o *ComPublish* oferece serviços de integração de informações de componentes agrupados pelo conceito de domínio de aplicação (KRUEGER, 1992) (MILLI et al., 1995). Tais informações são exportadas por cada servidor *Le Select* na rede e armazenadas em um

repositório de metadados gerenciado pelo *ComPublish*. Isto provê ao usuário de pesquisa um acesso transparente aos componentes publicados, independente de suas localizações remotas.

Os serviços de integração, acessíveis apenas pelos administradores do sistema, devem ser acionados a cada publicação de um novo componente, ou em caso de atualização das informações de um componente já publicado.

4.2.3 – Serviços para Acesso a Ontologias

Como extensão dos serviços de integração de informações de componentes, o *ComPublish* oferece ainda serviços de acesso a ontologias (SOUZA et al., 2002). Ontologias no *ComPublish* são termos que permitem associar informações similares representadas por diferentes vocabulários. Desta forma, o uso de ontologias ajuda a estabelecer relacionamentos entre informações de componentes de um mesmo domínio ou de domínios diferentes, auxiliando usuários na descoberta de novos componentes. A manipulação de ontologias dentro do *ComPublish* é um assunto bem mais extenso e que é desenvolvido como parte de uma outra tese de mestrado atualmente em fase de finalização (OLIVEIRA, 2002).

4.2.4 – Serviços para Pesquisa e Recuperação de Componentes

Além dos mecanismos tradicionais de consulta da Web baseados em palavras chave, o *ComPublish* oferece recursos avançados de pesquisa baseados em uma linguagem de consulta segundo o padrão OQL. Consultas são direcionadas a um domínio de aplicação específico de componentes e são processadas sobre as bases de metadados gerenciadas pela camada de integração *ComPublish*. Os resultados de cada consulta, gerados no formato de um documento XML, contêm as descrições de todos os componentes encontrados com seus respectivos sites de publicação. De posse destas informações, o usuário é capaz de fazer uma avaliação prévia do componente e executar sua transferência, caso este possa vir a atender suas necessidades.

4.3 – Arquitetura

O *ComPublish* foi projetado segundo uma arquitetura em camadas, dividida em três diferentes níveis (figura 4.1): (1) O nível mais baixo corresponde à camada de sites publicados, constituída pelos repositórios remotos de componentes e pelo módulo publicador de componentes, o *LeSelect*; (2) O nível intermediário corresponde à camada de integração, baseada no conceito de Mediadores e Tradutores, seguindo os padrões definidos pela arquitetura HIMPAR; e (3) O nível mais alto corresponde à camada de aplicações clientes, que acessam os serviços dos mediadores da camada de integração para pesquisa e recuperação de componentes.

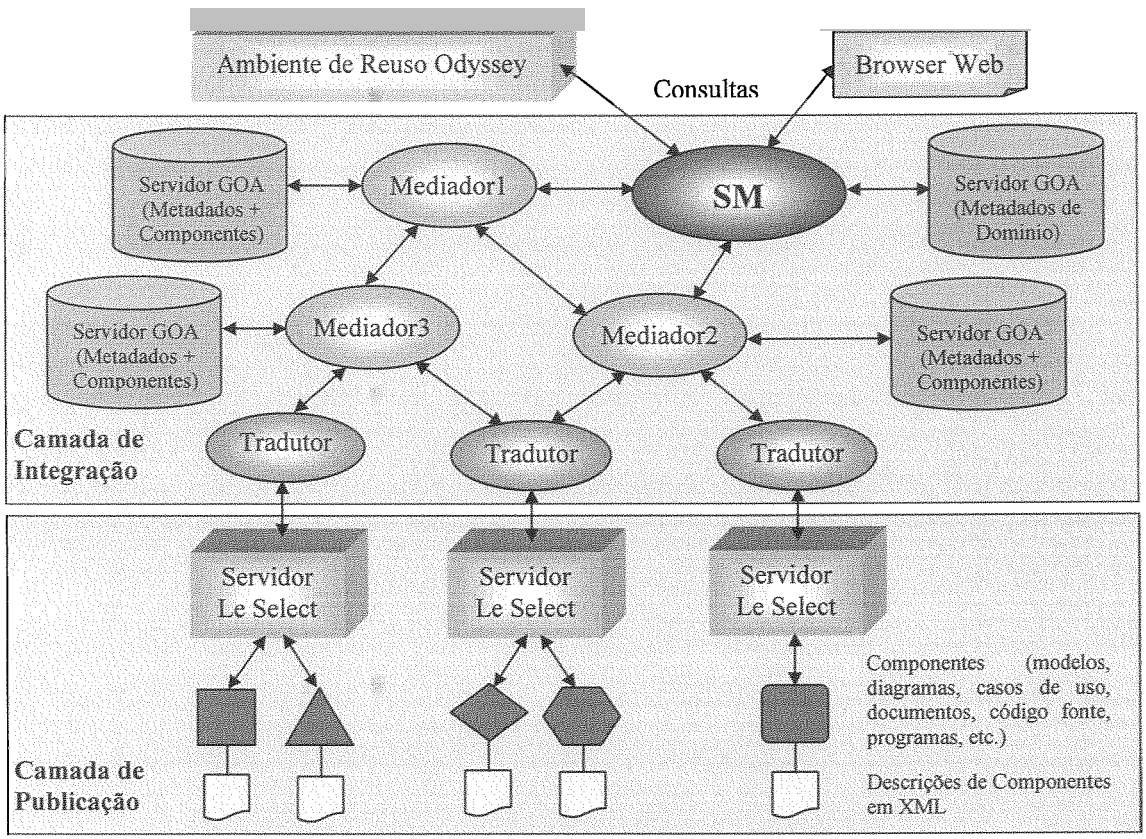


Figura 4.1 – Arquitetura do Sistema *ComPublish*.

A seguir, são descritas, em maiores detalhes, cada uma das três camadas mencionadas no parágrafo anterior.

4.3.1 – Camada de Repositórios Publicados

Esta camada é constituída pelo conjunto de repositórios remotos, contendo variados componentes de software, disponibilizados por desenvolvedores interessados em publicar seus trabalhos em qualquer parte do mundo. Um repositório pode ser qualquer gerenciador de dados, desde um simples sistema de arquivos a um poderoso sistema de banco de dados. O módulo publicador de dados, representado pelo servidor *Le Select*, fornece os mecanismos necessários para abstração da heterogeneidade de diferentes repositórios de componentes a serem publicados (como descrito na seção 2.3.5).

O *Le Select* oferece também serviços para a publicação de metadados associados aos dados publicados. Assim, para publicar um componente, o publicador deve antes construir um documento no padrão XML contendo um conjunto de especificações sobre o componente. Este documento tem por objetivo servir como fonte de metadados para a integração e a pesquisa de componentes, além de ajudar no próprio entendimento do componente. A fim de auxiliar o publicador a desenvolver uma documentação para seus componentes e ajudar a estabelecer um padrão de documentação genérico para componentes de software, o *ComPublish* oferece um conjunto de três modelos pré-definidos de DTDs XML (descritos no apêndice A), divididos nas categorias dados, serviços e código fonte. Estes modelos foram formulados a partir da observação de padrões já existentes (seção 3.4), porém voltados sempre para descrever componentes do tipo código. Os modelos sugeridos buscam agregar tanto características genéricas a todos os componentes (como nome, autor, versão, domínio, etc), quanto características mais específicas (como linguagem, sistema operacional, etc) de acordo com a categoria do componente. No entanto, é importante ressaltar que, de acordo com o domínio de aplicação do componente, novos termos comuns ao vocabulário do domínio deverão ser agregados à documentação do componente, criando assim novas especializações dos modelos sugeridos. Portanto, caberá ao gerente do domínio definir a estrutura de documentação a ser utilizada para um dado domínio de aplicação.

Uma vez que um repositório tenha sido publicado, todos os componentes armazenados neste repositório passam a ser referenciados por um mediador específico dentro da camada de integração, através das documentações relativas a cada

componente exportadas pelo módulo publicador. Desta forma, cada componente se torna acessível às consultas submetidas por usuários na camada de aplicações. Uma vez descoberto e selecionado, o módulo publicador oferece serviços de *ftp* para transferir o componente para a máquina local do usuário.

4.3.2 – Camada de Integração de Informações de Componentes

Uma vez que diferentes repositórios de componentes de software tenham sido publicados na Internet via *Le Select*, estes ainda constituem "ilhas de informações" isoladas uma das outras. Nestas condições, já é possível ao usuário ter acesso aos componentes publicados em um site específico. Para isso, basta acessar via browser o endereço do site desejado, associado a porta *http* padrão disponibilizada pelo servidor *Le Select* (porta 3080), por exemplo: http://oxossi.cos.ufrj.br:3080. Como visto na figura 4.2, o *Le Select* provê aos usuários uma visualização simplificada na forma relacional (tabular) de todos os componentes publicados, e mais uma listagem dos documentos no formato XML produzidos para cada componente. Além da visualização, o sistema oferece ao usuário uma linguagem de consulta bem simples, no padrão SQL, para execução de pesquisas sobre a visão relacional dos componentes publicados.

A fim de prover um sistema de consulta mais amplo, que englobe todos os sites de componentes publicados na Internet através do *Le Select* de forma transparente para o usuário, a arquitetura *ComPublish* desenvolve uma camada de integração capaz de interligar estes sites, agrupando virtualmente os componentes publicados de acordo com o seu domínio de aplicação.

Dentre os modelos de arquitetura de integração de dados heterogêneos e distribuídos apresentados no capítulo 2, optou-se por utilizar neste trabalho o modelo de mediadores e tradutores, pois o conceito de mediador é o que melhor se adapta ao conceito de domínio de aplicação, ou seja, cada mediador pode ser associado a uma visão lógica de componentes distribuídos que pertençam a um mesmo domínio de aplicação.

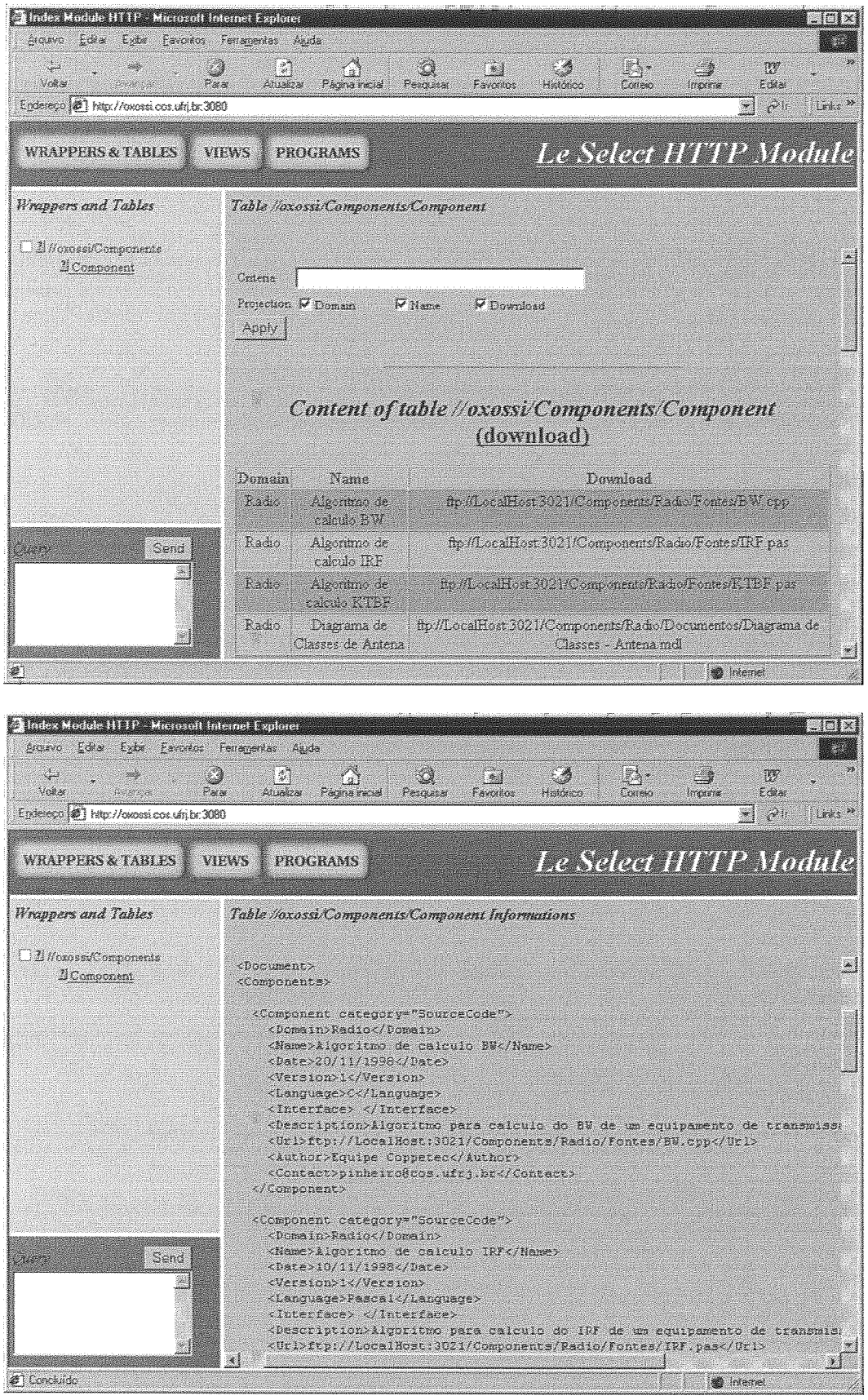


Figura 4.2 – Visualização de componentes publicados via *Le Select*.

