Ana Ruth Viana da Silva

UTILIZAÇÃO DE AGENTES PARA DEFINIÇÃO E ALTERAÇÃO DE BANCOS DE DADOS HETEROGÊNEOS

Florianópolis -SC
2003

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ana Ruth Viana da Silva

UTILIZAÇÃO DE AGENTES PARA DEFINIÇÃO E ALTERAÇÃO DE BANCOS DE DADOS HETEROGÊNEOS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Dr. João Bosco Mangueira Sobral

Florianópolis, Março/ 2003

UTILIZAÇÃO DE AGENTES PARA DEFINIÇÃO E ALTERAÇÃO DE BANCOS DE DADOS HETEROGÊNEOS

Ana Ruth Viana da Silva

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração Engenharia de Sistemas Distribuídos e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Fernando Álvaro Ostuni Gauthier, Dr.

Banca Examinadora

_____

Prof. João Bosco Mangueira Sobral, Dr.

(Orientador)

_____

Prof. Bernardo Gonçalves Riso, Dr

_____

Prof. Rosvelter João Coelho da Costa, Dr.

“Feliz aquele que transfere o que sabe e aprende o que ensina”.
Cora Coralina

Ao Fabio que me incentivou para a
conclusão deste trabalho.

AGRADECIMENTO

Meus agradecimentos são a todos que tornaram possível este trabalho.

Um agradecimento muito especial a Daniela pela dedicação, estando sempre disponível nos momentos mais difíceis e ao professor Bosco pela orientação e incentivo.

Um agradecimento com carinho aos meus pais e a minha irmã por estarem sempre presentes.

E por fim, ao Fabio que me incentivou durante todo o trabalho, a amiga Polyana que leu e sugeriu correções gramaticais e a professora Marilene que me auxiliou no *Abstract*.

RESUMO

Este trabalho pretende dentro da Administração de Banco de Dados (DBA – *Database Administrator*), a definição e alteração de banco de dados heterogêneos, utilizando o paradigma de agentes e a Internet.

É proposta uma sociedade de agentes que trafegam na rede para atender as requisições do Administrador de Banco de Dados. Isto é feito sem utilizar uma ferramenta proprietária para cada tipo de banco de dados, e ainda sem o deslocamento de profissionais para manutenção destas bases.

A utilização de *Extensible Markup Language* (XML) na requisição do DBA e na configuração do ambiente desenvolvido tornou este flexível, uma vez que basta editar os arquivos XML para que se possa inserir uma nova funcionalidade.

A aplicação de agentes móveis em banco de dados mostra-se vantajosa em relação à arquitetura cliente-servidor, considerando que o agente desloca-se até o *host* onde se encontra a base de dados. Finalmente, o agente estabelece uma conexão com esta base e executa a tarefa localmente, minimizando assim a tráfego da rede.

Assim, a utilização da mobilidade de código e a *Web* vêm facilitar a manutenção em bases de dados como também minimizar custos.

ABSTRACT

This work aims to define and alter the heterogeneous database within the DBA (Database Administration), by utilizing the agent paradigm and the Internet.

A society of agents that moves through the net to see to the requests of the DBA is proposed. This is achieved without utilizing a proprietary software for each type of database or else without having technical 'staff providing maintenance to these bases.

The use of XML (Extensible Markup Language) in the request of DBA and in the configuration of the environment being developed rendered it flexible, as it suffices to edit the XML files in order to insert a new functionality.

The application of mobile agents in the database has shown to be advantageous regarding the client-server architecture, considering that the agent moves to the host where the database is located. Finally, the agent establishes a connection with the base and executes the task locally, thus minimizing traffic in the net.

Hence, the utilization of code mobility and the Web facilitates the maintenance of databases, as well as, minimizes costs.

SUMÁRIO

1. INTRODUÇÃO 16
1.1 MOTIVAÇÃO 16
1.2 TRABALHOS EXISTENTES 17
1.3 ORGANIZAÇÃO DO TRABALHO 18
2. BANCO DE DADOS 20
2.1 BENEFÍCIOS DE SE UTILIZAR BANCO DE DADOS 21
2.2 SISTEMAS GERENCIADORES DE BANCO DE DADOS 23
2.3 BANCO DE DADOS RELACIONAL 26
2.3.1 Características 26
2.3.2 A Linguagem SQL 29
2.4 BANCO DE DADOS ORIENTADO A OBJETOS 30
2.4.1 Características 31
2.4.2 Persistência de Objetos 34
2.4.3 Armazenamento e Acesso a Objetos Persistentes 35
2.4.4 A Linguagem OQL 36
2.5 BANCO DE DADOS OBJETO-RELACIONAL 36
2.5.1 Características 37
2.5.2 A Linguagem SQL3 40
3. AGENTES 42
3.1 CARACTERÍSTICAS DOS AGENTES 42
3.2 CLASSIFICAÇÃO DE AGENTES 43
3.3 AGENTES E OBJETOS 45
3.4 LINGUAGENS DE AGENTES 46
3.5 BENEFÍCIOS DA UTILIZAÇÃO DE AGENTES 48
4. MODELAGEM DE SISTEMAS DE AGENTES 50
4.1 METODOLOGIA GAIA 50
4.2 METODOLOGIA MAS-COMMONKADS 51
4.3 PROPOSTA DE KENDALL 53
4.4 AGENT UML 53

5. UTILIZAÇÃO DE AGENTES PARA DEFINIÇÃO E ALTERAÇÃO DE BANCO DE DADOS HETEROGÊNEOS 58

5.1 DEFINIÇÃO E MANUTENÇÃO DE BANCO DE DADOS 58

5.2 AGENTES E XML NO AMBIENTE DESENVOLVIDO 59

5.3 PLATAFORMAS DE AGENTES 61

6. IMPLEMENTAÇÃO 64

6.1 ASPECTOS GERAIS DA IMPLEMENTAÇÃO 64

6.1.1 Usuário da Aplicação 64

6.1.2 Acesso a Aplicação 65

6.1.3 Plataforma de Execução 67

6.2 CLASSIFICAÇÃO DOS AGENTES DA APLICAÇÃO 69

6.2.1 Agente Facilitador 69

6.2.2 Agente Receptor 69

6.2.3 Agentes Administradores 70

6.3 BASE DE DADOS 72

6.3.1 Relacional 72

6.3.2 Objeto Relacional 73

6.3.3 Orientado a Objetos 73

6.4 LINGUAGENS UTILIZADAS 73

6.5 FERRAMENTAS UTILIZADAS 74

7. CONCLUSÃO 75

8. REFERÊNCIAS BIBLIOGRÁFICAS 77

LISTA DE FIGURAS

Fig. 2.1 Estrutura de um Sistema Gerenciador de Banco de Dados 25

Fig. 2.2 Tabela de Alunos 27

Fig. 2.3 Exemplo de Tupla 27

Fig. 2.4 Tabela de Municípios 28

Fig. 4.1 Modelagem Gaia 50

Fig. 4.2 Diagrama de Seqüência: Comunicação entre Agentes 56

Fig. 4.3 Extensões para suporte a comunicação concorrente 57

Fig. 5.1 Ambiente Desenvolvido 60

Fig. 6.1 Tela inicial da aplicação 65

Fig. 6.2 Tela com as operações disponíveis na aplicação 66

Fig. 6.3 Tela inicial da plataforma SACI 68

Fig. 6.4 Tela do Servidor SACI 68

1. INTRODUÇÃO

Com a evolução das tecnologias de banco de dados é comum encontrar em uma mesma empresa, sistemas que utilizam modelos de dados diferentes. Além disso, nota-se uma descentralização gradativa das empresas e organizações, o que torna crítica a tarefa de administrar estes bancos de dados distintos e armazenados em locais diferentes.

Atualmente, existem algumas ferramentas disponíveis no mercado para administrar banco de dados. No entanto, são ferramentas proprietárias e não contemplam a administração de bancos de dados de modelos diferentes. Além disso, essas ferramentas são de difícil utilização quando as empresas possuem unidades físicas separadas umas das outras.

Assim, fica difícil para o Administrador de Banco de Dados (DBA), executar suas tarefas de forma rápida e segura. Outro fato, relativo aos custos, diz respeito ao deslocamento de pessoal qualificado, que requer gastos excessivos para a organização.

Inserido neste contexto, este trabalho tem como objetivo, a definição e a alteração de base dados de forma transparente para o DBA, não importando os modelos de dados e o local de armazenamento. Para isso, são utilizadas a Internet e a linguagem XML (MARTIN, 2001), juntamente com o paradigma de agentes móveis.

Os Agentes Móveis fazem parte de uma tecnologia que surgiu na área da Computação Móvel e Sistemas Distribuídos. Um dos objetivos do agente é resolver o problema da integração e da mobilidade de informações. Um agente móvel é um programa autônomo capaz de transferir-se de um ponto a outro da rede e continuar sua execução (BRENNER, 1998). Desta forma, o agente pode ser utilizado para executar tarefas de administração de banco de dados em vários pontos de uma rede.

Por fim, este trabalho propõe uma sociedade de agentes, com agentes específicos para cada modelo de dados (Relacional, Orientado a Objetos e Objeto-Relacional), sendo estes agentes capazes de se mover em uma rede para executar tarefas de definição e alteração de banco de dados.

1.1 Motivação

Este trabalho foi motivado pela leitura da dissertação *Integração de Base de Dados Utilizando a Mobilidade do Código* (CLARO, 2000), a qual foi empregado o paradigma de agentes móveis para a integração de bases de dados relacional, objeto-relacional e orientada a objetos. Desta forma, a autora obteve a integração de base de dados de modelos distintos, usando a tecnologia de agentes para consultar banco de dados. Em tal dissertação, a autora propôs ainda, como trabalho futuro, a alteração de base de dados usando os agentes e a Internet, que é o objetivo da presente dissertação.

1.2 Trabalhos Existentes

No decorrer da pesquisa para a elaboração desta dissertação, alguns trabalhos foram selecionados por estarem relacionados com o assunto proposto.

O trabalho *Integração de Base de Dados Utilizando a Mobilidade do Código* (CLARO, 2000), usa o paradigma de agentes para integrar bases de dados de modelos distintos de modo transparente para o usuário, visto que o agente é o responsável pela pesquisa de informações armazenadas nos bancos de dados. Este trabalho foi desenvolvido com a linguagem Java (DEITEL, 2001) e a plataforma de agentes Voyager (VOYAGER, 2003).

No artigo *Administração de Banco de Dados via Web* (SCHERER, 2002), é utilizada a Internet para administração de banco de dados em várias máquinas na rede. Este trabalho foi desenvolvido com a linguagem Java Server Page (BROWN, 2001) e foi utilizado um banco de dados relacional.

A ferramenta *TableEditorR* (KATTAN, 2001) tem como propósito administrar banco de dados através da *Web*. Esta ferramenta utilizou a linguagem Active Server Page (ASBURY, 2001) e banco de dados relacional.

O trabalho *Um Estudo de SGBDs Baseados em Agentes* (MACÊDO, 2000) propõe uma nova arquitetura de Sistemas Gerenciadores de Banco de Dados baseados em agentes.

Diante dos trabalhos apresentados acima, a presente dissertação tem como objetivo a definição e alteração de banco de dados relacional, objeto-relacional e orientado a objetos, utilizando para isso, a Internet, o padrão XML e o paradigma de agentes.

1.3 Organização do Trabalho

O presente trabalho está organizado em 8 capítulos, sendo que os dois subseqüentes contêm a fundamentação teórica de banco de dados e agentes. O capítulo 4 apresenta algumas linguagens e técnicas para modelagem de sistemas de agentes. Os capítulos 5 e 6 fornecem uma abordagem sobre a definição e alteração de banco de dados e a implementação do ambiente desenvolvido, os demais capítulos constituem a conclusão e a referência bibliográfica.

Capítulo 2 – Banco de Dados

Este capítulo apresenta os conceitos fundamentais sobre banco de dados e os modelos de dados tratados neste trabalho. Para isso, foram utilizados banco de dados relacional, objeto-relacional e orientado a objetos.

Capítulo 3 – Agentes

Descreve o paradigma de agentes, dando subsídios para o entendimento da utilização dos agentes. São apresentados os principais conceitos sobre os agentes, bem como suas características e classificação. Além disso, são apresentadas algumas linguagens de comunicação entre agentes.

Capítulo 4 – Modelagem de Sistemas de Agentes

Neste capítulo são expostas algumas propostas de linguagens e metodologias para a modelagem de sistemas de agentes, visto que os agentes possuem particularidades difíceis de modelar com as linguagens ou técnicas tradicionais.

Capítulo 5 – Utilização de Agentes para Definição e Alteração de Banco de Dados Heterogêneos

Neste capítulo aborda-se a definição e alteração de banco de dados tratado neste trabalho, em que é mostrada a função dos agentes móveis na implementação e a configuração dos arquivos XML do ambiente.

Capítulo 6 – Implementação

Esta fase tem como objetivo exemplificar a utilização do paradigma de agentes para um subconjunto da administração de banco de dados. Para tanto, foram utilizados bancos de dados de modelos diferentes, ou seja, banco de dados relacional, objeto-relacional e orientado a objetos.

Capítulo 7 - Conclusão

Para finalizar, o capítulo 7 compõe-se das conclusões obtidas no desenvolvimento deste trabalho.