Sendo assim, o projeto da camada de integração formulada para o *ComPublish* segue os conceitos de mediação propostos pela arquitetura HIMPAR (seção 2.3.2). A seguir, são apresentados os detalhes de projeto de cada um dos módulos desta camada dentro da arquitetura *ComPublish*.

4.3.2.1 – Mediadores

O mediador é o módulo responsável por prover uma visão lógica de todos os componentes publicados no sistema, segundo um determinado domínio de aplicação. Para prover uma visão integrada, o mediador importa de cada módulo publicador *Le Select*, via tradutor, todas as descrições de componentes publicados para o seu domínio de aplicação, e as armazenam dentro de sua base de metadados interna. Através destes metadados, o mediador consegue informar aos clientes do sistema quais componentes estão disponíveis, onde buscá-los, além de oferecer serviços de consultas a componentes segundo critérios de pesquisa estabelecidos pelo usuário.

Da mesma forma que um domínio pode possuir vários sub-domínios associados, um mediador pode também possuir vários sub-mediadores a ele ligados. Desta forma, consultas submetidas a um mediador são também repassadas aos sub-mediadores associados, e todas as descrições de componentes encontradas em cada um deles retornadas ao usuário.

Um mediador especial, denominado *Service Manager* (SM), é responsável por gerenciar a lista de todos os mediadores ativos na camada de integração, e por atender às solicitações de consultas das aplicações clientes. Este mediador oferece serviços de consulta aos domínios de aplicação disponíveis no sistema e às informações de componentes de um domínio de aplicação específico.

O modelo de dados interno de todos os mediadores segue o padrão orientado a objetos (OO). Todas as informações colhidas dos sites remotos relacionadas aos componentes publicados chegam no mediador no formato XML e são convertidas internamente para modelo OO. Cada mediador é capaz de gerenciar um esquema próprio, independente dos demais. Desta forma, o gerente de domínios possui a

flexibilidade para definir diferentes esquemas por mediador, cada qual com termos específicos do domínio de aplicação que o mediador representa.

No *ComPublish*, todos os serviços relacionados à gerência de metadados (armazenamento, recuperação e processamento de consultas) não são implementados diretamente dentro do módulo mediador. Estas funcionalidades, por serem complexas e típicas de um SGBD, foram desenvolvidas em um sistema a parte, ao qual o mediador é ligado: o Servidor de Objetos GOA (MAURO, 1997) (MATTOSO et al., 1994, 2000 e 2002). O GOA é um sistema desenvolvido pela COPPE/UFRJ que provê serviços para gerência de objetos em disco, segundo os padrões ODMG. Dentre estes serviços, estão a inserção, remoção e consulta de objetos, e a importação e exportação de documentos XML.

O processamento de consultas, assim como o modelo de dados, segue o padrão OO. Todas as consultas (no padrão OQL) enviadas a um mediador são repassadas ao Servidor GOA e processadas sobre a base de metadados armazenada. As respostas de saída, originalmente geradas também no formato OO, são, no entanto, exportadas no formato XML.

4.3.2.2 – Tradutores

O tradutor é o módulo responsável por fazer a ligação entre os mediadores e os repositórios de componentes remotos, tratando variados problemas de interoperabilidade, tais como: diferentes sistemas operacionais, diferentes protocolos de acesso ao repositório, diferentes formas de consulta e extração das informações do repositório, dentre outros aspectos. Dependendo do tipo da fonte de dados (sistemas de arquivos, SGBDs Relacional ou OO, dentre outras), a complexidade de se construir um tradutor pode ser bastante elevada. No *ComPublish*, a maioria destes problemas são tratados pelo módulo publicador *Le Select*, que oferece facilidades para a construção de tradutores através de sua fábrica de tradutores (seção 2.3.3).

Portanto, dentro do *ComPublish*, mediadores têm acesso aos componentes publicados em um repositório remoto através dos serviços oferecidos pelo *Le Select*. A fim de manter a flexibilidade da arquitetura, os serviços de comunicação entre

mediadores e o *Le Select* foram isolados em um módulo tradutor a parte, ao invés de serem embutidos diretamente no mediador. Isso permite que outros tradutores, independente do *Le Select*, possam ser adotados mais facilmente pela arquitetura, uma vez que se mantenha um conjunto de serviços semelhantes de acesso aos repositórios. Este módulo tradutor extra deverá, portanto, ser instalado junto com o servidor *Le Select* na máquina onde está o componente a ser publicado.

4.3.3 – Camada de Aplicações Clientes

O acesso de aplicações clientes ao *ComPublish* é feito através do *Service Manager*, que disponibiliza serviços de consultas e pedidos de transferência de componentes ao usuário do sistema. No próximo capítulo, são descritos, em maiores detalhes, os passos necessários para a construção de uma aplicação cliente, mostrando, inclusive, exemplos de aplicações clientes já construídas para efeito de gerência e testes do sistema.

4.4 – O ComPublish no Contexto do Projeto Odyssey

O Projeto *Odyssey* (WERNER et al., 1999, 2000 e 2002), desenvolvido pela COPPE/UFRJ, propõe uma infra-estrutura de apoio à reutilização de componentes de software, com o objetivo de dar suporte à criação e recuperação de componentes que possam ser reutilizados em todas as fases do desenvolvimento de uma dada aplicação. Dentro deste projeto, muitos trabalhos vêm sendo desenvolvidos com o objetivo de explorar diferentes atividades relacionadas ao processo de desenvolvimento com reutilização, sejam elas de desenvolvimento, planejamento, acompanhamento gerencial ou controle de qualidade. Como exemplo, pode-se citar as ferramentas de elicitação de requisitos (TRANNIN et al., 1999), diagramadores genéricos, gerador de código executável (WERNER et al., 2000), especificação de arquiteturas específicas de domínio (XAVIER, 2000), planejamento e análise de risco (BARROS, 2000), acompanhamento de processos (MURTA, 2000), navegador inteligente (BRAGA et al.,

2000a), armazenamento e recuperação de componentes (BRAGA, 2000b), documentação de componentes (MURTA, 1999), dentre outras.

O *Odyssey*, assim como definido por KRUEGER (1992), divide o processo de reutilização em duas fases bem distintas: o desenvolvimento **para** reutilização, representado pela *Odyssey-DE*, e o desenvolvimento **com** reutilização, representado pela *Odyssey-AE* (BRAGA, 2000b). A *Odyssey-DE* provê suporte às diferentes etapas da engenharia de domínio, através da análise, projeto e implementação do domínio. A *Odyssey-AE*, por sua vez, provê suporte às etapas da engenharia de aplicação, apoiando atividades como o armazenamento, a busca, a recuperação e a integração de componentes produzidos pela *Odyssey-DE*.

O *ComPublish* é inserido no contexto do projeto *Odyssey* como uma ferramenta de apoio à busca, à recuperação e ao armazenamento de componentes externos publicados por terceiros em um *site* qualquer da Internet, e que possam ser agregados à infra-estrutura de reutilização. Nosso trabalho constitui uma extensão do trabalho de armazenamento e recuperação de componentes iniciado por BRAGA (2002b) em sua tese de doutorado, onde o *ComPublish* apresenta: (1) uma proposta mais abrangente para publicação de componentes na Internet, através da utilização de metadados em XML; e (2) novos mecanismos de integração e consulta de informações de componentes, através da utilização de metodologias que mesclam os conceitos das tecnologias XML e de Orientação a Objetos. O *ComPublish* é integrado ao *Odyssey* através do *CompAgent*, um sistema de agentes de filtragem para *Web* desenvolvido por COSTA (2002), que proporciona um mecanismo de pesquisa por componentes mais amplo em toda Internet, através da associação das abordagens de busca na *Web* e busca em biblioteca digitais (SOUZA et al., 2001).

5 Implementação do Sistema ComPublish

5.1 - Introdução

Como visto no capítulo 4, o sistema *ComPublish* é projetado segundo uma arquitetura em camadas, construída com base em idéias de outras arquiteturas e produtos em desenvolvimento: (1) a publicação de dados e serviços remotos é realizada pelo sistema *LeSelect* (seção 2.3.3); (2) a camada de integração, baseada no conceito de mediadores e tradutores, segue os princípios da arquitetura HIMPAR (seção 2.3.2); e (3) o armazenamento de metadados e componentes locais, e mais o processamento de consultas do mediador (segundo o modelo orientado a objetos) são feitos pelo sistema GOA (MATTOSO et al., 1994, 2000 e 2002) (MAURO, 1997).

Neste capítulo, são apresentados os principais aspectos de implementação realizados sobre cada um dos protótipos listados acima, a fim de viabilizar a construção do sistema *ComPublish*. Com exceção do *Le Select* (desenvolvido pelo grupo *Caravel* no INRIA na França), para o qual nosso esforço, em relação à implementação, se restringiu a desenvolver uma API Cliente de comunicação com o seu servidor, os demais protótipos (HIMPAR e GOA) sofreram diversas adaptações, acréscimo de funcionalidades e correções de erros, tornando-os mais modularizados, extensíveis e robustos. Assim, buscando descrever todo este trabalho, este capítulo é organizado da seguinte forma: A seção 5.2 descreve a implementação de cada um dos módulos que compõe a camada de integração *ComPublish*. A seção 5.3, por sua vez, descreve as ferramentas de gerência construídas para administrar os módulos da camada de integração. Por fim, a seção 5.4 apresenta um exemplo no domínio de Telecomunicação para mostrar a viabilidade de uso do *ComPublish*.

5.2 – Módulos da Camada de Integração

Todos os módulos da camada de integração *ComPublish*, que incluem mediadores, tradutores e o servidor GOA, foram implementados na linguagem C++. Em relação à comunicação, mediadores e tradutores trocam informações entre si segundo o padrão CORBA. Desta forma, cada um destes módulos são implementados a partir da definição de uma interface IDL[6], como descrito nas próximas sub-seções. Para esta primeira versão de implementação do sistema *ComPublish*, tanto o ambiente de desenvolvimento (*Borland C++ Buider 5.0*), quanto a versão do CORBA (*Borland VisiBroker 4.0*) utilizados, são fornecidos pela empresa *Borland* (BORLAND) na plataforma *Windows*.

A seguir, são apresentados os principais detalhes de implementação de cada um dos módulos desta camada. A figura 5.1 ilustra o diagrama de classes, segundo notação UML, do projeto desta implementação.

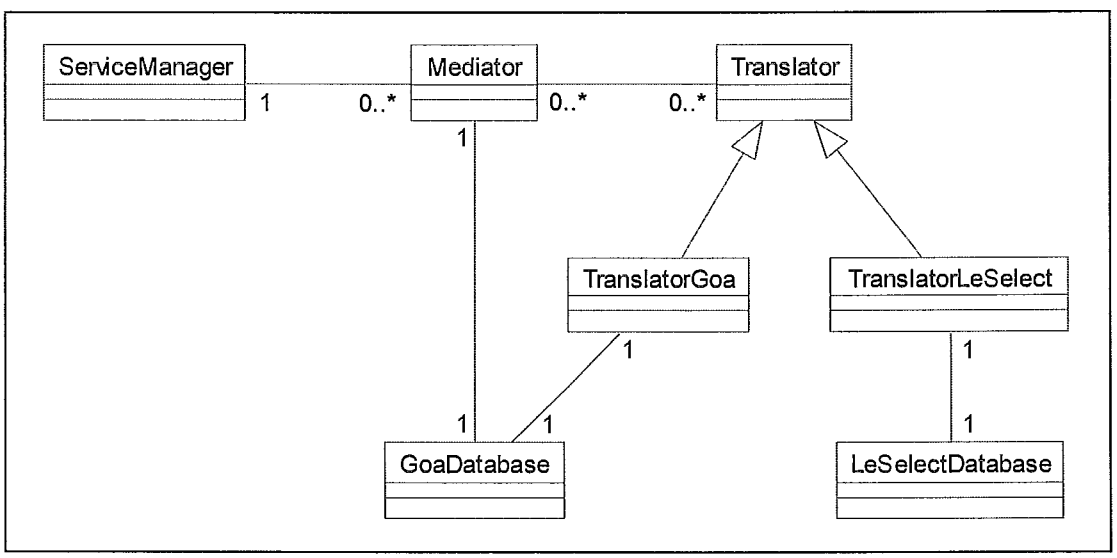


Figura 5.1 – Diagrama de classes da camada de integração *ComPublish*.

[6] IDL (*Interface Definition Language*) é a linguagem de definição da interface de 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.

5.2.1 – Tradutores

O tradutor é o módulo responsável por fazer o elo de comunicação entre mediadores e as fontes de dados. No *ComPublish*, os problemas de interoperabilidade de acesso às fontes de dados são tratados pelo módulo publicador *Le Select*. Sendo assim, a principal funcionalidade do tradutor é prover acesso aos serviços oferecidos pelo servidor *Le Select*, dentre os quais destacam-se a consulta ao catálogo de componentes publicados e a recuperação dos metadados associados a estes componentes.

A fim de possibilitar a comunicação via CORBA entre mediadores e tradutores, cada tradutor é implementado a partir da interface IDL apresentada na figura 5.2. Esta interface encapsula um conjunto de métodos básicos para acesso a qualquer fonte de dados, como, por exemplo, abertura e fechamento de base (*openBase* e *closeBase*), consultas ao esquema e dados da base (*getSchema*, *getClassDefinition* e *executeQuery*), e consulta aos metadados da base (*getMetadata*). Todos os dados recuperados da fonte pelo tradutor e repassados a um mediador são encapsulados pela estrutura *TObject*. Esta interface genérica possibilita a construção de diferentes tradutores, como visto na figura 5.1, onde, além do tradutor *Le Select*, é ilustrada também a construção do tradutor GOA para acesso direto às bases de dados do servidor GOA.

```
interface TranslatorInterface
{
  struct TObject
  {
    string className;
    string attributes;
    string values;
  };

  typedef sequence <TObject>
    TObjectList;
  // Base
  boolean closeBase(in string baseName);
  boolean openBase(in string baseName);
  // Metadata
  string getMetadata();
  // Query
  TObjectList executeQuery(in string query);
  // Schema
  string getClassDefinition(in string className);
  string getSchema();
};
```

Figura 5.2 – Interface IDL do tradutor para comunicação com os mediadores.

Já a comunicação de tradutores com *Le Select* e de tradutores com o GOA é feita via *socket*. Para acesso aos serviços oferecidos pelo servidor *Le Select*, foi implementada em C++ a *API Cliente Le Select*, um conjunto de classes projetadas

segundo o modelo visualizado na figura 5.3 e que são instanciadas pelo tradutor. A classe *LeSelectDatabase* encapsula os métodos de conexão e todos os métodos de acesso aos serviços oferecidos pelo servidor *Le Select*. As classes *LeSelectTable* e *LeSelectQuery* permitem, no lado cliente, a estruturação em memória (na forma tabular) dos dados recuperados do servidor.

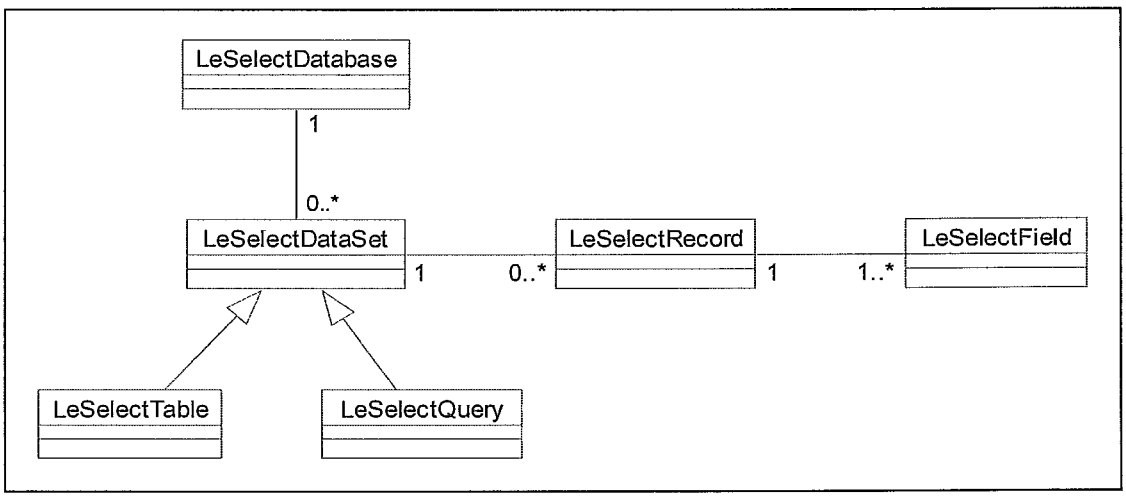


Figura 5.3 – Diagrama de Classes da API Cliente *Le Select*.

Para o acesso aos serviços oferecidos pelo servidor GOA, por sua vez, foi implementada em C++ a *API Cliente GOA* (figura 5.4). De modo análogo às classes da *API Le Select*, a classe *GoaDatabase* encapsula os métodos de conexão e todos os métodos de acesso aos serviços oferecidos pelo servidor GOA. As classes *GoaDataSet* e *GoaObject* permitem, no lado cliente, a estruturação em memória (na forma de objetos) dos dados recuperados do servidor.

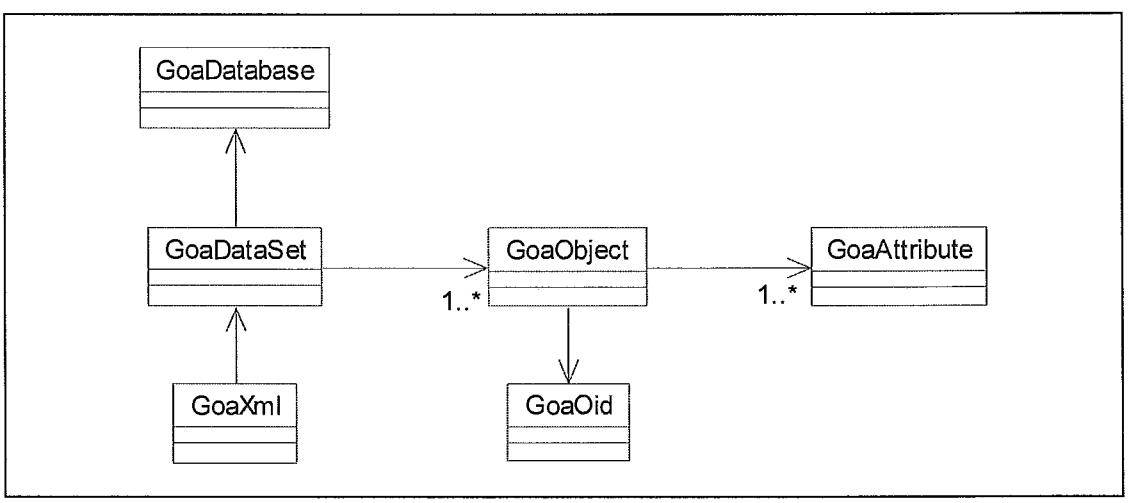


Figura 5.4 – Diagrama de Classes da API Cliente GOA.

5.2.2 – Mediadores

O mediador é o módulo responsável por gerenciar os metadados relacionados tanto aos componentes publicados remotamente (segundo um determinado domínio de aplicação), quanto os metadados relacionados às informações internas de gerência da camada de integração, provendo serviços de armazenamento, consulta e recuperação destes metadados. Por serem estes serviços típicos de um SGBD, o mediador utiliza os recursos do servidor de objetos GOA para tais atividades, como descrito na seção 5.2.4.

Assim como o tradutor, o meio de comunicação disponibilizado pelo mediador para solicitação de seus serviços é feito através do padrão CORBA. Desta forma, cada mediador é implementado a partir da interface IDL apresentada na figura 5.5.

```
interface MediatorInterface
{
  struct TOid {
    short baseVolume;
    short pageIndex;
    long pageNumber;
  };
  struct TObject {
    string attributes;
    string className;
    string values;
  };
  struct TSubMediator {
    string name;
    TOid oid;
  };
  struct TTranslator {
    string name;
    TOid oid;
  };
  typedef sequence <TObject>
    TObjectList;
  typedef sequence <TSubMediator>
    TSubMediatorList;
  typedef sequence <TTranslator>
    TTranslatorList;
  // Metadata
  string importMetadata(in string translatorName);
  void storeMetadata(in string metadata, in string
    location);
  // Query
  string executeMetadataQuery(in string query);
  TObjectList executeQueryInAllTranslators(in
    string Query);
  TObjectList executeQueryInSingleTranslator(
    in string query, in string translatorName);
  // Schema
  string getClassDefinitionMediator(in string className);
  string getClassDefinitionTranslator(in string
    className, in string translatorName);
  string getSchemaMediator();
  string getSchemaTranslator(in string
    translatorName);
  // Sub Mediators
  boolean addSubMediator(in TsubMediator subMediator);
  TSubMediatorList getSubMediators();
  boolean removeSubMediator(in string subMediatorName);
  // Translators
  boolean addTranslator(in TTranslator translator);
  TTranslatorList getTranslators();
  boolean removeTranslator(in string translatorName);
};
```

Figura 5.5 – Interface IDL do mediador.

5.2.3 – Gerente de Serviços (*Service Manager*)

O *Service Manager* (SM) é um mediador especial construído com o objetivo de prover serviços de gerência de todos os demais mediadores ativos na camada de integração e serviços de consulta às informações de componentes disponibilizadas por um determinado mediador.

O SM, assim como os demais mediadores da camada de integração, disponibiliza uma interface IDL (figura 5.6) para solicitação de seus serviços via CORBA. Estes serviços incluem: (1) a adição, remoção e consulta aos mediadores existentes; e (2) consulta às informações de componentes de um mediador específico.

```
interface ServiceManagerInterface
{
  struct TOid
  {
    short baseVolume;
    short pageIndex;
    long pageNumber;
  };
  struct TMediator
  {
    string name;
    TOid oid;
  };
  typedef sequence <TMediator>
    TMediatorList;
  // Mediator
  boolean addMediator(in TMediator mediator);
  TMediatorList getMediators();
  boolean removeMediator(in string mediatorName);

  // Query
  string executeMetadataQuery(in string query, in
    string mediatorName);
};
```

Figura 5.6 – Interface IDL do Gerente de Serviços.

5.2.4 – Gerente de Metadados GOA

Todos os serviços de gerência de metadados referentes a cada um dos mediadores da camada de integração são implementados dentro do gerente de objetos GOA. Projetado segundo um modelo Cliente/Servidor, seu módulo servidor, também implementado em C++, oferece serviços de inserção, remoção e consulta (segundo linguagem de consulta no padrão OQL) a objetos armazenados em disco.

Todo mediador, quando ativado, deverá possuir uma base de dados no servidor GOA para armazenar seus metadados. Os esquemas destas bases deverão estar preparados para armazenar informações relacionadas aos metadados internos de gerência da camada de integração e informações relacionadas aos metadados dos componentes publicados remotamente, que podem variar de mediador para mediador, de acordo com o dicionário de termos do domínio de aplicação. A figura 5.7 apresenta a definição das classes (segundo padrão ODMG ODL) que constituem o modelo base de esquema para o armazenamento dos metadados referentes a um mediador. As classes *Datum*, *SourceCode* e *Service* são responsáveis por armazenar os metadados referentes aos componentes publicados remotamente. Os atributos de cada uma destas classes são mapeados diretamente dos DTDs XML (apêndice A) sugeridos nesta dissertação para documentar cada uma das categorias de componentes formuladas. No entanto, é importante ressaltar que estes atributos não são fixos, podendo variar de mediador para mediador, de acordo com cada DTD XML formulado pelo gerente de domínio para documentar os componentes de um dado domínio de aplicação. Já as classes *Translator* e *Mediator* são utilizadas para cadastrar, respectivamente, as informações dos tradutores e mediadores que constituem a camada de integração. A classe *Translator* é utilizada no esquema de todos os mediadores, enquanto a classe *Mediator* é utilizada apenas no esquema do *Service Manager*.

Uma vez criada a base de dados do mediador no servidor GOA, o mediador é capaz de solicitar, através da *API Cliente GOA*, os serviços de importação e pesquisa de informações de componentes publicados remotamente. Para que os metadados importados pelo mediador no padrão XML pudessem ser armazenados pelo servidor GOA no formato de objetos, foi implementado, nesta API, a classe *GoaXml* (figura 5.4), que oferece serviços de mapeamento de documentos XML para objetos e vice-versa, como descrito em (MATTOSO et al., 2002). Assim, apesar dos metadados serem armazenados internamente no formato orientado a objetos, o que garante um melhor desempenho no processamento de consultas, a apresentação das informações dos componentes publicados para o usuário final é feita sempre em XML, o que garante uma melhor visualização e entendimento dos componentes (GUERRIERI, 1998) (ALONSO e FRAKES, 2000).

```
interface Component
extend(Components)
{
  attribute String Domain;
  attribute String Name;
  attribute String Date;
  attribute String Version;
  attribute String Author;
  attribute String Description;
  attribute String Url;
  attribute String Contact;
};

interface Datum: Component
extend(Data)
{
  attribute String Type;
  attribute String FileType;
  attribute String AssociatedApplications;
};

interface SourceCode: Component
extend(SourceCodes)
{
  attribute String Language;
  attribute String Interface;
};

interface Service: Component
extend(Services)
{
  attribute String OperationSystem;
  attribute String CommandLine;
  attribute String Interface;
};

interface Translator
extend(Translators)
{
  attribute String Name;
};

interface SubMediator
extend(SubMediators)
{
  attribute String Name;
};
```

Figura 5.7 – Esquema padrão GOA para o armazenamento dos metadados de um mediador.