2. BANCO DE DADOS

A organização de um volume grande de informações e a classificação dos dados significativos, sempre foi uma necessidade do ser humano. Assim, essas informações eram organizadas em papéis ou fichas, gerando uma enorme quantidade de documentos muitas vezes desatualizados, inconsistentes ou redundantes.

Por muito tempo, muitas pessoas eram responsáveis pela classificação e organização de informações manualmente.

Hoje em dia, no entanto, este processo pode ser automatizado por um computador através dos Sistemas Gerenciadores de Banco de Dados (SGBD's) (DATE, 1995). Trata-se de um conjunto de programas específicos para a manipulação e o gerenciamento de informações em computadores. A este conjunto de dados relacionados e armazenados em computador dá-se o nome de banco de dados ou base de dados (DATE, 1995).

Há uma série de dificuldades relativas à administração de dados de uma organização quando não se utiliza banco de dados. A seguir serão discutidos alguns destes problemas.

A redundância, isto é, a replicação de dados e de procedimentos para tratamento dos mesmos, torna difícil a manutenção destas informações porque quando um dado destes for alterado, tem-se que alterá-lo em todos os locais onde se encontra armazenado. Com isso, é comum a ocorrência de inconsistências, caso esta manutenção não seja controlada rigidamente. Por exemplo, suponha que um novo professor passou a trabalhar no departamento de informática. A inclusão dos seus dados foi feita no setor acadêmico. Em virtude do não compartilhamento de dados, apenas alguns dias depois foi enviado um relatório para o setor de pessoal e a inclusão realizada neste setor. Com isto, o novo professor pode perder alguns dias de salário.

Uma outra dificuldade é a falta de padronização dos dados, que ocorre porque cada aplicação define os dados de seu interesse, sem um projeto global dos dados da organização. Sendo assim, fica difícil a integração de aplicações, pois não é possível reutilizar definições de arquivos e procedimentos para manipulá-los. Pode-se tomar como exemplo as definições, mostradas anteriormente, dos arquivos de professores feitas nos setores Acadêmico e Pessoal.

A manutenção da aplicação é trabalhosa quando se deseja flexibilizar o acesso a dados, pelo fato de que cada aplicação, no seu projeto inicial, define certas operações sobre cada arquivo. Com isso, caso seja necessária posteriormente, uma nova operação, como por exemplo, uma consulta a disciplinas pelo código do departamento, além do código da própria disciplina, deve-se implementar mais um procedimento e re-compilar a aplicação.

Não há controle de segurança dos dados. Imagina-se que cada aplicação implementa rotinas básicas para garantir a consistência dos dados, como verificação de valores válidos quando ocorre uma inclusão. Porém, controles mais sofisticados, como autorizações de acesso e prevenção contra falhas, não são suportados, devido à complexidade de implementação.

Como foi visto, é muito trabalhoso e problemático armazenar informações em arquivos ou fichários. No sentido, de solucionar estes problemas e facilitar a manutenção de dados, surgiram os Sistemas Gerenciadores de Banco de Dados [1] (SGBD).

2.1 Benefícios de se utilizar Banco de Dados

Com a utilização de banco de dados todos os problemas do uso de arquivos ou fichários foram reduzidos ou eliminados. Além de gerar um desempenho melhor, a utilização de banco dados proporciona a recuperação e a atualização de dados muito mais rapidamente que um ser humano seria capaz de fazer.

A maioria dos autores entende que **Banco de dados** (BD) é um sistema computadorizado para armazenar dados inter-relacionados, cujo objetivo geral é manter a informação e torná-la disponível sob demanda (DATE, 1995).

Torna-se relevante ressaltar que dado é diferente de informação, apesar de muitas vezes serem utilizados como sinônimos.

Dado é o valor do **campo** (e é a menor unidade armazenada, (DATE, 1995)) quando se utiliza Banco de Dados. Já, a **informação** é o valor que este dado representa, acrescido de significado que tenha relevância para o indivíduo ou organização a que o sistema deve servir (DATE, 1995).

[1] Deste ponto em diante o Sistema Gerenciador de Banco de Dados será tratado pela sigla SGBD.

Entre as finalidades da utilização de banco de dados, destaca-se o armazenamento organizado das informações de uma empresa ou organização. Possibilitando uma maior otimização dos sistemas, devido à facilidade na entrada de dados, e consultas. A finalidade seria a manutenção destes dados com o intuito de manter a utilização destes de forma adequada.

A utilização da tecnologia de banco de dados traz inúmeras vantagens, se comparada com o uso de arquivos ou fichários, como:

- Redução ou Eliminação de Redundâncias - Possibilita a eliminação de dados privativos de cada sistema. Os dados, que eventualmente são comuns a mais de um sistema, são compartilhados por eles, permitindo que um dado seja consultado por vários sistemas com propósitos distintos.
- Eliminação de Inconsistências - Através do armazenamento do dado em um único local, ou com redundância controlada (fazendo com que o SGBD tenha conhecimento disso), o SGBD pode garantir que toda mudança feita em qualquer um dos dados redundantes seja propagada automaticamente em outra entrada que este possui.
- Compartilhamento dos Dados - Possibilita a utilização simultânea e segura de um dado, por mais de uma aplicação ou usuário, independentemente da operação que esteja sendo realizada. Deve ser observado apenas o processo de atualização concorrente, para não gerar erros de processamento (atualizar simultaneamente o mesmo campo).
- Restrições de Segurança - Define para cada usuário o nível de acesso a ele concedido ao arquivo e/ou dado. Este recurso impede que pessoas não autorizadas utilizem um determinado campo (valor de um dado).
- Padronização dos Dados - Permite que os campos armazenados na base de dados sejam padronizados segundo um determinado formato de armazenamento (padronização de tabela, conteúdo de campos, entre outros). Exemplo: para o campo "Sexo" somente será permitido armazenamento dos conteúdos "M" ou "F" (domínio, intervalo ou conjunto de valores válidos).

- Independência dos Dados - Representa a forma física de armazenamento dos dados no Banco de Dados e a recuperação das informações pelos programas de aplicação. Esta recuperação deverá ser independente da maneira pela qual os dados estão fisicamente armazenados. O conhecimento do formato de armazenamento do campo é totalmente transparente para o usuário.
- Manutenção da Integridade - Consiste em impedir que um determinado código ou valor único que represente o mesmo dado em outro local não tenha correspondência diferente neste. Naturalmente este problema pode surgir apenas se existir redundância nos dados armazenados. Porém podem ocorrer inconsistências mesmo não havendo redundância. O controle centralizado do banco de dados pode ajudar a evitar problemas, ao permitir que sejam definidas regras de integridade e estas regras sejam testadas sempre que se tente realizar determinada operação de atualização sobre estes dados.

2.2 Sistemas Gerenciadores de Banco de Dados

Um **Sistema Gerenciador de Banco de Dados** (SGBD) é um software responsável pelo gerenciamento (armazenamento e recuperação) dos dados no banco de dados, além de realizar controles operacionais necessários a sua integridade. Exemplos desses sistemas são Oracle (ORACLE, 2000), SQL Server (SHAPIRO, 2002), FastObject (POET, 1999), Jasmine (JASMINE, 2003), entre outros.

Os bancos de dados de maior porte exigem uma estrutura com muitas pessoas envolvidas para sua criação, definição, construção e manipulação. Sendo que cada pessoa possui um papel definido na arquitetura de um Sistema Gerenciador de Banco de Dados. Nessa arquitetura as tarefas são divididas de forma a permitir um bom aproveitamento dos recursos oferecidos.

Ao Administrador de Banco de Dados (DBA) cabe administrar o banco de dados e os *softwares* relacionados. Ele também é responsável por definir privilégios de acesso de usuários ao banco de dados, assim como coordenar, monitorar e definir necessidades de *software* e *hardware* para suportar a base de dados. Além disso, deve identificar os

requisitos de dados a serem armazenados e escolher a estrutura mais adequada para representá-los.

Os Usuários Finais são as pessoas que trabalham com o banco de dados, atualizando, consultando e gerando relatórios. O banco de dados é desenvolvido com o intuito de atingir os objetivos desta gama de usuários, caracterizando assim a organização do ambiente na sua totalidade.

Os Analistas de Sistemas desenvolvem suas especificações de acordo com os requisitos dos usuários finais, que são implementadas pelos programadores.

Na figura 2.1, pode-se observar a estrutura de um Sistema Gerenciador de Banco de Dados, no qual são identificados quatro componentes principais, ou seja, dados, *hardware*, *software* e usuários (SILBERSCHATZ, 1999).

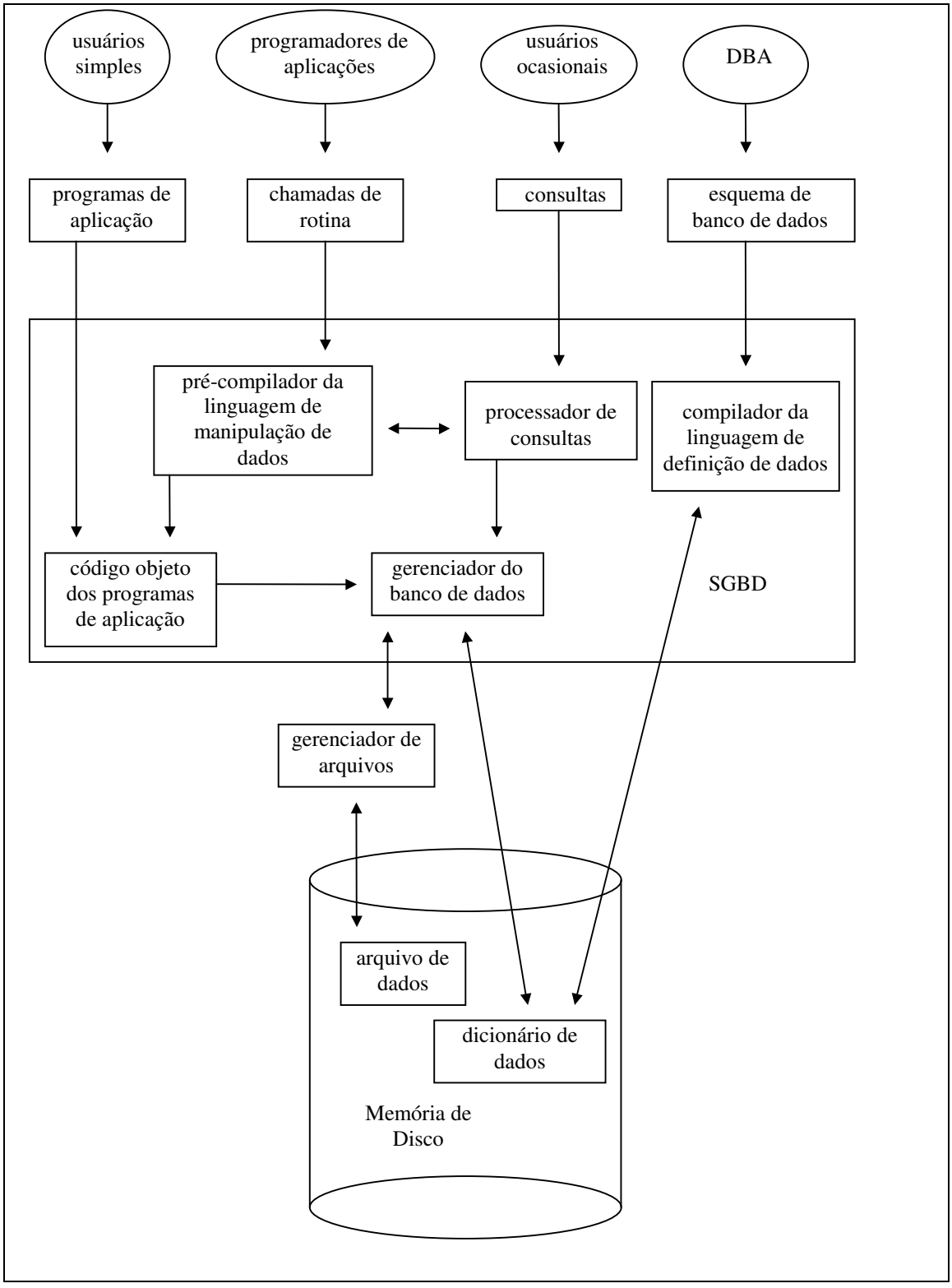

Fig. 2.1 Estrutura de um Sistema Gerenciador de Banco de Dados

2.3 Banco de Dados Relacional

O Modelo Relacional foi proposto inicialmente por E. F. Codd, no Laboratório IBM em San Jose, na Califórnia em 1970. O projeto inicial foi denominado de Sistema R, e o primeiro SGBD comercial foi o Ingres (INGRES, 2003).

No Modelo de Dados Relacional, as entidades e seus relacionamentos são representados através de relações. Relações equivalem ao conceito de conjunto na matemática e uma relação é vista como uma tabela. Sendo assim, um Banco de Dados Relacional é percebido pelo usuário como uma coleção de tabelas.

O Modelo Relacional é claramente baseado no conceito de relações, que podem ser tabuladas, onde as linhas ou *tuplas* (das tabelas) são os registros e as colunas ou atributos (das tabelas) correspondem aos campos. O conjunto de valores que podem ser assumidos por um atributo, é intitulado de **domínio**.