5.3 – Ferramentas de Gerência do Sistema

A fim de administrar cada um dos módulos (mediadores e tradutores) da camada de integração *ComPublish*, foram construídas duas ferramentas gráficas para gerência do sistema: o *Mediator Manager* e *ComPublish Manager*.

O *Mediator Manager* (figura 5.8) é uma ferramenta construída para gerenciar as informações manipuladas pelos mediadores. Uma vez conectada a um mediador qualquer da camada de integração, esta ferramenta oferece as seguintes funcionalidades: (1) cadastro dos tradutores acessados pelo mediador; (2) cadastro dos sub-mediadores associados ao mediador; (3) consultas e importações dos metadados de componentes publicados em um site remoto, ao qual um tradutor ligado ao mediador está conectado; e (4) consultas ao esquema do mediador e às informações de componentes já importadas.

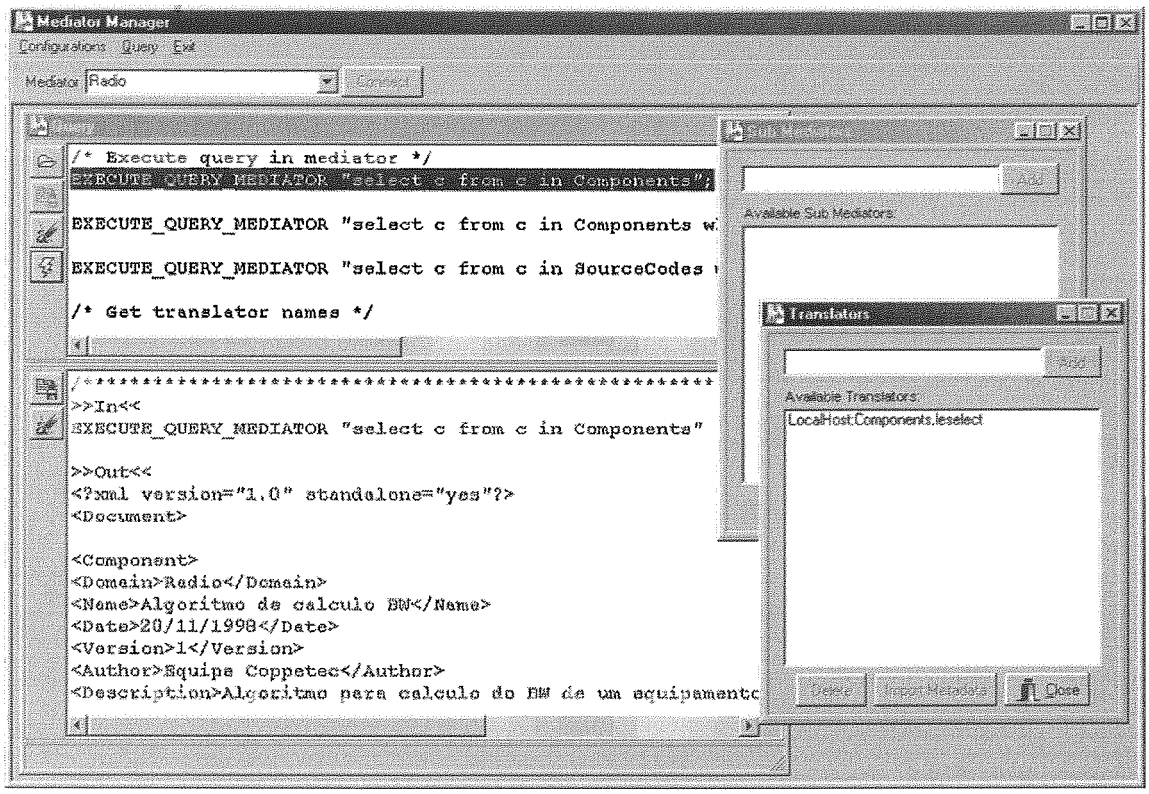


Figura 5.8 – *Mediator Manager.*

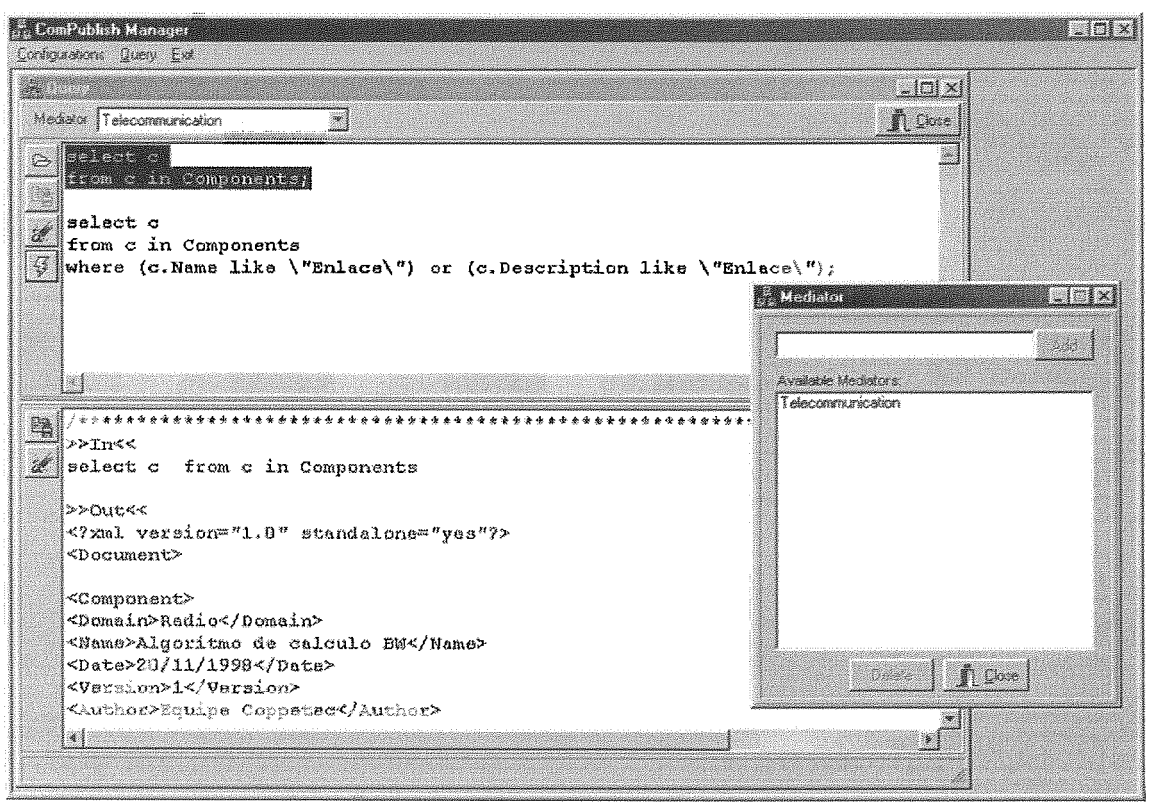


Figura 5.9 – *ComPublish Manager.*

O *ComPublish Manager* (figura 5.9), por sua vez, é uma ferramenta construída para gerenciar as informações manipuladas pelo *Service Manager*. Esta ferramenta oferece as funcionalidades de cadastros de mediadores e de consultas aos metadados de componentes armazenados em um mediador qualquer.

5.4 – Exemplo de Utilização

Para mostrar a utilidade e as facilidades do sistema *ComPublish* no processo de publicação e busca de componentes de software na Internet, é apresentado nesta seção um exemplo que ilustra, passo a passo, a utilização das funcionalidades deste sistema. Para isso, foi construída uma base de testes com componentes obtidos de um projeto real desenvolvido pela COPPE para uma empresa de telecomunicações, nos domínios Rádio e Satélite. Foram selecionadas para publicação algumas amostras de documentos textos, diagramas e códigos fonte, sem revelar, no entanto, o conteúdo de cada uma delas, devido ao compromisso de sigilo.

O cenário montado (figura 5.10) foi composto inicialmente por duas máquinas ligadas em rede, que simularam duas máquinas quaisquer ligadas a Internet em qualquer parte do mundo. Nestas duas máquinas, foram publicados alguns componentes dos dois domínios em questão. Para isso, foram executados os seguintes passos (papel a ser desempenhado pelo publicador):

1. Primeiramente, foi instalado, em ambas as máquinas, os módulos publicadores do sistema, compostos pelo servidor *Le Select* e mais o tradutor de serviços do *Le Select*.

2. O passo seguinte foi construir um documento, segundo padrão XML, contendo os metadados referentes a cada um dos componentes que foram publicados. Para isso, utilizou-se como base os modelos de DTDs XML descritos no apêndice A, de acordo com a categoria do componente (dados, serviços ou código fonte). Embora o sistema não ofereça nenhuma ferramenta específica para a construção deste documento, muitos editores XML podem ser encontrados na Web (como, por exemplo, em www.xml.com) para ajudar o publicador a construir tal documentação, dado o seu DTD.

3. Por fim, foi acrescentado ao diretório de trabalho do *Le Select* o arquivo *components.wd* (figura 5.11), que é um arquivo de configuração interno do sistema, também escrito em XML, onde foram definidos o tipo de tradutor utilizado para acessar os componentes (atualmente, o *Le Select* disponibiliza, já prontos, tradutores para banco de dados JDBC e para arquivos textos (ASCII) estruturados na forma tabular) e o arquivo de metadados em XML contendo a descrição de cada um dos componentes publicados. Neste exemplo, cada componente foi tratado como uma espécie de "caixa preta", ou seja, um arquivo digital armazenado em disco no qual não foi necessário conhecer sua estrutura interna de armazenamento. Assim, para publicá-los, foi definido apenas uma lista de componentes em um arquivo texto (*component.txt*) no formato tabular (com as colunas domínio, nome e endereço para download) (figura 5.12) e configurado o tradutor de arquivos texto para exibição desta lista via *Le Select*.

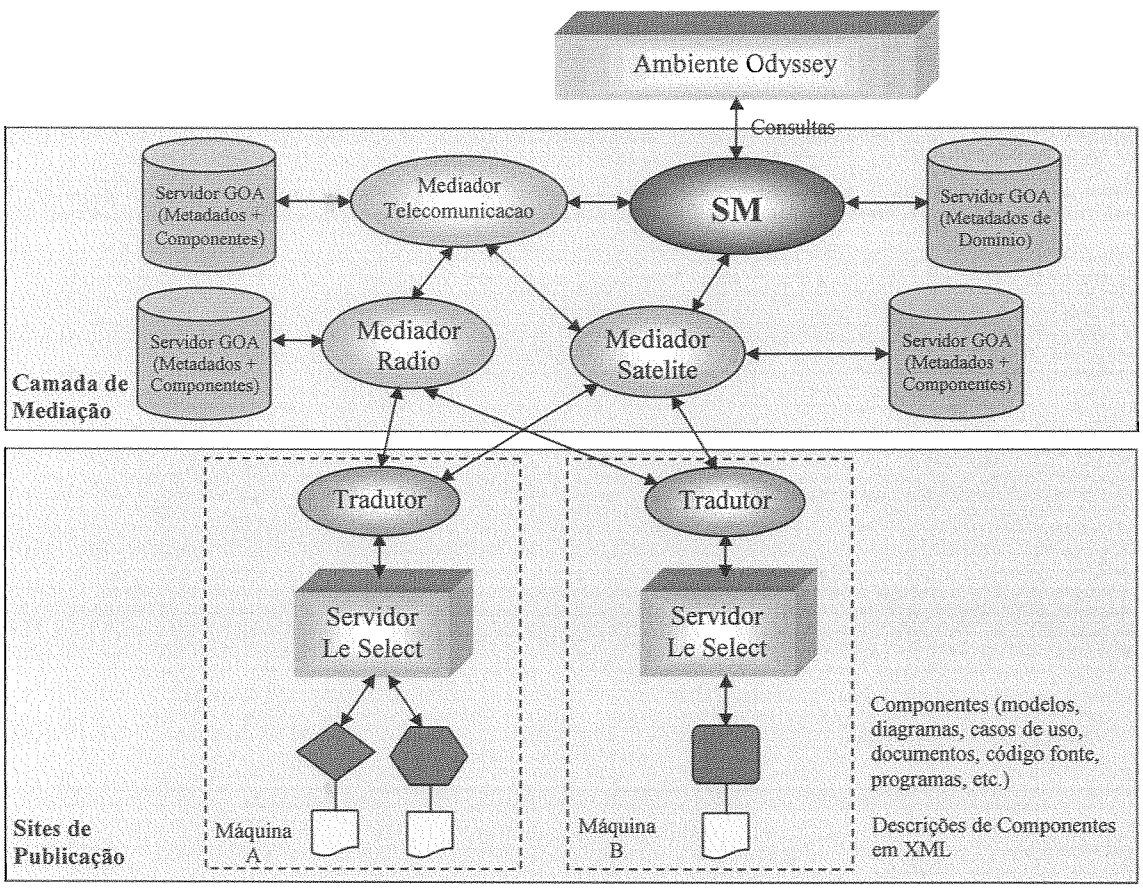


Figura 5.10 – Estudo de Caso no domínio Telecomunicações.

```
<Wrapper WrapperClass="LeSelect.Wrappers.Text.TableWrapperFactory">
  <Parameters>
    <Table name="Component" file="../DataSources/Components/Component.txt">
      <Column name="Domain" type="string" size="30" />
      <Column name="Name" type="string" size="50" />
      <Column name="Download" type="string" size="100" />
    </Table>
  </Parameters>
  <Documents>
    <WrapperDocument>
      <attribute name="TableList" value="Component" />
    </WrapperDocument>
    <TableDocument tableName="Component"
       XMLFileName="../DataSources/Components/Component.xml" />
  </Documents>
</Wrapper>
```

Figura 5.11 – Arquivo *components.wd.*

```
Radio     Algoritmo de calculo BW         ftp://LocalHost:3021/Components/Radio/Fontes/BW.cpp
Radio     Algoritmo de calculo IRF        ftp://LocalHost:3021/Components/Radio/Fontes/IRF.pas
Radio     Diagrama de Classes de Antenas  ftp://LocalHost:3021/Components/Radio/Docs/Antena.mdl
Radio     Relacao de Equipamentos         ftp://LocalHost:3021/Components/Radio/Docs/Eqps.doc
...
Satelite  Diagrama de Classes de Antena   ftp://LocalHost:3021/Components/Satelite/Docs/Antena.mdl
Satelite  Diagrama de Classes de Enlace   ftp://LocalHost:3021/Components/Satelite/Docs/Enlace.mdl
...
```

Figura 5.12 – Arquivo *component.txt.*

Uma vez concluída a etapa de publicação e ativando o servidor *Le Select*, já é possível visualizar na Web os componentes publicados em cada máquina, como explicado na seção 4.3.2. No entanto, falta ainda prover a integração das informações de ambos repositórios (papel a ser desempenhado pelo gerente de domínio). Para isso, foi instanciado, em uma terceira máquina, os mediadores Rádio e Satélite, responsáveis por integrar as informações dos componentes pertencentes aos domínios Rádio e Satélite, respectivamente, armazenados nas outras duas máquinas de publicação. Um terceiro mediador, denominado Telecomunicações, foi instanciado e interligado aos outros dois, de modo a prover um acesso mais genérico a informações de todos os componentes do domínio Telecomunicações. Para cada mediador, foram criadas as respectivas bases GOA para armazenamento dos metadados, de acordo com o esquema de classes apresentado na figura 5.7. Por fim, os mediadores Rádio e Satélite foram interligados

aos tradutores *Le Select* ativados junto com os servidores *Le Select* nas máquinas de publicação e foi executada a importação dos metadados dos componentes publicados remotamente. Para isso, foi utilizado o aplicativo *Mediator Manager* (figura 5.8), que é capaz de se conectar a um mediador, realizar os cadastros de seus tradutores e disparar os serviços de importação. Feito isto, o último passo de todo o processo foi instanciar o *Service Manager* e cadastrar os mediadores ativados, através do aplicativo *ComPublish Manager* (figura 5.9).

Uma vez executadas as etapas de publicação e integração, o sistema *ComPublish* está pronto para receber consultas. Para simular um processo de pesquisa, foi construído, dentro do ambiente de reutilização *Odyssey* (seção 4.4), um exemplo prático para a busca de componentes no domínio de Telecomunicações. O *Odyssey* fornece uma interface gráfica baseada em agentes inteligentes, denominada *CompAgent* (COSTA, 2002), que ajuda o usuário a desenvolver suas consultas de forma transparente, baseada no uso de palavras chave fornecidas. Ao solicitar a busca por componentes na Internet, agentes internos disparam consultas tanto em *sites* de busca na Web, quanto para o sistema *ComPublish* (SOUZA et al., 2001).

Para executar uma busca geral por componentes, o usuário abre a janela de Busca na Web (figura 5.13) e insere a palavra chave desejada:

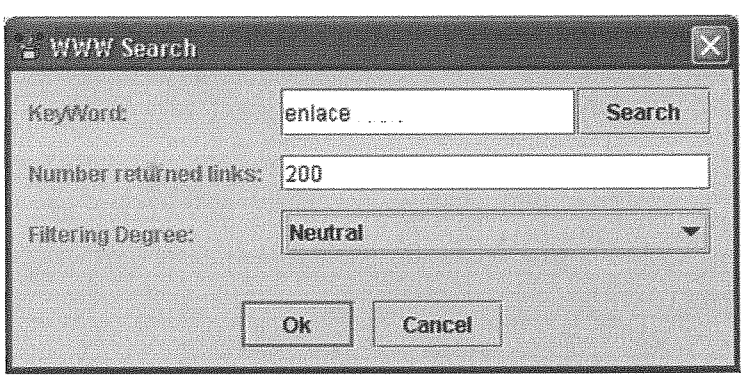


Figura 5.13 – Janela *Odyssey* para pesquisa na Web.

Embora seja uma pesquisa baseada em palavras chave, o *Odyssey CompAgent* constrói de forma transparente uma consulta baseada na linguagem OQL (figura 5.14) e a envia para o *ComPublish*, através do módulo *Service Manager*. O *Service Manager* repassa a consulta para o mediador Telecomunicação, que por sua vez a distribui aos sub mediadores Rádio e Satélite. Em cada um dos mediadores, é feito o processamento

da consulta sobre a base de metadados do servidor GOA. Os resultados retornados de ambos mediadores, segundo o padrão XML, são juntados pelo mediador Telecomunicação e retornados ao cliente, como visualizado na figura 5.15.

```
select c
from c in Components
where (c.Name like \"enlace\") or (c.Domain like \"enlace\") or
  (c.Date like \"enlace\") or (c.Version like \"enlace\") or
  (c.Author like \"enlace\") or (c.Description like \"enlace\") or
  (c.Url like \"enlace\") or (c.Contact like \"enlace\") or
  (c.Type like \"enlace\") or (c.FileType like \"enlace\") or
  (c.AssociatedApplications like \"enlace\") or (c.Language like \"enlace\") or
  (c.OperationSystem like \"enlace\")
```

Figura 5.14 – Exemplo de consulta OQL enviada pelo *Odyssey CompAgent* ao *ComPublish.*

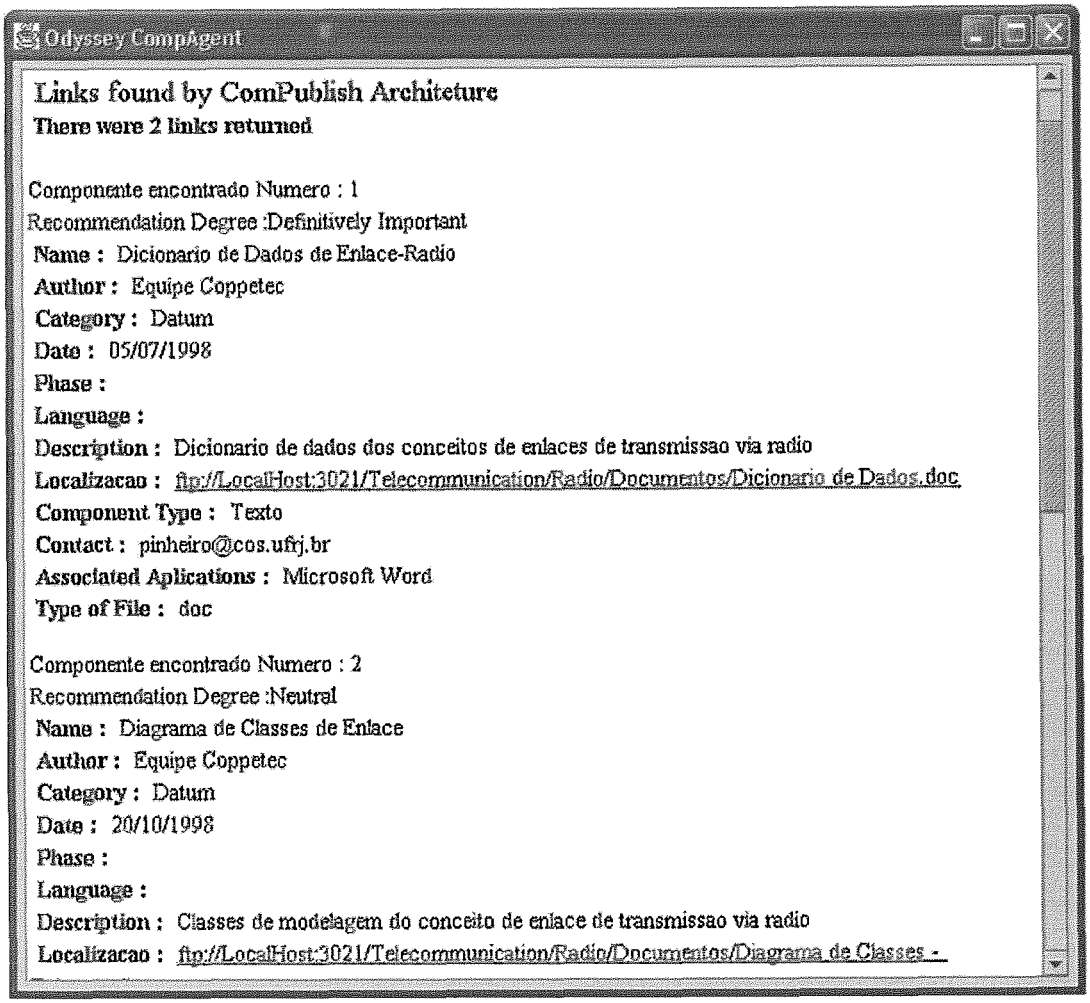
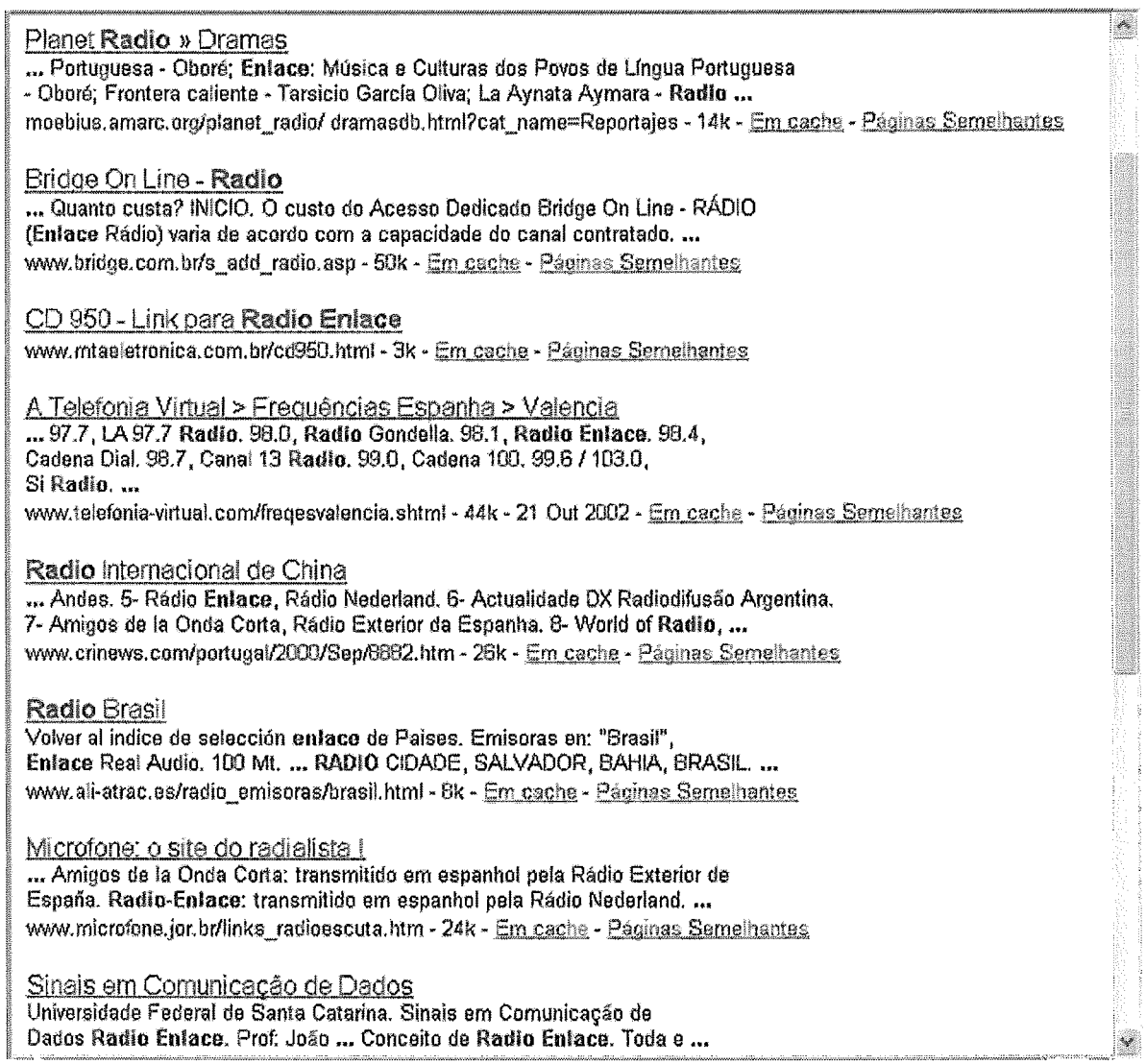


Figura 5.15 – Resultado retornado pelo *ComPublish* ao *Odyssey CompAgent.*

Como pode ser visto na figura 5.15, os resultados retornados são bastante focados no domínio de interesse do desenvolvedor (Telecomunicações) e na palavra chave sugerida. Se fosse tentado este tipo de consulta em um sistema de pesquisa da Web, como por exemplo o *Google*, usando as palavras chave *radio* e *enlace*, os resultados retornados seriam muito mais volumosos e imprecisos (figura 5.16), o que iria exigir do usuário um grande esforço manual de filtragem destes resultados, através do acesso e verificação do conteúdo de cada um dos links retornados. Este é um trabalho que, na maioria das vezes, despende muito tempo e paciência do desenvolvedor, podendo gerar, consequentemente, um desincentivo à reutilização.