2.3.1 Características

De uma maneira informal, o banco de dados relacional é um conjunto de tabelas, e uma **tabela** é composta por um cabeçalho de nomes de atributos, onde cada atributo ou coluna representa uma informação atômica, isto é, um dado indivisível (escalar). As tabelas armazenadas fisicamente no banco de dados são chamadas de tabelas básicas. A tabela básica é uma tabela que recebe um nome e é autônoma, isto é, não é derivada de outra tabela. Veja a figura 2.2:

Alunos				
Matrícula	Nome	Idade	Fase	Cod_Cidade
98232015	José da Silva	26	4	01
97232058	João Ferreira	23	6	02
99232048	Gilberto Gil	21	2	09
96132587	Maria Rodrigues	30	9	08
99132746	Fabio Araújo	25	3	01
99132573	Amélia Silva	20	3	02

Figura 2.2: Tabela de Alunos

Cada **atributo** possui um nome, um tipo de dado e a obrigatoriedade ou não de especificação de valor nas linhas de dados. Não podem existir dois ou mais atributos em uma mesma tabela com o mesmo nome. O número de atributos de uma tabela é chamado de **grau**. A figura 2.2 tem como exemplo de atributos as colunas: *Matrícula*, *Nome*, *Idade*, *Fase* e *Cod_Cidade* ; como são cinco atributos o grau da tabela é cinco.

O número de tuplas de uma tabela é chamado de **cardinalidade**. Exemplo de uma tupla é mostrado na figura 2.3, nesse caso, a cardinalidade é um porque a tabela tem uma única tupla.

96132587	Maria Rodrigues	30	9	03

Fig. 2.3 Exemplo de Tupla

Já o **domínio** é um intervalo ou conjunto de valores válidos que podem aparecer em uma determinada coluna de tabela. Um exemplo de domínio é dado pelos valores possíveis (pode-se definir que a idade máxima de um aluno é 60 (sessenta) anos e a mínima é 18 (dezoito) anos) para a coluna *Idade* da tabela de alunos, representada na figura 2.2.

A **chave primária** de uma tabela é o conjunto mínimo de atributos necessários para identificar unicamente uma tupla qualquer da tabela. Na tabela da figura 2.2, o atributo *matrícula* é a chave primária, pois nunca existirá dois ou mais alunos com a

mesma *matrícula*. Quando uma chave primária possui mais de um atributo é denominada de chave primária composta.

A **chave estrangeira** de uma tabela é definida por um conjunto de atributos que existam, na forma de chave primária, em outra tabela. A tabela com as chaves estrangeiras é denominada de tabela filha, e a tabela que contém a chave primária é denominada de tabela pai. Os atributos da chave primária na tabela pai e da chave estrangeira na tabela filha não necessitam ter o mesmo nome, porém devem necessariamente ser definidas no mesmo domínio. Quando tuplas na tabela filha possuem valores para as colunas da chave estrangeira, estes valores devem existir como chave primária de alguma tupla da tabela pai. Além disso, a chave estrangeira segue os mesmos critérios de chave primária, ou seja, quando uma chave estrangeira é formada por mais de um atributo é denominada de chave estrangeira composta. Um exemplo de chave estrangeira é o atributo *Cod_Cidade*, da tabela de Alunos localizada na figura 2.2, pois este atributo é chave primária da tabela de Municípios, da figura 2.4.

Municípios	
Cod_Cidade	Nome_Cidade
01	Florianópolis
02	Joinville

Fig. 2.4 Tabela de Municípios

Um **relacionamento** é uma associação que define a relação entre entidades. A associação entre os elementos de uma tabela é representada com outra tabela. O relacionamento aparece somente no modelo de dados, no banco de dados ele é representado pela presença ou não de chaves estrangeiras ou por uma tabela.

No modelo relacional, o **dicionário de dados** é formado de tabelas (mais precisamente, tabelas de sistema, assim chamadas para distingui-las de tabelas básicas). As tabelas de sistema contêm descritores para os vários itens que são de interesse para o sistema (tabelas básicas, atributos, visões, índices, usuários, entre outras). Os usuários podem consultar o dicionário de dados exatamente da mesma forma que fazem com os seus próprios dados.

O **índice** é uma estrutura de dados que visa agilizar o acesso às informações do banco de dados.

2.3.2 A Linguagem SQL

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas à sua definição e manipulação. O departamento de pesquisas da IBM desenvolveu uma linguagem de consulta estruturada (SQL - *Structured Query Language*) como forma de interface para o sistema de BD Relacional denominado SYSTEM R, no início dos anos 70. Em 1986 o *American National Standard Institute* (ANSI) e a *International Standards Organization* (ISO), publicaram o padrão SQL. A versão padrão atual nos banco de dados relacionais é a SQL-92 (DATE, 2000).

A SQL estabeleceu-se como a linguagem padrão de banco de dados relacional.(DATE,2000). Esta apresenta uma série de comandos que permitem a definição dos dados, chamada de DDL (*Data Definition Language*), que são destinados à criação, alteração e remoção do banco, das tabelas, entre outros (Date, 2000).

Os comandos da DML (*Data Manipulation Language*), são destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas.

Estrutura básica de um comando SQL para consulta em banco de dados:

select *resultado*
from *operando*
where *predicado*

- O ***select*** corresponde à operação de projeção da álgebra relacional (DATE,2000). Ele é usado para relacionar os atributos desejados no resultado de uma consulta.
- O ***from*** corresponde à operação de produto cartesiano da álgebra relacional (DATE, 2000). Ele associa as relações que serão pesquisadas durante a evolução de uma expressão.

- O ***where*** corresponde à seleção do predicado da álgebra relacional (DATE, 2000). Ele consiste em um predicado que envolve atributos da relação que aparece na cláusula ***from***.

Exemplo:

select *nome*
from *funcionário*
where *idade > 18*

O modelo relacional é o primeiro modelo de dados que formalizou e implementou linguagens para manipulação de dados completamente independentes de uma linguagem de programação.

2.4 Banco de Dados Orientado a Objetos

O desenvolvimento dos Sistemas de Gerenciamento de Banco de Dados Orientado a Objetos (SGBDOO) teve origem na combinação de idéias dos modelos de dados relacionais e de linguagem de programação orientada a objetos. Entretanto, somente em meados dos anos 80, eles começaram a se tornar produtos comercialmente viáveis (KOSHAFIAN, 1994).

O Modelo de Dados Orientado a Objetos foi proposto para atender a exigência de aplicações que surgiram recentemente, como: desenho auxiliado por computador (CAD), manufatura auxiliada por computador (CAM), engenharia de software (CASE) e banco de dados multimídia (NASSU, 1999). Tais aplicações não eram possíveis para os modelos de dados existentes. Este modelo de dados tem um papel importante nos SGBDs, pois são mais adequados para o tratamento de objetos complexos (textos, gráficos, imagens) e dinâmicos (programas, simulações).

No SGBDOO, a noção de objetos é usada no nível lógico e possui características não encontradas nas linguagens de programação estruturadas, como operadores de manipulação de estruturas, gerenciamento de armazenamento, tratamento de integridade e persistência dos objetos (NASSU, 1999).

O *Object Database Management Group* (ODMG) desenvolveu uma linguagem de consulta para SGBDOO em 1993, denominada **OQL** (*Object Query Language*) (ODMG, 2000).

2.4.1 Características

Objeto

Um objeto é uma entidade real ou abstrata, que possui uma identidade e que exibe um comportamento bem definido. Um objeto tem um estado que inclui todas as suas propriedades e seus valores. Além disso, eles respondem a mensagens que lhe são enviadas, retornando um valor, que também é um objeto.

Em um objeto, os dados armazenados são chamados de valores dos objetos, e as estruturas de cada valor são chamadas de tipo.

Identidade de Objeto

A identidade de objeto ou OID é um identificador implícito, único de cada objeto, que é independente do valor desse objeto e de seu local físico de armazenamento. Sendo assim, mesmo que dois objetos tenham os mesmos valores eles não são idênticos porque essa identidade é única. A identidade de objeto é geralmente gerada pelo sistema, e o objeto mantém a sua identidade mesmo que alguns ou todos os valores de variáveis mudem com o tempo.

Uma grande vantagem da identidade de objetos é que ela elimina as anomalias de atualização e de integridade referencial, uma vez que a atualização de um objeto será automaticamente refletida nos objetos que o referenciam e o identificador de um objeto não tem seu valor alterado.

Objetos Complexos

Os objetos complexos ou compostos representam a semântica de agregação de vários objetos formando um novo. Esta agregação espelha o relacionamento “parte de”. Eles são construídos através da aplicação de métodos construtores (conjuntos, listas, tuplas, registros, coleções, arrays) aplicados a objetos simples (inteiros, booleanos,

strings). Um exemplo de um objeto complexo pode ser o projeto de um carro, este contém várias outras partes como pneus, embreagem, volante, entre outras coisas.

No modelo orientado a objetos, os construtores são em geral ortogonais, isto é, qualquer construtor pode ser aplicado a qualquer objeto. Já no modelo relacional não são ortogonais, porque o conjunto de construtores somente pode ser aplicado a tuplas e o construtor de tuplas somente pode ser aplicado a valores atômicos.

Classes

As classes são definidas como um conjunto de objetos que compartilham a mesma estrutura e o mesmo comportamento. Um objeto é uma instância de uma classe, ou seja, suas características são definidas pela classe na qual é instanciado.

Extensibilidade

Todos os sistemas de BD têm um conjunto pré-definido de tipos básicos (inteiro, booleano, string). Entretanto, nos SGBDOO tem-se novo tipo de dados criado pelos usuários. Estes não podem ter tratamento diferente em relação aos tipos pré-definidos, pelo menos na visão do usuário. Assim, os sistemas de BD devem permitir a criação de novos tipos de dados, garantindo a integridade do mesmo.

Versões de Objetos

O controle de versões de objetos é uma característica que está presente em alguns SGBDOO. Este controle de versões de estados anteriores de objetos é importante para que se possa retornar a um estado anterior do banco de dados, no caso de algum objeto possuir algum erro. O controle de versões é também utilizado quando se deseja manter uma versão já testada de um objeto enquanto a nova versão do objeto está em estágio de teste.

Atributo

Na tecnologia da orientação a objetos um atributo não se limita a um valor atômico ou conjuntos sem ordem, como acontece no modelo relacional. A referência a

um atributo na orientação a objetos é feita através do operador "**.**" (ponto). Por exemplo, o atributo nome de um objeto f da classe *Funcionário* é referenciado por: ***f.nome***.

Atributo Simples

São atributos que contêm valores simples como inteiros e strings, incluindo os atributos BLOBs (*Binary Long Objects*) que representam valores extensos como imagens, textos, áudio, etc. Um exemplo de atributos simples na classe *Funcionário* é *nome (**nome: string**).*

Atributo de Referência

São atributos utilizados para representar relacionamentos entre os objetos. Eles podem ser vistos como ponteiros para objetos e substituem as chaves estrangeiras do modelo relacional. Entretanto, esses atributos possuem vantagens em relação a ponteiros ou chaves. Ao contrário de ponteiros, atributos de referência não podem ser danificados, são automaticamente invalidados quando o objeto referenciado é removido. Estes atributos não são associados a um valor visível, como na chave estrangeira, e sim ao objeto como um todo através do seu OID (KHOSHAFIAN, 1994).

Um exemplo de atributo de referência na classe *Funcionário* é *dept* (***dept:Departamento***). Sendo assim, o atributo *dept* contém um objeto (referência) da classe *Departamento* e não um valor representativo (chave estrangeira) como o nome do departamento.

Atributo de Coleção

São atributos usados para representar listas, conjuntos e vetores. No caso de listas e vetores há uma ordem explícita entre os valores mantidos pelo SGBD. Essa semântica é muito freqüente na modelagem de entidades. No modelo relacional, por exemplo, para a utilização de listas deve-se utilizar uma linguagem de programação, porque o SGBD não suporta este tipo de atributo.

Os elementos do atributo de coleção podem ser valores (atributos simples) ou objetos (atributos de referência). Um exemplo de atributo de coleção de referências na classe *Funcionário* é *telefones* (***telefones: set [Telefone]***).

Atributo Chave de Objeto

Neste modelo de objetos também existe o conceito de chave de objetos, pois o identificador do objeto não pode substituir o conceito de chave. Por exemplo, sem chaves, dois funcionários diferentes podem ser criados erradamente com o mesmo número de CPF, já que seriam duas instâncias de objetos e teriam identificadores de objeto distintos. No exemplo da classe *Funcionário*, o atributo *matricula* foi eleito como chave para que o SGBD faça uma verificação da restrição de integridade e unicidade entre chaves. Essa verificação não pode ser substituída por um método da classe, já que os métodos atuam sobre instâncias individualmente e não sobre a extensão da classe como os operadores da álgebra relacional (MATTOSO, 1998).

Relacionamento

A representação de relacionamentos na orientação a objetos é feita através de atributos de referência. Essa forma de representar relacionamentos aumenta o poder semântico do SGBD, já que é mais natural para o usuário e também facilita a criação de tipos complexos. Essa representação é um dos mais importantes aspectos diferenciais entre o modelo relacional e o modelo orientado a objetos.

2.4.2 Persistência de Objetos

Um objeto é considerado persistente em uma base de dados quando ele continua existindo após o final da transação que o criou, caso contrário é chamado de transiente. Para garantir que um objeto continuará existindo, é preciso torná-lo persistente. Há várias formas de implementar persistência (MATTOSO, 1998):

- Persistência por tipo: a persistência é obtida na criação do objeto, baseado no seu tipo. O tipo pode ser identificado como persistente na sua declaração ou por ser subtipo de um tipo persistente.
- Persistência explícita (por nome): o usuário pode estipular a persistência do objeto explicitamente, mesmo após a sua criação como transiente. Objetos e valores como nome são persistentes.