Planet **Radio** » Dramas
... Portuguesa - Oboré; **Enlace**: Música e Culturas dos Povos de Língua Portuguesa - Oboré; Frontera caliente - Tarsicio García Oliva; La Aynata Aymara - **Radio** ...
moebius.amarc.org/planet_radio/ dramasdb.html?cat_name=Reportajes - 14k - Em cache - Páginas Semelhantes

Bridge On Line - **Radio**
... Quanto custa? INICIO. O custo do Acesso Dedicado Bridge On Line - RÁDIO (**Enlace** Rádio) varia de acordo com a capacidade do canal contratado. ...
www.bridge.com.br/s_add_radio.asp - 50k - Em cache - Páginas Semelhantes

CD 950 - Link para **Radio Enlace**
www.rntaeletronica.com.br/cd950.html - 3k - Em cache - Páginas Semelhantes

A Telefonia Virtual > Frequências Espanha > Valencia
... 97.7, LA 97.7 **Radio**. 98.0, **Radio** Gondella. 98.1, **Radio Enlace**. 98.4, Cadena Dial. 98.7, Canal 13 **Radio**. 99.0, Cadena 100. 99.6 / 103.0, Si **Radio**. ...
www.telefonia-virtual.com/freqesvalencia.shtml - 44k - 21 Out 2002 - Em cache - Páginas Semelhantes

Radio Internacional de China
... Andes. 5- Rádio **Enlace**, Rádio Nederland. 6- Actualidade DX Radiodifusão Argentina. 7- Amigos de la Onda Corta, Rádio Exterior da Espanha. 8- World of **Radio**, ...
www.crinews.com/portugal/2000/Sep/6882.htm - 26k - Em cache - Páginas Semelhantes

Radio Brasil
Volver al índice de selección **enlace** de Paises. Emisoras en: "Brasil", **Enlace** Real Audio. 100 Mt. ... **RADIO** CIDADE, SALVADOR, BAHIA, BRASIL. ...
www.aii-atrac.es/radio_emisoras/brasil.html - 8k - Em cache - Páginas Semelhantes

Microfone: o site do radialista !
... Amigos de la Onda Corta: transmitido em espanhol pela Rádio Exterior de España. **Radio-Enlace**: transmitido em espanhol pela Rádio Nederland. ...
www.microfone.jor.br/links_radioescuta.htm - 24k - Em cache - Páginas Semelhantes

Sinais em Comunicação de Dados
Universidade Federal de Santa Catarina. Sinais em Comunicação de Dados **Radio Enlace**. Prof. João ... Conceito de **Radio Enlace**. Toda e ...

Figura 5.16 – Resultado retornado pelo *Google*.

5.5 – Considerações Finais

Neste capítulo, foi apresentado o protótipo de implementação do *ComPublish*, um sistema baseado no uso de tecnologias padronizadas (XML, CORBA, etc) e nas idéias de outras arquiteturas e sistemas em desenvolvimento (HIMPAR, GOA, *Le Select*). Por ser um sistema voltado para Internet, o *ComPublish* é desenvolvido com base em uma arquitetura de camadas modular e flexível. Isto permite que tanto novos repositórios de componentes quanto novos domínios de aplicação possam ser agregados facilmente a arquitetura, o que garante uma fácil extensão da biblioteca de componentes distribuídos.

Ao final, foi descrito um exemplo de utilização do sistema, onde foram mostradas a utilidade e as facilidades providas para um processo de publicação e busca de componentes de software na Internet, e mais qualidade do processo de pesquisa, quando comparado aos tradicionais mecanismos de busca na Web. Com isso, queremos reforçar que a adoção de idéias como (1) a organização de informações por domínio de aplicação, (2) a utilização de metadados para descrição de componentes e para a melhoria dos mecanismos de busca, e (3) a utilização de uma linguagem de consulta mais expressiva, dentre outros aspectos, podem ajudar bastante a melhorar a qualidade dos mecanismos de busca e recuperação de componentes de software na Internet, como foi demonstrado neste exemplo de utilização do *ComPublish*.

6 Conclusões

6.1 – Considerações Finais

Conforme discutido nesta dissertação, a **reutilização** de software é uma atividade que vem crescendo cada vez mais dentro das empresas desenvolvedoras de software, como forma de aumentar e melhorar qualitativamente a produção, além de reduzir os custos e o tempo de desenvolvimento. Reutilizar significa reaproveitar componentes já prontos, produzidos por qualquer desenvolvedor dentro da empresa ou até mesmo por desenvolvedores de outras organizações. No entanto, para construir um processo efetivo de reutilização de componentes já prontos, é preciso oferecer a todos os desenvolvedores uma abordagem integrada para a publicação, busca, compreensão e recuperação de componentes.

Baseado nestas necessidades, nossas pesquisas foram primeiramente direcionadas para um estudo detalhado das principais tecnologias na área de Banco de Dados voltadas para a integração de informações heterogêneas e distribuídas. Numa segunda etapa, foi feito um estudo detalhado, na área de Engenharia de Software, das necessidades apontadas por pesquisadores para se prover um melhor acesso a componentes de software distribuídos na Internet. Casando as conclusões obtidas em ambos estudos, foi formulada, então, a proposta do *ComPublish*, um sistema que visa auxiliar desenvolvedores de software a publicar, buscar e recuperar variados artefatos de software na Internet, tal como modelos, diagramas, código fonte, documentos, programas ou qualquer outro tipo de artefato utilizado ou produzido nas diferentes etapas do processo de desenvolvimento de um software. As principais idéias arquiteturais apresentadas no *ComPublish*, representadas pela utilização de uma abordagem baseada em mediadores e tradutores para organizar componentes segundo o conceito de domínios de aplicação, seguem as idéias descritas pela *Odyssey-Search*

(BRAGA, 2000b). No entanto, o *ComPublish* estende este trabalho, apresentando (1) uma proposta mais abrangente para a publicação de componentes na Internet, através da utilização de metadados em XML; (2) novos mecanismos de integração e consulta de informações de componentes, através da utilização de abordagens que mesclam os conceitos das tecnologias XML e de Orientação a Objetos; e (3) o uso do *Le Select*, que provê facilidades para publicação de dados na Internet, além da integração entre dados e programas.

6.2 – Contribuições

Como principais contribuições desta dissertação, podemos citar:

- A proposta de uma abordagem de recuperação de componentes de software voltada para o domínio de aplicação do engenheiro de domínio, o que facilita e aumenta a qualidade do processo de busca por componentes (KRUEGER, 1992) (MILLI et al., 1995).
- O projeto e implementação do *ComPublish*, um sistema modular, flexível e distribuído, baseado em tecnologias padronizadas e abertas (XML, CORBA, etc), e que pode ser tanto utilizado na Web quanto integrado a um ambiente de reutilização qualquer, como feito para o ambiente *Odyssey* (SOUZA et al., 2001) (WERNER et al., 2002).
- A utilização com sucesso do sistema *Le Select* (XHUMARI e MOKRANE, 1999) como uma ferramenta facilitadora para publicação de dados.
- A evolução do sistema GOA, com alguns aspectos destacados em (MATTOSO et al., 2000 e 2002), através de uma reimplementação mais modularizada e correções de *bugs* em seu módulo servidor, o reprojeto do mapeamento de objetos entre o disco e a memória, a criação de serviços de remoção e de "*garbage collection*" de objetos em disco, além da criação de APIs Clientes, que englobam novas funcionalidades o para armazenamento de documentos XML.

Em relação às publicações, este trabalho foi aceito para apresentação em dois congressos nacionais (SOUZA et al., 2000 e 2002) e mais em um congresso internacional (SOUZA et al., 2001). Além disso, o *ComPublish* foi destacado ainda como parte de publicações recentes, ligadas às ferramentas *Odyssey* (WERNER et al., 2002) e GOA (MATTOSO et al., 2002), no Simpósio Brasileiro de Engenharia de Software, na sessão de ferramentas.

6.3 – Limitações e Trabalhos Futuros

A fim de dar continuidade a este trabalho, são sugeridos alguns pontos de melhorias e estudos futuros:

- A proposta em utilizar o *Le Select* como módulo publicador de informações não se limita às facilidades que este sistema oferece para publicar dados. Acreditamos que outros recursos importantes, não explorados no escopo desta dissertação, como a integração de dados e programas, podem contribuir bastante para a interação de componentes de software distribuídos. Além disso, deve ser pesquisada a viabilidade de outras tecnologias lançadas recentemente com propósito semelhante, como, por exemplo, a tecnologia de *Web Services* (W3C, 2002b).
- Quanto à implementação, apontamos como limitação desta primeira versão do *ComPublish* as dependências da plataforma operacional (*Windows*) e de desenvolvimento (*Borland*), por conta da implementação em C++ da camada de integração. Assim, sugerimos a reimplementação de mediadores e tradutores em linguagem *Java*, com o objetivo de tornar estes módulos independentes de máquina ou sistema operacional, obtendo desta forma a flexibilização plena de todos os módulos do sistema, não somente do *Le Select*. Além disso, isso viabilizaria o uso de novos recursos de mapeamento XML-Objeto, segundo o padrão DOM, implementados recentemente em *Java* na *API Cliente GOA* (MATTOSO et al., 2002), permitindo assim uma solução genérica para o armazenamento de qualquer documento XML na camada de integração.

- Em se tratando de uma ferramenta voltada para Internet, onde XML é o padrão escolhido para interface de comunicação com o usuário, a utilização de uma linguagem de consulta também baseada em XML pode facilitar o usuário a desenvolver suas consultas para pesquisas no sistema. O *XVerter* (VIEIRA et al., 2002), por exemplo, é uma proposta que pode facilitar esta evolução.

- Outro ponto importante a ser mais aprofundado é a questão dos metadados ligados à documentação de componentes. Uma pesquisa mais detalhada sobre o assunto pode ajudar na evolução das primeiras versões de DTDs XML sugeridos neste trabalho.

- Em relação à avaliação, é preciso fazer uma análise detalhada com relação ao desempenho do processamento de consultas em uma arquitetura baseada em camadas, como é a arquitetura de mediadores. Alguns especialistas criticam que este tipo de estrutura pode degradar o tempo de resposta das pesquisas, comprometendo o desempenho da ferramenta como um todo.

Referências Bibliográficas

ALONSO, O.; FRAKES, W. B.; "Visualization of Reusable Software Assets"; 6th International Conference on Software Reuse (ICSR-6), pg. 251-265; Viena, Áustria, 2000.

ARANGO, G.; "Domain Engineering for Software Reuse"; Technical Report UCI-ICS, 88-27; Universidade da Califórnia, Estados Unidos; 1998.

ATKINSON, C. et al.; "Component-Based Software Engineering: The KobrA Approach"; International Workshop on Component-Based Software Engineering; Limerick, Irlanda; Junho de 2000.

BARJA, M. L.; BRATVOLD T.; MYLLYMAKI J.; SONNENBERGER G.; "Informia: A Mediator for Integrated Access to Heterogeneous Information Sources"; Proceedings of the Conference on Information and Knowledge Management CIKM'98, online no site: http://www.informia.com, acessado em Março de 2001; 1998.

BARROS, M.; "Reutilização de Conhecimento no Gerenciamento de Projetos Baseado em Cenário"; Engenharia de Domínio e Desenvolvimento Baseado em Componentes, Relatório Técnico do Projeto Odyssey 10/2000, COPPE/UFRJ; Rio de Janeiro, Brasil; 2000.

BARU, C.; PAPAKONSTANTINOU Y. et al.; "XML-Based Information Mediation with MIX"; ACM Conference on Management of Data (SIGMOD'99); Philadelphia, Estados Unidos; 1999.

BECKER, K.; PEREIRA, W. A. L.; "Data Warehouse: Arquitetura Funcional e Ferramentas de Apoio ao Projeto e Implementação"; XIII Simpósio Brasileiro de Banco de Dados; Florianópolis, Santa Catarina, Brasil; Outubro de 1999.

BERNSTEIN, P.; MOLINA, H. G.; LESK, M. et al.; "The Asilomar Report on Database Research"; ACM SIGMOD-Record, vol. 27, n. 4; Dezembro de 1998.

BORLAND; online no site: http://www.borland.com, acessado em Novembro de 2002.

BRAGA, R. M. M.; WERNER, C. M. L.; MATTOSO, M. L. Q.; "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, pp. 542-546; Santa Barbara, Califórnia; Agosto de 1999a.

BRAGA, R. M. M. et al.; "Odyssey: A Reuse Environment based on Domain Models"; Proceedings of IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'99), pp. 49-57; Texas, Estados Unidos; 1999b.

BRAGA, R. M. M.; WERNER, C. M. L.; MATTOSO, M. L. Q.; "A Multi-Agent System for Domain Information Discovery and Filtering"; XIV Simpósio Brasileiro de Engenharia de Software; João Pessoa, Paraíba, Brasil; Outubro de 2000a.

BRAGA, R. M. M.; "Busca e Recuperação de Componentes em Ambientes de Reutilização de Software"; Tese de Doutorado, Coordenação dos Programas de Pesquisa e Pós-Graduação em Engenharia - COPPE/UFRJ; Rio de Janeiro, RJ, Brasil; Dezembro de 2000b.

BRÜGGER, T. S.; "Serviços de Gerência de Metadados para Mediadores: Uma Implementação sobre o Servidor de Objetos GOA++"; Dissertação de Mestrado, Coordenação dos Programas de Pesquisa e Pós-Graduação em Engenharia - COPPE/UFRJ; Rio de Janeiro, RJ, Brasil; 2000.

CAMPOS, M. L. M.; BORGES, V. J. A. S.; "Diretrizes para a Modelagem Incremental de Data Marts"; XVII Simpósio Brasileiro de Banco de Dados, pp. 110-120; Gramado, Rio Grande do Sul, Brasil; Outubro de 2002.

CATTEL, R. G. G.; "The Object Database Standard: ODMG-93 - Release 1.1"; Morgan Kaufmann Publishers Inc.; San Francisco, California; 1993.

CAVALCANTI, M. C.; MATTOSO, M. L. Q.; CAMPOS, M. L. M.; LLIRBAT, F.; SIMON, E.; "Sharing Scientific Models in Environmental Applications"; Symposium on Applied Computing (SAC), pp. 453-457; Madrid, Espanha; Março de 2002a.

CAVALCANTI, M. C.; MATTOSO, M. L. Q.; CAMPOS, M. L. M.; SIMON, E.; LLIRBAT, F.; "An Architecture for Managing Distributed Scientific Resources"; Proc. of the 14th International Conference on Scientific and Statistical Database Management (SSDBM); Edinburgh, Scotland, UK; Julho de 2002b.

COHEN, S.; "Object-Oriented Technology and Domain Analysis"; 5th International Conference on Software Reuse (ICSR-5); Canadá; 1998.

COMPONENTSOURCE; online no site: www.componentsource.com, acessado em Maio de 2002.

CORBA; "CORBA 2.0 Specification"; Object Management Group, online no site: http://www.infosys.tuwien.ac.at/Research/Corba, acessado em Março de 2001; 1997.

COSTA, M. N.; "CompAgent: Uma ferramenta para apoio a recuperação de informações orientadas a domínio na Web"; Dissertação de Mestrado, Coordenação dos Programas de Pesquisa e Pós-Graduação em Engenharia - COPPE/UFRJ; Rio de Janeiro, RJ, Brasil; 2002.

DATAJOINER; "IBM Co. DB2 DataJoiner: Administration Guide and Application Programming. IBM C."; Relatório Técnico IBM; San Jose; 1997.

DISCO; "Distributed Information Search Component"; online no site: http://www.cs.huji.ac.il/~sdbi/1999/rica/sem1024x768/tsld001.htm, acessado em Março de 2001.

DITTRICH, K.; DOMENIG, R.; "Towards Exploitation of the Data Universe"; In: III International Conference on Business Information System; Abril de 1999.

DREILINGER D.; HOWE A. E.; "Experiences with Selecting Search Engines Using Metasearch"; ACM Transactions on Information Systems, vol. 15, n. 3, pp. 195-222; Julho de 1997.

EZRAN, M. et al.; "Practical Software Reuse: The Essential Guide"; Addison-Wesley, ed. 1, pp. 512; 1999.

GUERRIERI, E.; "Software Document Reuse with XML"; Proceedings of the Fifth International Conference on Software Reuse; Vitória, Canada; 1998.

HALL, P.; "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 de 2000.

HUGHES, K.; "Oracle Transport Gateway – Installation and User's Guide for IBM DRDA fro RS/6000"; Oracle Co.; 1996.

JACOBSON, I.; GRISS, M.; JONSSON, P.; "Software Reuse: Architecture, Process and Organization for Business Success"; Addison Wesley Longman; Maio de 1997.

KIM, W.; SEO, J.; "Classifying Schematic and Data Heterogeneity in Multidatabase Systems"; IEEE Computer, vol. 24, n. 12, pp. 12-18; Dezembro de 1991.

KRUEGER, C.; "Software Reuse"; ACM Computing Surveys, vol. 24, n. 2, pp. 131 183; 1992.

LIM, W.; "Managing Software Reuse"; Prentice Hall; 1998.

MANOLESCU, I.; FLORESCU, D.; KOSSMANN, D.; "Agora: Living with XML and Relational"; 26th VLDB Conference; Fevereiro de 2000.

MANOLESCU, I.; FLORESCU, D.; KOSSMANN, D.; "Pushing XML Queries Inside Relational Databases"; INRIA Technical Report, INRIA, No. 4112; Janeiro de 2001a.

MANOLESCU, I.; FLORESCU, D.; KOSSMANN, D.; "Answering XML Queries over Heterogenous Datasources"; Proc. of the Int'l. Conf. on Very Large Databases (VLDB); Roma, Itália; 2001b.

MATTOSO, M. L. Q. et al.; "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 de 1994.

MATTOSO, M. L. Q.; WERNER, C. M. L.; SOUZA, R. P. et al.; "Persistência de Componentes num Ambiente de Reuso"; XIV Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas; João Pessoa, Paraíba, Brasil; Outubro de 2000.

MATTOSO, M. L. Q.; SOUZA, R. P.; WERNER, C. M. L. et al.; "Gerência de Documentos XML no GOA"; XVI Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, pp. 402-407; Gramado, Rio Grande do Sul, Brasil; Outubro de 2002.

MAURO, R.; ZIMBRÃO, G. Z.; BRÜGGER, T. et al.; "GOA++: Tecnologia, implementação e extensões aos serviços de gerência de objetos"; XII Simpósio Brasileiro de Banco de Dados, pp. 272-286; Fortaleza, Brasil; Outubro de 1997.

MDC – "Meta Data Coalition"; online no site: http://www.mdcinfo.com, acessado em Fevereiro de 2002.

MELO, W.; SOUZA, C. A. A.; HOLANDA, C. B. S.; "ProReuso: Um Repositório de Componentes para Web Dirigido por um Processo de Reuso"; Simpósio Brasileiro de Banco de Dados; Rio de Janeiro, Brasil; Outubro de 2001.

MILLI, A. et al.; "Reusing Software: Issues and Research Directions"; IEEE Transactions on Software Engineering, vol. 21-6, pp. 528-562; 1995.

MIX – "Mediation of Information using XML"; online no site: http://www.npaci.edu/DICE/MIX/, acessado em Março de 2001.

MOURA, A. M. C.; CAMPOS, M. L. M.; BARRETO, C. M.; "A Survey on Metadata for Describing and Retrieving Internet Resources"; World Wide Web 1, vol. 4, pp. 221-240; Fevereiro de 1999.

MURTA, L.; "FRAMEDOC: Um FrameWork para a Documentação de Componentes Reutilizáveis"; Projeto Final de Curso, DCC/IM/UFRJ; Novembro de 1999.

MURTA, L.; "Uma Máquina de Processos de Desenvolvimento de Software Baseada em Agentes Inteligentes"; XIV Simpósio Brasileiro de Engenharia de Software, Workshop de Teses; João Pessoa, Paraíba, Brasil; Outubro de 2000.

OIM – "Open Information Model"; online no site: http://www.mdcinfo.com/OIM/index.html, acessado em Fevereiro de 2002.

OLIVEIRA, A. M.; "GOS: Serviços de Ontologia na Integração de Bases de Dados"; Dissertação de Mestrado, Coordenação dos Programas de Pesquisa e Pós-Graduação em Engenharia - COPPE/UFRJ; Rio de Janeiro, RJ, Brasil; 2002.

OMG – "Object Management Group"; online no site: http://www.omg.org, acessado em Março de 2002.

ÖSZU, M. T.; VALDURIEZ, P.; "Principles of Distributed Database Systems"; Prentice Hall International, 1ª edition; Paris, França; 1990.

PEREIRA, R. C. G.; "Uma Abordagem para Gerenciamento de Consistência em um Ambiente de Banco de Dados Heterogêneos"; Dissertação de Mestrado, UFPE; Recife, Pernambuco, Brasil; 1999.

PRIETO-DIAZ, R.; "A Software Classification Scheme for Reusability"; IEEE Software, vol.4-1, pp. 6-16; Janeiro de 1987.

PIRES, P. F.; MATTOSO, M. L. Q.; "Aspectos de Interoperabilidade da Arquitetura Heterogênea HIMPAR"; XI Simpósio Brasileiro de Banco de Dados, pp. 43-57; São Carlos, São Paulo, Brasil; Outubro de 1996.

PIRES, P. F.; "HIMPAR - Uma Arquitetura para Interoperabilidade de Objetos Distribuídos"; Dissertação de Mestrado, Coordenação dos Programas de Pesquisa e Pós-Graduação em Engenharia - COPPE/UFRJ; Rio de Janeiro, RJ, Brasil; 1997.

RIG; "Reusable Library Interoperability Group"; online no site: http://www.asset.com/rig/, acessado em Maio de 2002; 1996.

SEACORD, R.; HISSAN, S.; WALLNAU, K.; "Agora: A Search Engine for Software Components"; SEI Technical Report CMU/SEI-98-TR-011; 1998.

SEACORD, R.; "Software Engineering Component Repositories"; International Workshop on Component-Based Software Engineering; Los Angeles, Estados Unidos; Maio de 1999.

SAMETINGER, J.; "Software Engineering with Reusable Components"; Springer; 1997.

SELBERG, E.; ETZIONI, O.; "The MetaCrawler Architecture for Resource Aggregation on the Web"; IEEE Expert, pp. 11-14; Janeiro de 1998.

SHETH, A. P.; LARSON, J. A.; "Federated Database Systems for Managing Distributed, Heterogenous and Autonomous Databases"; ACM Computing Surveys, vol. 22, n. 3, pp. 183-236; Setembro de 1990.

SIMON, E.; TOMASIC, A.; "Improving Access to Environment Data using Context Information"; SIGMOD Record; 1997.

SOUZA, R. P.; MATTOSO, M. L. Q.; WERNER, C. M. L.; "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, Paraíba, Brasil; Outubro de 2000.

SOUZA, R. P.; COSTA, M. N.; MATTOSO, M. L. Q.; WERNER, C. M. L.; "Software Components Reuse Through Web Search and Retrieval"; International Workshop on Information Integration on the Web; Itaipava, Rio de Janeiro; Abril de 2001.

SOUZA, R. P.; OLIVEIRA, A. M.; BRAGA, R. M. M.; MATTOSO, M. L. Q.; WERNER, C. M. L.; "Uso de Mediadores e Ontologias para Recuperação de Componentes de Software"; Workshop de Desenvolvimento Baseado em Componentes; Itaipava, Rio de Janeiro; Agosto de 2002.

THOMAS, G. et al.; "Heterogeneous Distributed Database Systems for Production Use"; ACM Computing Surveys, vol. 22, n. 3; Setembro de 1990.

TOMAZIC, A.; RASCHID, L.; VALDURIEZ, P.; "Scaling Access to Heterogenous Data Sources with DISCO"; IEEE Transactions on Knowledge and Data Engineering, vol. 10, n. 5; Setembro de 1998.

TRANNIN, M.; SANTOS, F.; WERNER, C. M. L.; BORGES, M.; "Uma Infra-estrutura de apoio à Aquisição Cooperativa de Conhecimento em Engenharia de Domínio"; XIII Simpósio Brasileiro de Engenharia de Software; Florianópolis, Santa Catarina, Brasil; Outubro de 1999.

UCHÔA, E. M. A.; "Heros - Um Sistema de Bancos de Dados Heterogêneos"; Dissertação de Mestrado, Departamento de Informática da PUC; Rio de Janeiro, RJ, Brasil; 1995.

VALDURIEZ, P.; TANAKA, A. et al.; "The Ecobase Environmental Information System: Applications, Architecture and Open Issues"; In Networking and Information Systems Journal (NISJ), vol. 3, n. 5; 2000.