- Persistência por alcance: quando um objeto ou valor é tornado persistente, todos os seus componentes ganham persistência, ou seja, todos os objetos que podem ser alcançados por um objeto ou valor persistente são também persistentes.

2.4.3 Armazenamento e Acesso a Objetos Persistentes

Armazenar um objeto em um banco de dados significa armazenar a parte de dados do objeto. Logicamente, o código que implementa os métodos de uma classe deve ser armazenado em um banco de dados como parte do esquema, junto com as definições de tipo das classes.

Existem alguns SGBDOO que possuem uma linguagem particular para definição dos objetos, como é o caso do Jasmine (JASMINE, 2003). Entretanto, há os que usam linguagens de programação orientada a objetos, que é o caso do Poet (C++ e Java) (POET, 2003), entre outros.

Há várias maneiras de recuperar um objeto em uma base de dados. Uma delas é a nomeação dos objetos que funciona para um número relativamente pequeno de objetos, mas não para uma quantidade mais significativa de objetos. Uma segunda abordagem é a alocação dos identificadores de objetos ou ponteiros persistentes a objetos que podem ser armazenados externamente. Ao contrário dos nomes, esses ponteiros não têm de ser mnemônicos e podem mesmo ser ponteiros físicos dentro de um banco de dados. Uma terceira abordagem seria armazenar conjuntos de objetos e permitir que programas interajam sobre estes conjuntos para encontrar os objetos exigidos.

Um caso especial de um conjunto é uma classe extensão, que é a coleção de todos os objetos pertencentes à classe. Se uma classe extensão existe para uma classe, então sempre que um objeto da classe é criado, esse objeto é inserido na classe extensão automaticamente.

Muitos sistemas de banco de dados orientados a objeto aceitam os três modos de acessar objetos persistentes.

Uma forma de recuperar as informações contidas nas classes de extensão seria a utilização da OQL, que é uma linguagem de consulta a objetos, padrão da ODMG, adotada por vários SGBDOOs. (SILBERSCHATZ, 1999).

2.4.4 A Linguagem OQL

A linguagem OQL padrão da ODMG é declarativa e foi baseada na sintaxe do SQL 92, enquanto que esta manipula tabelas, linhas e colunas, aquela manipula classes de extensão, objetos e atributos.

O OQL não é uma linguagem completa para desenvolvimento de aplicações, porque não possui comando de definição, alteração e exclusão, estes são realizado através da utilização de operações definidas na LPOO (Linguagem de programação orientada a objetos). Em OQL, consultas são especificadas usando objetos e retornam conjunto de objetos ou um literal. Uma consulta usada freqüentemente pode ser armazenada no banco de dados sendo delimitada para ela um nome com a qual poderá ser referenciada posteriormente.

O sistema de tipos adotado pelo OQL é baseado no mesmo sistema das LPOO. Além disso, OQL também provê primitivas para manipular coleções (conjuntos, listas, entre outras) e consultas, que podem ser embutidas dentro da LPOO.

A linguagem OQL permite a construção de expressões mais abrangentes do que *select-from-where* do SQL. Possui construções que permitem o acesso a estruturas próprias de sistemas orientados a objeto: navegação em estruturas aninhadas; operadores para construtores do tipo lista, tupla, conjunto; ativação de métodos; entre outras.

2.5 Banco de Dados Objeto-Relacional

A mudança de paradigma da utilização de banco de dados relacionais para os banco de dados orientados a objetos decorreu em um forte repúdio desta tecnologia. Com o intuito de amenizar a utilização de armazenamento de objetos na sua forma real, surgiram os banco de dados objeto-relacionais.

Os Bancos de Dados Objeto-Relacional buscam estender os sistemas de banco de dados relacionais com funcionalidades necessárias para dar suporte a uma classe mais ampla de aplicações e, de certa forma, prover uma ponte entre os paradigmas relacionais e orientados a objetos (SALGADO, 2000).

O banco de dados objeto-relacional permite acrescentar tipos dados extensíveis, objetos complexos e herança ao SQL 92. A esta extensão dá-se o nome de SQL3.

2.5.1 Características

O modelo de dados objeto-relacional incorporou conceitos de orientação a objetos que não existiam no modelo relacional. A seguir serão mostrados alguns desses conceitos.

Extensão de Tipos

A característica de extensão de tipos permite que o usuário possa manipular além dos tipos pré-definidos pelo banco de dados, novos tipos de dados criados por ele mesmo. Pode haver as seguintes extensões:

- Tipos de Dados Extensíveis: são tipos embutidos no próprio banco de dados objeto-relacional. Como por exemplo, no SGBD Ilustra da Informix, há tipos de dados geométricos, que podem ser utilizados pelo usuário para atribuir aos seus dados este tipo. Existem também funções pré-definidas para manipular estes dados, como por exemplo, o cálculo da distância entre dois pontos em um polígono.
- Tipo Abstrato de Dados: torna encapsulada a implementação, sendo visível apenas as interfaces das funções e os atributos, sendo composta em duas partes:
  1. Especificação dos atributos.
  2. Especificação das funções.
- Criação de Tipos pelo Usuário: são tipos que o próprio usuário define dependendo de suas necessidades.

- Funções Definidas pelo Usuário: estas funções podem ser escritas em uma linguagem de manipulação de dados, como a SQL3 ou com uma linguagem de programação, por exemplo, C ou Java.
- Referência a Tipos: estas referências a tipos de objetos são equivalentes a ponteiros em linguagens de programação, servem para representar relacionamentos e se referenciam ao OID de um objeto persistente. A referência a um tipo de objeto substitui a chave estrangeira do modelo relacional.

Objetos Complexos

São os objetos que o usuário poderá manipular, além dos dados simples como inteiros, reais, caracteres, entre outros. Eles podem ser: registros, conjuntos, listas, pilhas, filas, array e referências, que são ponteiros para instâncias em outras tabelas e são os substitutos naturais de chaves primárias e estrangeiras.

Herança

A herança incentiva a modularidade e a reutilização consistente dos componentes do esquema. Ela pode ser no nível de tipos ou no nível de tabelas.

Exemplo: herança de tipos

```
create type Pessoa (
        nome varchar(50),
        cpf varchar (20) );

create type Estudante (
        curso varchar(20),
        departamento varchar(20) )
        under Pessoa;
```

create type Professor (
salario **integer**,
departamento **varchar(20)**)
under Pessoa;

Os subtipos *Estudante* e *Professor* herdam os atributos do supertipo *Pessoa*.

Exemplo: herança de tabelas

create table Pessoas (
nome **varchar(50),**
cpf **integer**);

create table Estudantes (
curso **varchar(20),**
departamento **varchar(20)**)
under Pessoas;

create table Professores (
salario **integer**,
departamento **varchar(20)**)
under Pessoas;

Há requisitos de consistência sobre subtabelas e supertabelas:

- Cada tupla da supertabela *Pessoas* pode corresponder a no máximo uma tupla em cada uma das tabelas *Estudantes* e *Professores*, isto é, ter os mesmos valores para todos os atributos herdados.
- Cada tupla em *Estudantes* e *Professores* deve ter, exatamente, uma tupla correspondente em *Pessoas*, isto é, com os mesmos valores para todos os atributos herdados.

  Sem a primeira condição, poderíamos ter duas tuplas em *Estudantes* ou *Professores* que corresponderiam à mesma pessoa. Sem a segunda condição,

poderíamos ter uma tupla em *Estudantes* ou em *Professores* que não corresponderiam a qualquer tupla em *Pessoas*, ou corresponderiam a várias tuplas em *Pessoas*. Esses estados não corresponderiam ao mundo real e consequentemente tornam-se inconsistentes (SILBERSCHATZ, 1999).

2.5.2 A Linguagem SQL3

A linguagem SQL3 foi desenvolvida como uma extensão do SQL-92 com suporte para os objetos, além de novos tipos, por exemplo, conjunto, lista, BLOB.

A linguagem de definição e manipulação de dados é semelhante a do SQL-92.

- Definição de tipos:

Sintaxe:

create type *nome_do_tipo* ***as object (***
Lista de atributos);

Exemplo:

create type *Pessoa* ***as object (***
nome ***varchar(50),***
profissao ***varchar(20),***
data_nasc ***varchar(15),***
sexo ***varchar(15));***

Definição de funções:

Sintaxe:

create function *nome_da_função(arg1, …, argn)*
return *arg* ***as***
(expressão SQL ou nome do arquivo);

Onde ***return*** retorna o tipo do argumento que foi enviado, e *expressão SQL* ou *nome do arquivo* é a funcionalidade da função que pode ser tanto uma expressão SQL, como um arquivo já compilado escrito em alguma linguagem de programação.

A declaração ***select*** pode retornar um conjunto de valores. Se o retorno da função for um tipo complexo, o resultado será todo o conjunto. Caso o tipo de retorno não for um tipo-conjunto, o resultado gerado pela declaração ***select*** deve conter somente uma *tupla*, que é retornado como resposta. Se existir mais de uma *tupla* no resultado do comando ***select***, o sistema tem duas possibilidades: tratar esta situação com um erro ou selecionar uma *tupla* arbitrariamente e retorná-la como resposta.

Exemplo:

create function *INSS (valor* ***float****)*
returns float as
select *(valor*0.08);*

- Criação de tabela: a sintaxe abaixo mostra a criação de uma tabela de objetos.

Sintaxe:

create table *nome_da_tabela* ***of*** *nome_do_tipo;*

Exemplo:

create table *tabPessoa* ***of*** *Pessoa;*

3. AGENTES

Segundo AMANDI (1997), um agente é uma entidade computacional com um comportamento autônomo que lhe permite tomar suas decisões para agir, levando em consideração as mudanças acontecidas no ambiente em que atua e o seu objetivo. Os agentes podem ser aplicados em vários tipos de aplicações, por exemplo: gerenciamento de redes, controle de tráfego aéreo, gerenciamento e recuperação de informações, comércio eletrônico, gerenciamento de horário/agenda, integração de banco de dados, entre outros.

3.1 Características dos Agentes

A seguir serão descritas em detalhes algumas características de um agente:

- Autonomia: executam a maior parte de suas ações sem interferência direta de agentes humanos ou de outros agentes computacionais, possuindo controle total sobre suas ações e estado interno.
- Habilidade Social: são os agentes que interagem com outros agentes e possivelmente com seres humanos, através de algum tipo de linguagem de comunicação.
- Reatividade: percebem e reagem a alterações nos ambientes em que estiverem inseridos.
- Pró-Atividade: são agentes, do tipo deliberativo, além de atuar em resposta às alterações ocorridas em seu ambiente, apresentam um comportamento orientado a objetivos, tomando iniciativas quando julgarem apropriadas.
- Orientação a Objetivos: os agentes devem ser capazes de lidar com tarefas complexas em alto nível. A decisão de como uma tarefa é melhor subdividida em tarefas menores, em qual ordem e de que modo devem ser executadas, é realizada pelo próprio agente.
- Capacidade de Aprendizagem: o agente deve possuir um certo grau de inteligência que o habilite a agir perante mudanças no seu ambiente de

execução e também estar apto a aprender com situações presenciadas no seu ambiente.

- Mobilidade: é a habilidade do agente mover-se dentro de redes de comunicação eletrônica.
- Cooperação e Comunicação*:* por impossibilidade de resolução de certos problemas ou por outro tipo de conveniência, os agentes interagem com outros agentes (humanos ou computacionais), para completarem a resolução dos seus problemas, ou ainda para auxiliarem os outros agentes.
- Adaptabilidade: um agente deve ser capaz de adaptar-se aos hábitos, métodos de trabalho e preferências de seus usuários.

Segundo OLIVEIRA (1996), os Sistemas Multi-Agentes (SMA) são compostos por entidades computacionais, denominadas agentes, com capacidades e objetivos individuais que, uma vez agrupados em sociedade, trabalham juntos visando atingir o objetivo do sistema; sendo que os agentes devem raciocinar a respeito das ações e sobre o processo de coordenação entre si.

JENNINGS (1196) afirma que as características dos sistemas multi-agentes têm vantagens significativas, dentre elas:

- Maior rapidez na resolução de problemas através do aproveitamento do paralelismo.
- Diminuição da comunicação por transmitir somente soluções parciais em alto nível para outros agentes ao invés de dados brutos para um lugar central.
- Mais flexibilidade por ter agentes de diferentes habilidades dinamicamente agrupados para resolver problemas.
- Aumento da segurança por permitir que agentes assumam responsabilidades de agentes que falham.

3.2 Classificação de Agentes

Com base no trabalho de (JENNINGS, 1996), os agentes podem ser classificados de acordo com: (a) sua capacidade de resolver problemas, (b) autonomia, aprendizagem e cooperação e (c) outros agentes.

(a) Classificação de agentes baseado na sua capacidade de resolver problemas:

- Agentes Reativos: Eles reagem a mudanças de seu ambiente ou mensagens provenientes de outros agentes. As suas ações se realizam como resultado de regras que se disparam ou da execução de planos. Os sistemas especialistas de primeira geração (composta de uma base de conhecimento que contém conjuntos de regras, fatos e uma máquina de inferências) são exemplos deste tipo de agentes. Dentro de um SMA, este tipo de agentes pode também se comunicar com outros agentes; escolhendo e enviando ou recebendo e interpretando mensagens de acordo com a situação atual.
- Agentes Intencionais: São capazes de raciocinar sobre suas intenções e conhecimentos, criar planos de ações, e executá-los. Os agentes intencionais podem ser considerados como sistemas de planejamento. Em SMA, os agentes intencionais coordenam entre si trocando informação sobre suas crenças, metas ou ações. Esta informação é somada a seus planos.
- Agentes Sociais: Esses possuem a capacidade dos agentes intencionais, e os modelos explícitos de outros agentes.