VALDURIEZ, P.; TANAKA, A. et al.; "The Ecobase Project: Database and Web Technologies for Environmental Information Systems"; In ACM SIGMOD-Record; Setembro de 2001.

VIEIRA, H.; HUBERG, G.; MATTOSO, M. L. Q.; "XVerter: Armazenamento e Consulta de Dados XML em SGBDs"; XVII Simpósio Brasileiro de Banco de Dados, pp. 224-238; Gramado, Rio Grande do Sul, Brasil; Outubro de 2002.

WERNER, C. M. L., MATTOSO, M. L. Q., BRAGA, R. M. M. et al.; "Odyssey: Infra-estrutura de Reutilização baseada em Modelos de Domínio"; XIII Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, pp. 17-20; Florianópolis, Santa Catarina, Brasil; Outubro de 1999.

WERNER, C. M. L.; BRAGA, R. M. M.; MATTOSO, M. L. Q. et al.; "Odyssey: Estágio Atual"; XIV Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas; João Pessoa, Paraíba, Brasil; Outubro de 2000.

WERNER, C. M. L. et al.; "OdysseyShare: Um ambiente para o Desenvolvimento Cooperativo de Componentes"; XVII Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, pp. 444-449; Gramado, Rio Grande do Sul, Brasil; Outubro de 2002.

WIEDERHOLD, G.; "Mediators in the Architecture of Future Information Systems"; IEEE Computer Society Press, vol. 25, pp. 38-49; Março de 1992.

WIEDERHOLD, G.; "Mediation to Deal with Heterogenous Data Sources"; Proceeding of the Interop'99, online no site: http://hake.stanford.edu/pub/gio/, acessado em Março de 2001; Zuric; 1999.

W3C – "World Wide Web Consortium"; online no site: http://www.w3c.org, acessado em Maio de 2002; 2002a.

W3C – "Web Services"; online in site: http://www.w3.org/2002/ws/, acessado em Setembro de 2002; 2002b.

XAVIER, J.; "Uma Ferramenta de Apoio à Definição e Instanciação de Arquiteturas Específicas de Domínio"; Dissertação de Mestrado, Coordenação dos Programas de Pesquisa e Pós-Graduação em Engenharia - COPPE/UFRJ; Rio de Janeiro, Brasil; 2000.

XHUMARI, F.; MOKRANE, A.; "Le Select: a Middleware System for Publishing Autonomous and Heterogenous Information Sources"; INRIA, Groupe Caravel; França; 1999.

YATES, R. B.; NETO, B. R.; "Modern Information Retrieval"; Addison Wesley; New York, Estados Unidos; 1999.

Apêndices

Apêndice A – DTDs XML para Documentação de Componentes

```
<!--                                                                -->
<!-- Component Documentation DTD for ComPublish - Version 1.0       -->
<!-- Category: Data (texts, images, diagrams, models)               -->
<!-- Comments: pinheiro@cos.ufrj.br                                 -->
<!--                                                                -->

<!ELEMENT component(domain,
                    name,
                    category,
                    type,
                    date,
                    version,
                    description,
                    associatedApplications,
                    webSite,
                    authors,
                    contacts)>

<!---->
<!--Domain of the component-->
<!---->
   <!ELEMENT domain (#PCDATA)>

<!---->
<!--Name of the component-->
<!---->
   <!ELEMENT name (#PCDATA)>

<!---->
<!--Category of the component. Ex: text, image, diagram, model-->
<!---->
   <!ELEMENT category (#PCDATA)>

<!---->
<!--Type of the component. Ex: txt, doc, pcx, jpeg, mdl-->
<!---->
   <!ELEMENT type (#PCDATA)>

<!---->
<!--Fabrication date of component-->
<!---->
```

```
   <!ELEMENT date (#PCDATA)>

<!---->
<!--Version of the component. Ex: 1, 2.0, 3.1-->
<!---->
   <!ELEMENT version (#PCDATA)>

<!---->
<!--Description of component-->
<!---->
   <!ELEMENT description (#PCDATA)>

<!---->
<!--URL of the component. Ex:
http://www.cos.ufrj.br/ComPublish/Index.html-->
<!---->
   <!ELEMENT webSite (#PCDATA)>

<!---->
<!--Associated applications of component-->
<!---->
   <!ELEMENT associatedApplications (#PCDATA)>

<!---->
<!--Authors of component-->
<!---->
   <!ELEMENT authors (author+)>

<!---->
<!--Author of component-->
<!---->
   <!ELEMENT author (#PCDATA)>

<!---->
<!--Contacts-->
<!---->
   <!ELEMENT contacts (contact+)>

<!---->
<!--Contact-->
<!---->
   <!ELEMENT contact (description,
                      streetAddress,
                      city,
                      state,
                      postalCode,
                      country,
                      phoneNumbers,
                      mail)>

<!---->
<!--Brief explanation of what is available from this contact-->
<!---->
   <!ELEMENT description (#PCDATA)>

<!---->
<!--Line of the street address-->
<!---->
   <!ELEMENT streetAddress (#PCDATA)>
```

```
<!---->
<!--Name of the city-->
<!---->
   <!ELEMENT city (#PCDATA)>

<!---->
<!--Name of the state-->
<!---->
   <!ELEMENT state (#PCDATA)>

<!---->
<!--Number of postal code-->
<!---->
   <!ELEMENT postalCode (#PCDATA)>

<!---->
<!--Name of country-->
<!---->
   <!ELEMENT country (#PCDATA)>

<!---->
<!--List of phone numbers-->
<!---->
   <!ELEMENT phoneNumbers (phoneNumber+)>

<!---->
<!--Phone number-->
<!---->
   <!ELEMENT phoneNumber (#PCDATA)>

<!---->
<!--Mail-->
<!---->
   <!ELEMENT mail (#PCDATA)>
```

```
<!--                                                                    -->
<!-- Component Documentation DTD for ComPublish - Version 1.0          -->
<!-- Category: Services (programs)                                     -->
<!-- Comments: pinheiro@cos.ufrj.br                                    -->
<!--                                                                    -->

<!ELEMENT component (domain,
                     name,
                     date,
                     version,
                     operationSystem,
                     description,
                     commandLine,
                     interface,
                     webSite,
                     authors,
                     contacts)>

<!---->
<!--Domain of the component-->
<!---->
   <!ELEMENT domain (#PCDATA)>

<!---->
<!--Name of the component-->
<!---->
   <!ELEMENT name (#PCDATA)>

<!---->
<!--Fabrication date of component-->
<!---->
   <!ELEMENT date (#PCDATA)>

<!---->
<!--Version of the component. Ex: 1, 2.0, 3.1-->
<!---->
   <!ELEMENT version (#PCDATA)>

<!---->
<!--Operation System of component. Ex: Windows, Linux-->
<!---->
   <!ELEMENT operationSystem (#PCDATA)>

<!---->
<!--Description of component-->
<!---->
   <!ELEMENT description (#PCDATA)>

<!---->
<!--Command line to execute component-->
<!---->
   <!ELEMENT commandLine (#PCDATA)>

<!---->
<!--Params to execute component-->
<!---->
   <!ELEMENT interface (inParams,
                        outParams)>
```

```
<!---->
<!--In params to execute component-->
<!---->
   <!ELEMENT inParams (param+)>

<!---->
<!--Out params to execute component-->
<!---->
   <!ELEMENT outParams (param+)>

<!---->
<!--Param to execute component-->
<!---->
   <!ELEMENT param (paramName,
                    paramType,
                    paramDescription)>

<!---->
<!--Param name-->
<!---->
   <!ELEMENT paramName (#PCDATA)>

<!---->
<!--Param type-->
<!---->
   <!ELEMENT paramType (#PCDATA)>

<!---->
<!--Param description-->
<!---->
   <!ELEMENT paramDescription (#PCDATA)>

<!---->
<!--URL of the component. Ex:
http://www.cos.ufrj.br/ComPublish/Index.html-->
<!---->
   <!ELEMENT webSite (#PCDATA)>

<!---->
<!--Authors of component-->
<!---->
   <!ELEMENT authors (author+)>

<!---->
<!--Author of component-->
<!---->
   <!ELEMENT author (#PCDATA)>

<!---->
<!--Contacts-->
<!---->
   <!ELEMENT contacts (contact+)>

<!---->
<!--Contact-->
<!---->
   <!ELEMENT contact (description,
                      streetAddress,
                      city,
```

```
                        state,
                        postalCode,
                        country,
                        phoneNumbers,
                        mail)>

<!---->
<!--Brief explanation of what is available from this contact-->
<!---->
   <!ELEMENT description (#PCDATA)>

<!---->
<!--Line of the street address-->
<!---->
   <!ELEMENT streetAddress (#PCDATA)>

<!---->
<!--Name of the city-->
<!---->
   <!ELEMENT city (#PCDATA)>

<!---->
<!--Name of the state-->
<!---->
   <!ELEMENT state (#PCDATA)>

<!---->
<!--Number of postal code-->
<!---->
   <!ELEMENT postalCode (#PCDATA)>

<!---->
<!--Name of country-->
<!---->
   <!ELEMENT country (#PCDATA)>

<!---->
<!--List of phone numbers-->
<!---->
   <!ELEMENT phoneNumbers (phoneNumber+)>

<!---->
<!--Phone number-->
<!---->
   <!ELEMENT phoneNumber (#PCDATA)>

<!---->
<!--Mail-->
<!---->
   <!ELEMENT mail (#PCDATA)>
```

```
<!--                                                                    -->
<!-- Component Documentation DTD for ComPublish - Version 1.0          -->
<!-- Category: Source Code (classes, units, functions)                 -->
<!-- Comments: pinheiro@cos.ufrj.br                                    -->
<!--                                                                    -->

<!ELEMENT component (domain,
                     name,
                     date,
                     version,
                     language,
                     description,
                     interfaces,
                     webSite,
                     authors,
                     contacts)>

<!---->
<!--Domain of the component-->
<!---->
   <!ELEMENT domain (#PCDATA)>

<!---->
<!--Name of the component-->
<!---->
   <!ELEMENT name (#PCDATA)>

<!---->
<!--Fabrication date of component-->
<!---->
   <!ELEMENT date (#PCDATA)>

<!---->
<!--Version of the component. Ex: 1, 2.0, 3.1-->
<!---->
   <!ELEMENT version (#PCDATA)>

<!---->
<!--Development language of code component. Ex: C++, Java, Delphi-->
<!---->
   <!ELEMENT language (#PCDATA)>

<!---->
<!--Description of component-->
<!---->
   <!ELEMENT description (#PCDATA)>

<!---->
<!--Interfaces of code component-->
<!---->
   <!ELEMENT interfaces (interface+)>

<!---->
<!--Params to execute component-->
<!---->
   <!ELEMENT interface (interfaceName,
                        inParams,
                        outParam)>
```

```
<!---->
<!--Name of the class, function, etc-->
<!---->
   <!ELEMENT interfaceName (#PCDATA)>

<!---->
<!--In params to execute component-->
<!---->
   <!ELEMENT inParams (param+)>

<!---->
<!--Out params to execute component-->
<!---->
   <!ELEMENT outParam (param)>

<!---->
<!--Param to execute component-->
<!---->
   <!ELEMENT param (paramName,
                    paramType,
                    paramDescription)>

<!---->
<!--Param name-->
<!---->
   <!ELEMENT paramName (#PCDATA)>

<!---->
<!--Param type-->
<!---->
   <!ELEMENT paramType (#PCDATA)>

<!---->
<!--Param description-->
<!---->
   <!ELEMENT paramDescription (#PCDATA)>

<!---->
<!--URL of the component. Ex:
http://www.cos.ufrj.br/ComPublish/Index.html-->
<!---->
   <!ELEMENT webSite (#PCDATA)>

<!---->
<!--Authors of component-->
<!---->
   <!ELEMENT authors (author+)>

<!---->
<!--Author of component-->
<!---->
   <!ELEMENT author (#PCDATA)>

<!---->
<!--Contacts-->
<!---->
   <!ELEMENT contacts (contact+)>

<!---->
```

```
<!--Contact-->
<!---->
   <!ELEMENT contact (description,
                      streetAddress,
                      city,
                      state,
                      postalCode,
                      country,
                      phoneNumbers,
                      mail)>

<!---->
<!--Brief explanation of what is available from this contact-->
<!---->
   <!ELEMENT description (#PCDATA)>

<!---->
<!--Line of the street address-->
<!---->
   <!ELEMENT streetAddress (#PCDATA)>

<!---->
<!--Name of the city-->
<!---->
   <!ELEMENT city (#PCDATA)>

<!---->
<!--Name of the state-->
<!---->
   <!ELEMENT state (#PCDATA)>

<!---->
<!--Number of postal code-->
<!---->
   <!ELEMENT postalCode (#PCDATA)>

<!---->
<!--Name of country-->
<!---->
   <!ELEMENT country (#PCDATA)>

<!---->
<!--List of phone numbers-->
<!---->
   <!ELEMENT phoneNumbers (phoneNumber+)>

<!---->
<!--Phone number-->
<!---->
   <!ELEMENT phoneNumber (#PCDATA)>

<!---->
<!--Mail-->
<!---->
   <!ELEMENT mail (#PCDATA)>
```