(b) Classificação de agentes baseado em autonomia, aprendizagem e cooperação: esta classificação se baseia nas três características dos agentes: autonomia, cooperação e aprendizagem:

- Agentes Colaboradores: Estes agentes enfatizam sua autonomia e cooperação (com outros agentes) para realizar as suas tarefas. Para ter um grupo coordenado de agentes colaboradores, estes têm que negociar para alcançar compromissos mutuamente aceitados de alguma forma.
- Agentes de Interface: Os agentes de interface colocam ênfase na sua autonomia e capacidade de aprendizagem para realizar as suas tarefas. O exemplo mais

claro deste tipo de agente corresponde a um assistente pessoal que colabora com seu usuário no mesmo ambiente de trabalho. A colaboração com o usuário não requer necessariamente uma linguagem explícita de comunicação de agentes. Essencialmente, os agentes de interface assistem e dão apoio ao usuário para aprender o uso de uma aplicação. O agente do usuário observa e monitora suas ações através da interface com o usuário, e lhe dá sugestões para melhorar a sua tarefa.

(c) Outros tipos de agentes:

- Agentes Móveis: Os agentes móveis são programas capazes de viajar através de redes de computadores, como Internet, de interagir com *Host*, pedir informação em nome de seu usuário e voltar ao seu lugar de origem, onde recebeu as tarefas especificadas pelo seu usuário.
- Agentes de Informação/Internet: Os agentes de informação realizam a tarefa de administrar, manipular ou coletar informação proveniente de várias fontes distribuídas. Os agentes de informação podem ser estáticos ou móveis, podem ser não cooperativos ou sociais, e eles podem ou não aprender.
- Agentes Híbridos: Os agentes híbridos são os que possuem duas ou mais combinações das capacidades dos tipos anteriormente explicados.

3.3 Agentes e objetos

Uma outra discussão muito comum no entendimento do termo agente é a diferença entre agentes e objetos. As diferenças básicas entre agentes e objetos são (WOOLDRIDGE, 2000):

- agentes incorporam um forte noção de autonomia não encontrada em objetos. Em particular, eles decidem por eles mesmos se devem ou não executar uma ação quando requisitados por outros agentes.
- agentes são capazes de realizar comportamentos flexíveis (reativos, pró-ativos, sociais), o que não é encontrado num modelo de objetos padrão.

- sistemas multi-agentes são inerentemente *multi-threaded,* um thread é um processo. Todos os threads pertencentes a um processo compartilham a mesma área global. Um sistema *multi-threaded* apresenta diversos *threads* executando concorrentemente, onde cada agente executa em um *thread* separado.

3.4 Linguagens de Agentes

O KQML (Knowledge Query and Manipulation Language), do projeto Knowledge Sharing Effort (KSE), é uma linguagem e um protocolo para a comunicação entre agentes, visando suportar a troca de informação e conhecimento entre os agentes. KQML pode ser utilizada como a linguagem que um agente utiliza para interagir com um outro agente, ou para dois ou mais sistemas inteligentes compartilharem conhecimento para solução cooperativa de problemas (FININ, 1997).

A KQML faz parte de um projeto de pesquisa de maior abrangência chamada DARPA Knowledge Sharing Effort (KSE), que faz parte do NII (National Information Infrastructure), cujo objetivo é aumentar a produtividade na construção de sistemas de conhecimento complexos, através de compartilhamento e reutilização de módulos ou sistemas pré-definidos (FININ, 1993).

KSE usa a abordagem declarativa, que é baseada na idéia de que a comunicação pode ser modelada com a troca de afirmações declarativas, sendo, porém, ao mesmo tempo compactada e suficientemente expressiva para comunicar uma grande quantidade de tipos de informações. Este grupo definiu uma Linguagem de LCA, com os seguintes requisitos: um vocabulário, uma linguagem interna chamada KIF (Knowledge Interchange Format) e o KQML (Knowledge Query and Manipulation Language) (FININ, 1997).

Assim, segundo (FININ, 1997) uma mensagem LCA é uma expressão KQML na qual os argumentos são termos ou sentenças expressadas em KIF, formadas por palavras encontradas no vocabulário da LCA.

O vocabulário da LCA é armazenado em um dicionário com as palavras apropriadas às áreas comuns das aplicações. Trata-se de um dicionário aberto, permitindo acréscimo de palavras nas áreas já existentes ou então de novas áreas que possam surgir.

O KIF é uma linguagem interna necessária para expressar o conteúdo da mensagem. É uma representação em ASCII de uma versão estendida de cálculo de predicados de primeira ordem, com várias extensões para reforçar sua expressividade e dar suporte à codificação de dados simples.

Para que se estabeleça uma comunicação entre agentes, é preciso definir uma ontologia, ou seja, um conjunto de definições associadas aos termos para que todos os agentes, que utilizem desses termos, estejam compartilhando a mesma informação.

Segundo (FININ, 1997) KQML é uma linguagem que pode ser dividida em três camadas:

- A camada de conteúdo possui a mensagem e pode ter no seu conteúdo qualquer representação de linguagem.
- A camada de mensagem é usada para codificar a mensagem a ser transmitida (de uma aplicação para outra). A função primária desta camada é identificar o protocolo usado para transmitir a mensagem e a primitiva que o transmissor anexo no conteúdo (linguagem, ontologia e tipos de atos de fala). A primitiva pode ser uma declaração, uma consulta, um comando ou um conjunto de primitivas conhecidas.
- A camada de comunicação codifica um conjunto de características para aceitar parâmetros de alto nível, tais como a identidade de quem envia e recebe a mensagem e também um identificador único associado com a mensagem.

A KQML foi projetada para ser usado com vários mecanismos (atualmente há implementações que usam TCP/IP, SMTP (E-mail), HTTP e CORBA). O agente que usa KQML pode falar diretamente a outro agente ou pode enviar mensagens a múltiplos agentes do mesmo grupo (FININ, 1994).

O ambiente operacional para agentes KQML é altamente distribuído, heterogêneo e extremamente dinâmico. Para satisfazer as exigências de tal ambiente, a linguagem KQML provê ferramentas formais que trabalham com outras linguagens e protocolos.

Apesar de atender as principais características de uma linguagem de comunicação de agentes, KQML ainda não tratou adequadamente as questões de segurança e autenticação de agentes em mensagens.

A linguagem KQML simplifica a implementação em ambientes distribuídos, permitindo que as mensagens "carreguem" qualquer informação útil, tal como os nomes e endereços dos agentes de envio e recebimento, um único identificador de mensagem, e notações para qualquer agente interveniente. Há também características opcionais da linguagem KQML que contêm descrições do conteúdo: sua linguagem, a forma sintática que esta assume, e alguns tipos de descrições mais gerais, tais como um descritor nomeando um tópico em pacote.

3.5 Benefícios da Utilização de Agentes

Segundo OSHIMA (1998), agentes móveis apresentam uma série de vantagens como:

- Redução do tráfego da rede **-** Sistemas distribuídos demandam um grande volume de comunicação (interação) para realizar uma determinada tarefa, principalmente quando há restrições de segurança envolvidas. Agentes móveis permitem reduzir o tráfego da rede, pois permitem despachar tarefas que podem executar suas interações localmente. Agentes móveis podem ainda reduzir o tráfego de dados da rede, pois permitem mover o processamento para o local onde os dados estão armazenados ao invés de transferir os dados para depois processá-los. O princípio é simples: "Mover o processamento para os dados ao invés de mover os dados para o local de processamento".

- Oculta a latência da rede - Sistemas críticos necessitam de respostas em tempo real para mudanças no ambiente. O controle desses sistemas através de uma rede substancialmente grande ocasiona uma latência inaceitável. Agentes móveis oferece uma solução, pois podem ser despachados pelo controlador central para realizarem suas tarefas localmente.

- Encapsulamento de protocolo - Cada máquina em um sistema distribuído possui seu próprio código necessário para implementar a transferência de dados. Porém, novos requisitos de segurança e eficiência demandam mudanças no protocolo que podem ocasionar problemas na manutenção do código existente. Agentes móveis, por outro lado, podem mover-se para máquinas remotas a fim de estabelecer canais de comunicação baseados em protocolos proprietários.

- Execução assíncrona e autônoma - Tarefas podem ser embutidas em agentes móveis que podem ser despachados pela rede. Após serem despachados, os agentes são autônomos e independentes da criação de processo, podendo executar assincronamente. Este recurso é útil principalmente porque um dispositivo móvel (ex. laptops) pode se reconectar na rede para coletar o agente mais tarde.

- Adaptação dinâmica - Agentes móveis possuem a habilidade de perceber mudanças no ambiente de execução e reagir autonomamente. Múltiplos agentes podem interagir entre si e se distribuir pela rede, de modo a manter uma configuração ótima para resolver um problema em particular.

- Independência de plataforma - Redes de computadores, geralmente são heterogêneas, tanto na perspectiva de hardware como a de software. Agentes móveis são independentes da máquina e também da rede, sendo dependentes somente do seu ambiente de execução, não dificultando a integração de sistemas.

- Robustez e tolerância a falhas - A habilidade dos agentes móveis de reagirem dinamicamente a situações e eventos desfavoráveis torna fácil a construção de sistemas distribuídos robustos e tolerantes a falhas. Se uma máquina está para ser desligada, todos os agentes em execução na máquina podem ser advertidos para que possam ser despachados e continuar suas tarefas em outra máquina da rede.

4. MODELAGEM DE SISTEMAS DE AGENTES

Com a crescente utilização da tecnologia de agentes, vários pesquisadores e desenvolvedores propuseram metodologias para a modelagem de sistemas envolvendo agentes, sendo que cada um destas propostas possui sua própria notação. A seguir, serão apresentadas algumas destas metodologias.

4.1 Metodologia Gaia

É um processo de análise e projeto Orientado a Agentes (OA) proposto por WOOLDRIDGE (2000). A figura 4.1 mostra dois modelos utilizados para a fase de análise e o relacionamento com os três modelos da fase de projeto.

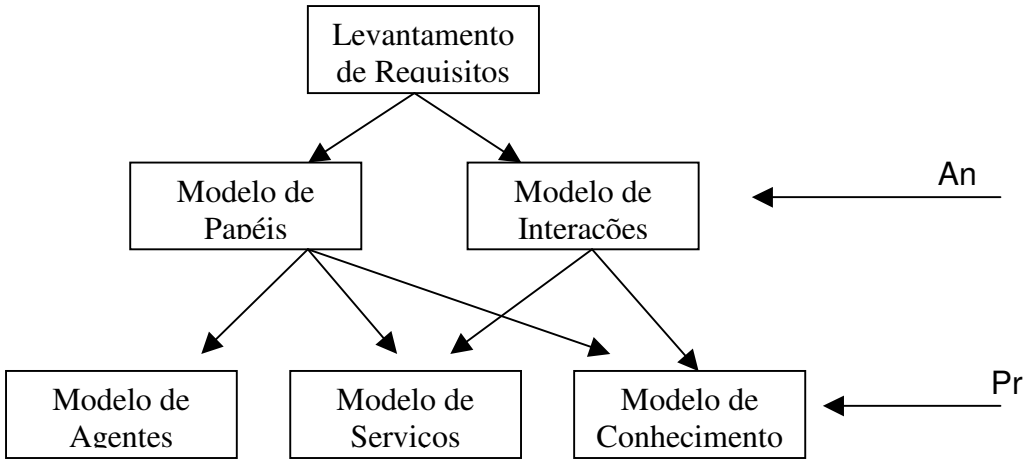

Fig. 4.1 Modelagem Gaia

Na metodologia Gaia, o objetivo da fase de análise é possibilitar o entendimento do sistema e de sua estrutura. Para isso, são definidos dois modelos na fase de análise:

- Modelo de Papéis (*Roles*): identifica os papéis existentes no sistema, sendo estes descrições abstratas de uma função pretendida por uma entidade. Estes papéis são caracterizados por dois tipos de atributos: permissões associadas ao papel e responsabilidades do papel (WOOLDRIDGE, 2000).

- Modelo de Interação: é constituído por um conjunto de definições de protocolo, para cada tipo de interação entre papéis. A definição do protocolo consiste em determinar valores para os atributos: iniciante, receptor, dados de entrada, dados de saída e descrição do processamento da interação.

Segundo WOOLDRIDGE (2000), estes dois modelos almejam garantir que aspectos como estrutura, organização, cooperação e interação sejam estabelecidos no nível de análise.

A fase de projeto da metodologia Gaia tem como objetivo transformar o modelo de análise em modelos de baixo nível de abstração, que possam ser utilizados na implementação dos agentes. São definidos três modelos:

- Modelo de Agente - define os tipos de agente que irão fazer parte do sistema.
- Modelo de Serviços - define os principais serviços que serão atribuídos a cada tipo
  de agente.
- Modelo de Conhecimento - define conhecimentos para cada tipo de agente.

A notação utilizada por Gaia, nos modelos, é baseada na notação FUSION (COL, 1994).

4.2 Metodologia MAS-CommonKADS

Esta metodologia foi proposta por Iglesias, é uma extensão multi-agente da metodologia estruturada de suporte à engenharia do conhecimento, chamada CommonKADS (IGLESIAS, 1998).

São propostos os seguintes modelos:

- Modelo do Agente - especifica as características do agente, tais como capacidade de raciocínio, percepção e ação, serviços, grupos e hierarquias do agente.

- Modelo de Tarefas - descreve as tarefas que o agente pode realizar.
- Modelo de Habilidades - descreve o conhecimento que os agentes necessitam para atingir os seus objetivos.
- Modelo de Organização - descreve a organização na qual o SMA está sendo introduzido e a organização da sociedade de agentes.
- Modelo de Coordenação - descreve as conversações entre agentes (interações, protocolos e capacidades requeridas).
- Modelo de Comunicação - descreve detalhes das interações entre humanos e agentes de software, como também fatores humanos a serem considerados no desenvolvimento de interfaces de usuário.
- Modelo de Projeto - coleta os modelos anteriores. Consiste de três submodelos: projeto da rede, do agente e da plataforma.

Além dos modelos apresentados acima, Iglesias propõe um ciclo de vida para o desenvolvimento de sistemas multi-agente, que é composto das seguintes fases (IGLESIAS, 1998):

- Fase de Conceitualização: obter uma descrição preliminar do problema a ser tratado. Pode ser utilizado o diagrama de Casos de Uso (UML, 1999).
- Fase de Análise: obter uma especificação de alguns requisitos do sistema, através do uso dos seguintes modelos: modelo do agente, modelo de tarefas, modelo de coordenação, modelo de conhecimento e modelo da organização.
- Fase de Projeto: consiste em determinar como os requisitos da fase de análise podem ser obtidos, mediante o desenvolvimento do modelo de projeto.
- Codificação e Teste: cada agente construído é codificado e testado.
- Integração: o sistema completo é validado.
- Manutenção: a filosofia de agentes facilita a manutenção do sistema, por ser de natureza modular.

O modelo de ciclo de vida proposto para o desenvolvimento destas tarefas na metodologia é o modelo de espiral (PRESSMAN, 1995). Porém, Iglesias sugere tanto para modelos pequenos como para o aprendizado da metodologia, o uso do ciclo de vida em cascata (PRESSMAN, 1995).

4.3 Proposta de Kendall

O trabalho de KENDLL (1999) propõe um modelo de papéis**.** Um papel define uma posição e um conjunto de responsabilidades dentro de um modelo de papéis, tendo colaboradores que são os outros papéis que interagem com este. A autora apresenta modelos de papéis como sendo uma extensão dos diagramas de colaboração utilizados em UML (FURLAN, 1998).

Segundo KENDLL (1999) existem alguns aspectos que precisam ser considerados em papéis de agentes, além dos aspectos representados em papéis de objetos:

- Responsabilidades - objetivos, obrigações e interdições.
- Especialidade - ontologia, inferência, conhecimento para resolução de problemas.
- Coordenação e negociação - protocolos, resolução de conflitos, permissões, conhecimento sobre a razão pela qual um papel está relacionado com outros papéis.
- Outros - recursos, aprendizado/adaptabilidade.

Nesta proposta, é utilizado o conceito de agregação de papéis, ou seja, uma vez tendo um modelo de papéis, pode-se construir novos modelos a partir da agregação de outros.

4.4 Agent UML

Proposta por ODELL (1999), sugere extensões para UML com o objetivo de apoiar o desenvolvimento de softwares baseados em agentes (ODELL, 2001). Sendo a AUML baseada na UML vale ressaltar o conceito de UML e os seus diagramas.

UML

É uma linguagem de modelagem que unifica os métodos de Grady Booch, James Rumbaugh e Ivar Jacobson. Além disso, foram acrescentados novos conceitos que não

são encontrados em outros métodos orientados a objeto.

A UML é a padronização da linguagem de desenvolvimento orientada a objetos para visualização, especificação, construção e documentação de sistemas. Pode ser usada com todos os tipos de processos, em todo o ciclo do desenvolvimento do software. Tendo passado por um processo de discussão para padronização pela OMG (Object Management Group), em novembro de 1997, tornou-se um padrão OMG (OMG, 2000).

Para descrever os vários aspectos de modelagem utilizando a UML, deve-se utilizar a notação definida pelos seus diagramas (UML, 199), (FURLAN, 1998), (FOWLER, 2000). Os diagramas propostos são:

- Caso de Uso: é utilizado para descrever e definir os requisitos funcionais de um sistema.
- Classe: é utilizado para denotar a estrutura estática de um sistema. O diagrama é considerado estático porque a estrutura descrita é válida em todo o ciclo de vida do sistema.
- Seqüência: mostra a colaboração dinâmica entre os vários objetos de um sistema. Pode-se observar a troca de mensagens dos objetos através do tempo.
- Colaboração: mostra de maneira semelhante, ao diagrama de seqüência, a colaboração dinâmica entre os objetos. E ainda, pode-se observar os relacionamentos entre os objetos.
- Estado: mostra todos os estados possíveis que objetos de uma determinada classe podem se encontrar e os eventos do sistema que provocam mudanças de estado.
- Atividade: tem como propósito mostrar os fluxos dirigidos por processamento interno, descrevendo as atividades desempenhadas em uma operação.
- Componente: este diagrama serve para descrever os componentes de *software* e suas dependências entre si, representando a estrutura do código gerado. Os componentes são tipicamente os arquivos implementados no ambiente de desenvolvimento.

- Utilização: mostra a arquitetura física do *hardware* e do *software* do sistema. Especificam-se também os componentes executáveis e objetos que são alocados para mostrar quais unidades de software são executados e em quais computadores são executados.

Após esta breve exposição da utilização dos diagramas da UML, serão apresentadas as principais propostas da AUML.

Segundo ODELL(1999), um agente é uma extensão do conceito de objeto, o que permite explorar extensões de linguagens de modelagem orientada a objetos com o objetivo de adaptá-las às características do paradigma orientado a agentes (ODELL, 1999). A partir disso, o autor propõe extensões a UML, descrevendo os requisitos mais comuns para modelagem de agentes e SMAs, criando assim a AUML(*Agent UML*) (BAUER, 1999).

A adoção da AUML está sendo estudada pela FIPA *(Foundation for Intelligent Physical Agents)* e OMG *(Object Management Group)* e teve como proposta inicial representar protocolos de interação de agentes (AIP), que é definido da seguinte forma:

"Um Protocolo de Interação de Agentes descreve o padrão de comunicação de como será feita a seqüência de mensagens entre agentes e o conteúdo destas mensagens" (ODELL, 2000).

Segundo Odell, a AUML é aplicada em três camadas (ODELL, 2001):

- Reutilização de Soluções: pacotes e modelos.
- Representação da Interação entre Agentes: diagrama de seqüência, diagrama de colaboração, diagrama de atividade e diagrama de estado.
- Representação do Processamento Interno: diagrama de atividade e diagrama de estado.

Reutilização de Soluções

Os padrões são idéias que podem ser consideradas úteis no contexto prático, podendo ser reutilizados em problemas semelhantes. São sugeridos por BAUER

(1999) e ODELL (2001) protocolos de interação de agentes que provêm soluções reutilizáveis, podendo ser usados por vários tipos de comunicações entre agentes.

Na UML, há duas formas para expressar soluções de protocolos reaproveitáveis, ou seja, através de pacotes (*packages*) e modelos (*templates*) (UML, 1999). Com isso, uma das propostas da AUML é a utilização de pacotes e modelos para uma reutilização dos protocolos de interação entre agentes.

Representação da Interação entre Agentes

Algumas das extensões sugeridas por ODELL (2001) podem ser visualizadas na figura 3.2, que descreve um formato básico para a comunicação de agentes, onde o retângulo representa um único agente ou um conjunto (papéis ou classes) de agentes. O retângulo é rotulado da seguinte forma:

nome_do_agente / papel :class

Esta sintaxe já é parte da UML, sendo que a UML indica o nome do objeto instanciado em vez do nome do agente. Ainda, a figura 4.2 mostra uma outra extensão da UML, por rotular as linhas horizontais com ações de comunicação (CA) de agente, enquanto que no paradigma orientado a objetos a linha é rotulada com nome de mensagens.

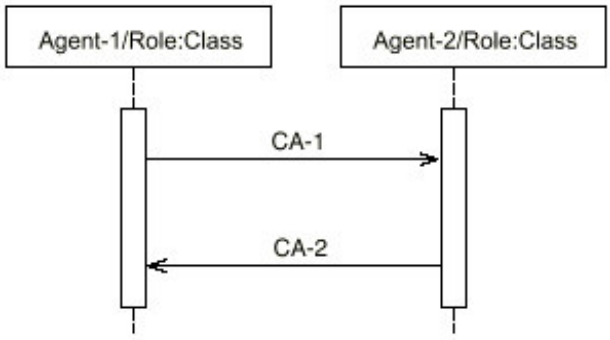

Fig. 4.2 Diagrama de Seqüência: Comunicação entre Agentes

A figura 4.3 mostra mais extensões para a UML, sugerindo representações para suporte de múltiplas *threads* de interação entre agentes. BAUER (2000) e ODELL (2001) reconhecem que *threads* existem em orientação a objetos, porém afirmam que

algumas extensões podem ser úteis em aplicações complexas. A figura 4.3 (a) indica que todos as *threads* CA-1 a CA-n serão enviados concorrentemente, representando o operador lógico E (*AND*). A figura 4.3 (b) inclui uma caixa de decisão vazia que representa o operador lógico OU (*OR*), sendo assim, zero ou mais CAs serão enviados. Por fim, a figura 4.3 (c) tem uma caixa de decisão que representa o operador lógico OU EXCLUSIVO (*XOR*), indicando que somente um CA será enviado.

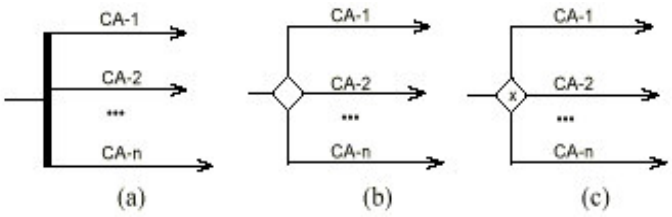

Fig. 4.3 Extensões para suporte a comunicação concorrente

Representação do Processamento Interno

O processamento interno do agente pode ser descrito utilizando diagramas de atividade ou diagramas de estado. Em (ODELL, 2001), foram sugeridas pequenas extensões para estes diagramas, tais como, rótulos com o nome do papel do agente associado a cada atividade.

Outras Considerações para AUML

São sugeridas pelos autores da AUML outras extensões (ODELL, 2001) para representar algumas peculiaridades de sistemas multi-agentes, como modificações no diagrama de utilização para modelar a mobilidade dos agentes. Os diagramas de seqüência e de classe podem ser utilizados também para mostrar a clonagem de agentes. A representação da mitose de um agente pode ser modelada através do diagrama de seqüência. E a modelagem da reprodução do agente pode ser mostrada com o diagrama de atividade.

5. UTILIZAÇÃO DE AGENTES PARA DEFINIÇÃO E ALTERAÇÃO DE BANCO DE DADOS HETEROGÊNEOS

Atualmente, com o surgimento de novas tecnologias é comum as empresas terem diferentes SGBDs espalhados pelos suas filiais, sendo praticamente impossível manter um profissional experiente para cada produto utilizado, em todos os locais que necessitem suporte técnico ou capacitar um profissional em todos os produtos utilizados.

Conseqüentemente, é comum hoje a busca por parte das empresas de auxílio para a realização da tarefa de administrar o ambiente de banco de dados moderno, que é heterogêneo e bastante complexo. No entanto, isto representa para a empresa um alto custo mensal.

É comum também a busca por ferramentas que através da Web auxiliam o DBA na administração, porém estas ferramentas são proprietárias e não funcionam com bancos de dados de modelos distintos.

Inserido neste contexto, este trabalho se propõe à criação de um ambiente o qual o DBA poderá configurar e utilizar para a definição e manutenção de banco de dados relacionais, objeto-relacional e orientado a objetos.

5.1 Definição e Manutenção de Banco de Dados

Segundo DATE (2000), o administrador de dados é a pessoa que toma as decisões estratégicas e de normas com relação aos dados da empresa e o administrador do banco de dados é a pessoa que fornece o suporte técnico necessário para implementar estas decisões, assim este é responsável pelo geral do sistema em um nível técnico. Normalmente, não há esta diferenciação, e as atribuições de um administrador de dados são sucumbidas pelo administrador de banco de dados. Neste caso, segue algumas tarefas do DBA:

- Definir o esquema conceitual.
- Definir o esquema interno.
- Monitorar o desempenho e responder a requisitos de mudança.

- Procedimentos de backup.
- Gerenciamento de espaço.
- Monitoramento do desempenho.
- Monitoramento de processos.
- Manutenção e reorganização do banco de dados.
- Dimensionamento de banco de dados.

Na definição do esquema de banco de dados pode-se citar como exemplo de operações, a criação de tabela, criação de visão, criação de índices, exclusão de tabelas, exclusão de visão, entre outras.

Na manutenção e reorganização do banco de dados, podemos citar todas as alterações em tabelas, tais como criação de um novo atributo, criação de chaves primárias e estrangeiras, exclusão de atributos, de índices de chaves, entre outros.

Cada modelo de dados tem uma linguagem na maioria das vezes padrão para a manipulação do banco de dados.

5.2 Agentes e XML no Ambiente Desenvolvido

A figura 5.1 mostra a arquitetura do ambiente desenvolvido, onde se pode observar o uso dos Agentes.

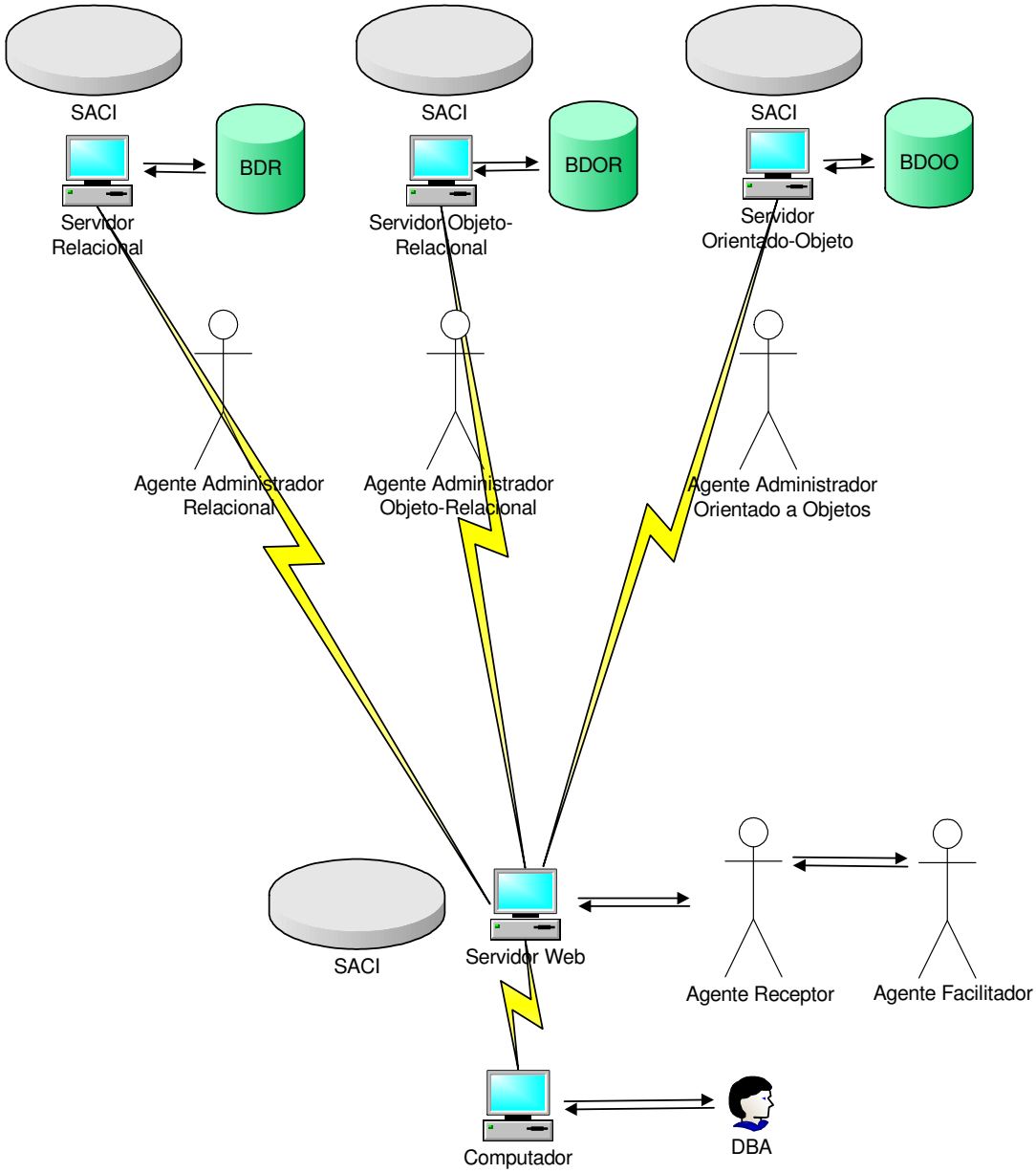

Fig. 5.1 Ambiente Desenvolvido

A execução do ambiente é iniciada quando o DBA solicita uma requisição, por exemplo, de incluir uma coluna em uma tabela, esta requisição dá início à criação do Agente Receptor no Servidor *Web,* este agente tem como principais funções gerar um arquivo XML, contendo a requisição do DBA e se comunicar com o Agente Facilitador solicitando um ou mais Agentes Administradores para realizarem a tarefa solicitada.

Portanto, o Agente Receptor gera o XML de acordo com a requisição do Administrador de Banco de Dados, sendo, que este pode optar, por exemplo, que a

alteração seja feita em um único banco de dados ou em vários bancos de dados que podem eventualmente ter modelos de dados distintos.

Após o Agente Receptor gerar o arquivo XML, este agente enviará uma mensagem ao Agente Facilitador, solicitando um ou mais Agentes Administradores que realizem esta tarefa, sendo que o número de Agentes Administradores será definido pelo Agente Receptor, dependendo da requisição do DBA. A partir disso, o Agente Facilitador enviará uma mensagem para um ou mais Agentes Administradores e estes por sua vez se moveram pela rede, de posse da requisição em XML, até a máquina destino onde devem executar sua missão. A tarefa de um Agente Administrador, após ter sido convocado pelo Agente Facilitador, é mover-se até o *host* onde as bases de dados estiverem armazenadas. Chegando, ao destino, o agente deve estabelecer uma conexão com o banco de dados e executar uma definição ou alteração neste banco de dados. Em seguida, o agente deve enviar uma mensagem ao Agente Receptor informando que a tarefa foi concluída.

Quanto à certeza do Agente Receptor em relação à conclusão da tarefa pelo Agente Administrador, se este agente não responder ao Agente Receptor em um determinado tempo, o Agente Receptor tem autonomia para solicitar ao Agente Facilitador um outro agente que realize a tarefa solicitada. Isto é possível porque o Agente Receptor terá a requisição do DBA em um arquivo XML e ainda porque se pode ter na sociedade mais de um agente com a mesma habilidade. Contudo, se após um certo número de tentativas o Agente Receptor não tiver a confirmação de que a tarefa foi executada, este por sua vez comunica o DBA de um possível problema na rede ou no servidor onde o banco de dados está armazenado.

5.3 Plataformas de Agentes

A linguagem Java viabilizou a concepção de diversos sistemas experimentais de agentes móveis. Numerosos sistemas estão atualmente em desenvolvimento e a maioria está disponível na Web. Neste trabalho é utilizada a plataforma SACI desenvolvida pela USP (SACI, 2000). Porém, existem diversas outras implementações, das quais foram selecionadas algumas que são abordadas brevemente a seguir.

- Odyssey - A General Magic Inc. criou o primeiro sistema de agentes móveis comercial, chamado Telescript. Entretanto, o Telescript teve pouco sucesso, pois era baseado em uma linguagem e arquitetura de rede proprietárias. A popularidade da Internet motivou a General Magic a reimplementar um sistema de agentes móveis chamado Odyssey, baseado em Java. Este sistema simplesmente mapeou os conceitos do Telescript em classes Java, permitindo os desenvolvedores a criar suas próprias aplicações.

- Aglets – Desenvolvido pela IBM, os são agentes autônomos baseados em Java. Eles possuem características básicas necessárias para implementar a mobilidade. Cada *aglet* tem um identificador global. Além disso, contém um itinerário que é responsável por armazenar as máquinas destinos onde este agente irá passar, além de armazenar as ações que os mesmos deverão executar em cada *host*. Para que um aglet possa executar e desempenhar suas tarefas em um *host*, o mesmo deve previamente estar rodando uma aplicação aglet, sendo generalizada como os ambientes de execução ou a infra-estrutura preparada para recepção do aglet, como citado anteriormente. Esta aplicação aglet, também denominada de servidor, e nomeado Thaithi

- Concórdia - Concebido pela Mitsubishi constitui-se de um *framework* para o desenvolvimento e gerenciamento de aplicações de agentes móveis. O Concórdia compreende múltiplos componentes, todos escritos em Java, na qual são combinados para prover um ambiente para aplicações distribuídas. Este sistema é simples e requer somente uma implementação padrão da máquina virtual Java. Seu ambiente é composto de um servidor e um conjunto de agentes.

- Voyager - Criado pela ObjectSpace, consiste em uma plataforma ORB (*Object Request Broker*) implementado em Java e com suporte a agentes. Voyager implementa os mecanismos tradicionais de troca de mensagens, somados à capacidade de objetos moverem-se através da rede como agentes.

- JATLite - desenvolvido na Universidade de Stanford, é um conjunto de programas escritos na linguagem Java, que possibilitam a criação de agentes de *software* que comunicam de uma forma pela Internet. O JATLite facilita

especialmente a construção de agentes que enviam e recebem mensagens usando a linguagem de comunicação KQML. Contudo, o JATLite não impõe nos agentes nenhuma teoria ou técnica de construção de agentes autônomos.

- SACI – desenvolvido na Universidade de São Paulo (USP), é uma plataforma que utiliza o KQML para a comunicação entre agentes. Possui o conceito de sociedade de agentes, onde os agentes "trabalham" juntos para resolver um "problema". Assim, na sociedade existe um agente especial da plataforma chamado de agente facilitador, que tem como principais funções auxiliar na comunicação entre os agentes e gerenciar os agentes, ou seja, o agente facilitador conhece a habilidade de cada agente de sua sociedade e onde encontrá-lo.

Os sistemas citados acima possuem muitas características em comum, como a utilização da máquina virtual Java padrão, mecanismos de serialização de objetos, e arquitetura baseada em um servidor. Contudo, os mecanismos de transporte de agentes e interação variam consideravelmente.

A plataforma utilizada neste trabalho foi o SACI da USP, devido as suas características de manipulação de agentes, além da facilidade de integração com a linguagem JSP (Java Server Pages) e por ser uma plataforma acadêmica tem todas suas funções liberadas.

6. IMPLEMENTAÇÃO

Para mostrar que é possível utilizar a tecnologia de agentes para administrar bancos de dados de modelos distintos separados fisicamente uns dos outros, foi desenvolvido um ambiente no qual o DBA pode executar tarefas de definição e alteração em banco de dados. Em seguida, são detalhados alguns aspectos mais relevantes da implementação.

6.1 Aspectos Gerais da Implementação

Na implementação foi desenvolvido um ambiente para a Internet, onde o Administrador de banco de dados pode realizar tarefas de definição e alteração de banco de dados de forma transparente, rápida e segura. O ambiente é totalmente configurável em termos de operações que o DBA pode realizar, sendo que este pode fazer todas as configurações no arquivo XML de configuração e nos arquivos XML de scripts de definição e alteração.

A plataforma utilizada provê o controle dos agentes, tais como mobilidade, comunicação entre agentes e segurança.

A implementação contém uma sociedade de agentes que se comunicam entre si, para resolverem as tarefas solicitadas pelo administrador de banco de dados.

6.1.1 Usuário da Aplicação

O ambiente desenvolvido é projetado para o uso de um Administrador de Banco de Dados (DBA), que tem como algumas de suas funções a definição e alteração de bases de dados, que é o assunto tratado neste trabalho.

Como o usuário da aplicação é um Administrador, têm conhecimentos para configurar a aplicação de acordo com as suas necessidades. Para tanto, basta que este atualize o arquivo de configuração em XML, descrevendo as operações que irá utilizar.

6.1.2 Acesso a Aplicação

O DBA terá acesso a aplicação através da Internet, em um *browser,* como mostrado na figura 6.1. A utilização da aplicação terá início com a validação de usuário e senha. Após isso, o administrador terá acesso à parte da aplicação onde este deve informar que tipo de operação é desejado, como é mostrado na figura 6.2, com a escolha da operação é apresentada ao DBA uma outra tela solicitando os parâmetros adequados para a realização da operação, conforme definido no arquivo de configuração XML da aplicação. Finalmente, após o preenchimento dos parâmetros a aplicação dará início a execução da tarefa solicitada pelo usuário.

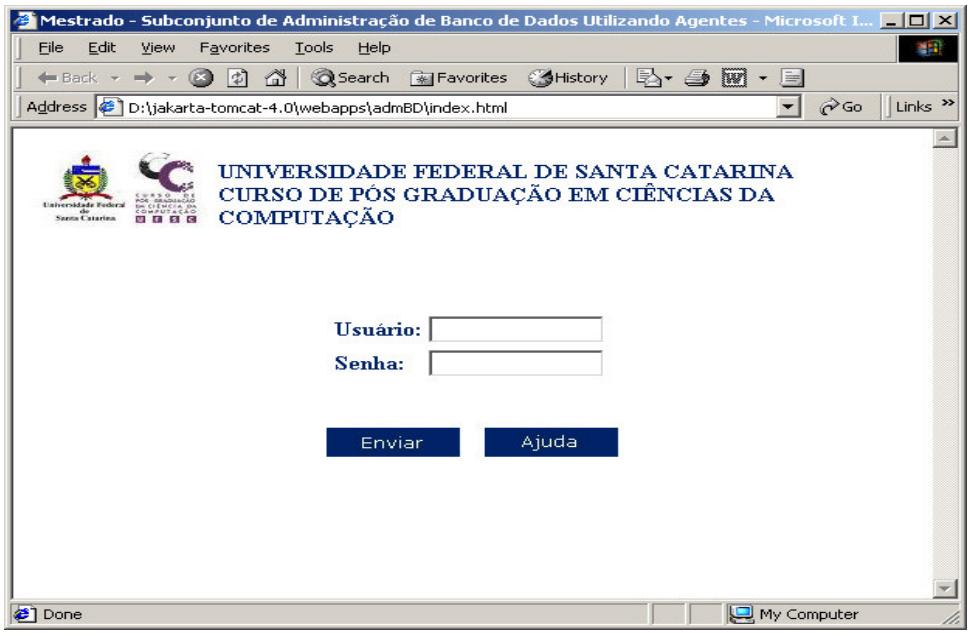

Fig 6.1 Tela inicial da aplicação

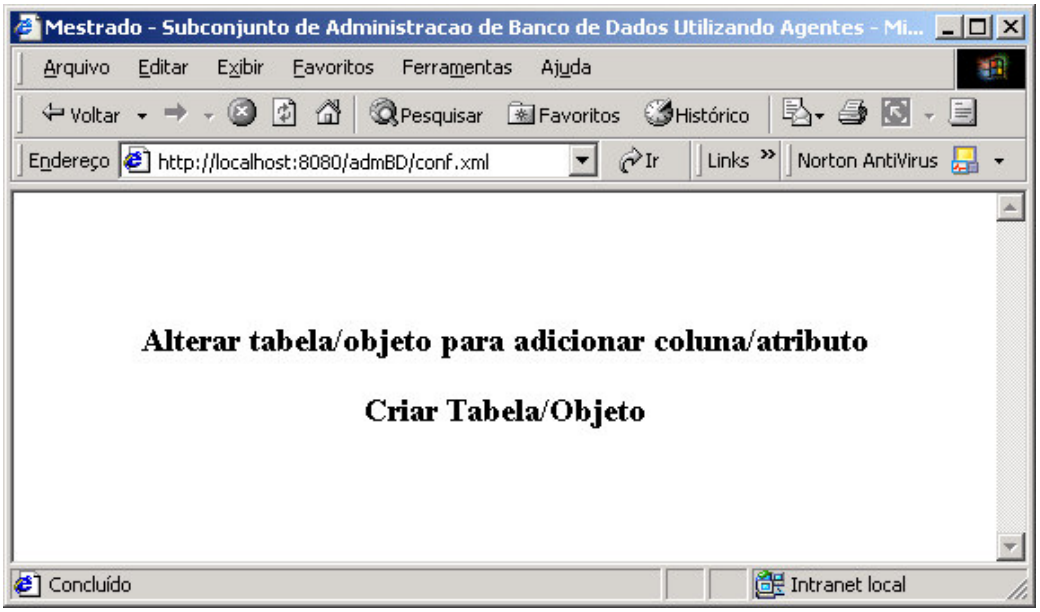

Fig. 6.2 Tela com as operações disponíveis na aplicação

Para acrescentar uma nova operação no ambiente, basta que o administrador de banco de dados edite o arquivo XML, mostrado abaixo.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="menu.xsl"?>
<operacoes>
        <operacao nome="alter_add_atributo" descricao="Alterar tabela/objeto para
        adicionar coluna/atributo" atributos="1">
                        <banco>Nome do Banco de Dados</banco>
                        <tabela>Nome da Tabela/ Objeto</tabela>
                        <atributo>Nome do Atributo</atributo>
                        <tipo>Tipo do Atributo</tipo>
        </operacao>
        <operacao nome="create_table" descricao="Criar Tabela/Objeto"
        atributos="0">
                        <banco>Nome do Banco de Dados</banco>
                        <tabela>Nome da Tabela/ Objeto</tabela>
                        <atributo>Nome do Atributo</atributo>
                        <tipo>Tipo do Atributo</tipo>
        </operacao>
</operacoes>
```

6.1.3 Plataforma de Execução

A plataforma utilizada é a responsável pela mobilidade e segurança dos agentes. Foram realizados estudos sobre o *Aglets* da IBM, o *Voyager* da *Object Space* e o SACI da Universidade de São Paulo (USP), e a opção foi pelo SACI pelo fato deste ser um produto de distribuição gratuita, ter uma boa documentação, ser compatível com o JDK 1.4, entre outras características. Sobre o Aglets, este ainda, só funciona até JDK 1.1.8. O *Voyager* é um produto utilizado comercialmente, porém, não é de distribuição livre, mas disponibiliza uma versão para testes.

O SACI utiliza o conceito de sociedade de agentes, assim uma aplicação pode ter várias sociedades de agentes, onde cada agente tem pelo menos uma habilidade dentro de sua sociedade. Assim como em uma sociedade humana, os agentes podem trabalhar juntos, cada um com sua habilidade para resolver problemas.

Existe, ainda o conceito de agente facilitador, que é um agente especial da plataforma que controla toda a sociedade e a comunicação entre os agentes. Todos os agentes de uma sociedade pedem autorização ao agente facilitador para entrar e sair da sociedade e também informam sua habilidade. Assim, um agente pode questionar o Agente Facilitador sobre os serviços oferecidos.

O Saci foi desenvolvido com base na especificação KQML, sendo assim os agentes utilizam KQML para se comunicar. Há funções para compor, enviar e receber mensagens KQML. Entretanto, dentro do conteúdo da mensagem KQML pode-se utilizar qualquer linguagem.

A figura 6.3 mostra uma janela DOS, onde é iniciado o SACI, com o seguinte comando no *prompt* do DOS:

C:\Saci\bin>saci.bat

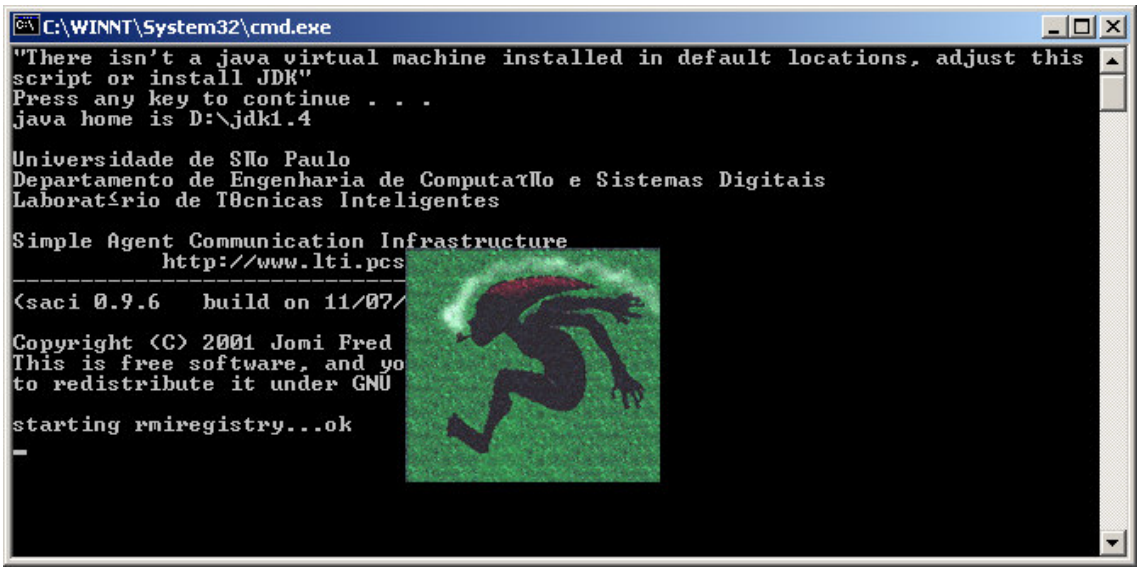

Fig. 6.3 Tela inicial da plataforma SACI

A plataforma depois de inicializada abre uma janela com o Servidor SACI, como mostrado na figura 6.4.

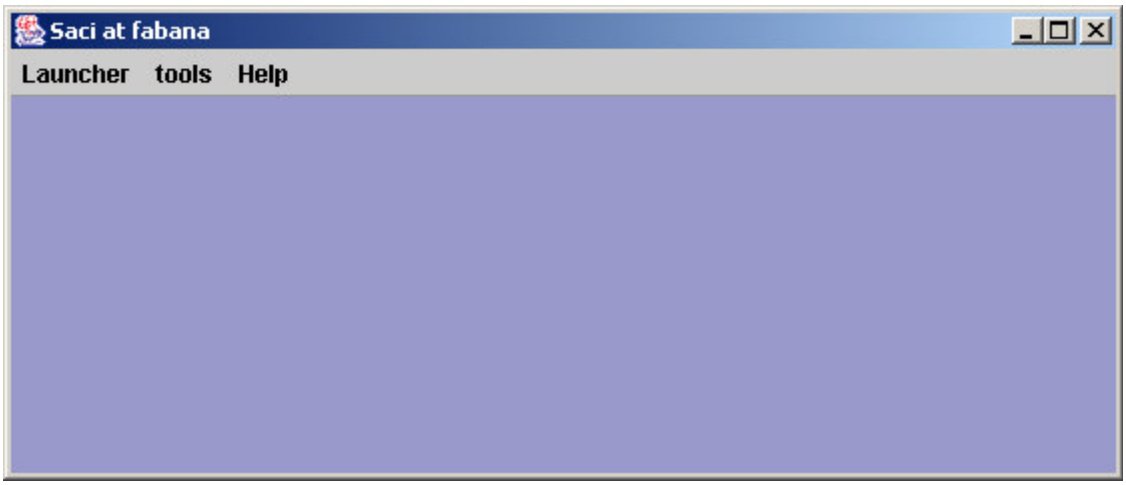

Fig. 6.4 Tela do Servidor SACI

A aplicação *Web* desenvolvida será composta de uma sociedade de agentes, onde cada agente pertencente possui pelo menos uma habilidade. Assim, a sociedade desta aplicação contém um Agente Facilitador que auxiliará na comunicação entre os agentes, um Agente Receptor que terá como principal função receber a requisição do DBA e

gerar um arquivo XML, e três Agentes Administradores sendo, um relacional, um objeto relacional e um orientado a objetos.

Após a validação de acesso do Administrador será criada a sociedade de agentes e será inicializado cada agente. Na inicialização os agentes pedem autorização para o Agente Facilitador para entrar na sociedade e divulgam as suas habilidades nesta. Com isso, o agente facilitador conhece cada agente pertencente a sua sociedade, o papel de cada um e onde encontrá-lo.

6.2 Classificação dos Agentes da Aplicação

Neste trabalho, foi desenvolvida uma sociedade composta de cinco agentes, sendo eles: Agente Facilitador, Agente Receptor, Agente Administrador Relacional, Agente Administrador Objeto Relacional e Agente Administrador Orientado a Objetos.

6.2.1 Agente Facilitador

O Agente Facilitador é um agente especial da plataforma que tem como principal função coordenar a sociedade e a comunicação entre os agentes.

Este agente conhece todos os agentes de sua sociedade, a habilidade de cada um e onde pode encontra-los. Ele também é responsável por autorizar a entrada e saída dos agentes em sua sociedade.

6.2.2 Agente Receptor

O Agente Receptor é um agente classificado como estático e tem como principais funções gerar um arquivo XML, contendo a requisição do DBA e se comunicar com o Agente Facilitador para solicitar um agente que realize a tarefa requerida pelo DBA. Este agente também informa ao DBA as prováveis falhas nos servidores de banco de dados.

O envio e recebimento de mensagens no SACI é através de um componente denominado *MBox* que serve de interface entre o agente e a sociedade, sua finalidade é

tornar transparente o envio e o recebimento de mensagens. Este componente possui funções que encapsulam a composição de mensagens KQML, o envio assíncrono de mensagens, o recebimento de mensagens, o anúncio e a consulta de habilidades dos agentes.

Quando o Agente Receptor recebe a requisição do DBA, ele envia uma mensagem ao Agente Facilitador, questionando-o de um ou mais agentes capazes, por exemplo, de adicionar um atributo em uma tabela ou objeto.

```
mbox.consultYP (”ask-one ” , null , ”XML” , ”Adicionar_Atributo” );
```

Para compor uma mensagem KQML, deve-se utilizar a classe Message, como mostrado no exemplo abaixo.

```
Message r = new Message (”(tell)”);
r.put( ”receiver ”, m.ge t (”sender”)) ;
r.put(”in-reply-to”, m.get(”reply-with”));
r.put (”content”, m.get(”content”));
mbox.sendMsg (r);
```

6.2.3 Agentes Administradores

Os Agentes administradores são classificados como móveis. A função de um Agente Administrador após ter sido convocado pelo Agente Facilitador é mover-se até o *host* onde a base de dados estiver armazenada. Chegando ao destino o agente deve estabelecer uma conexão com o banco de dados e executar uma definição ou alteração neste banco de dados, em seguida, o agente deve enviar uma mensagem ao Agente Receptor informando que a tarefa foi concluída, se tiver mais algum banco de dados em seu itinerário este continua seu trabalho se movendo de um *host* a outro.

Cada agente administrador tem um arquivo de configuração XML, com os *scripts* relacionados ao seu modelo de dados, por exemplo, o Agente Administrador Relacional tem acesso a um arquivo XML com os SQL para definir e alterar Banco de Dados Relacional.

Assim, baseado na configuração do arquivo XML o Agente Administrador será capaz de realizar suas tarefas.

Então, se o administrador de banco de dados configurou o arquivo de configurações com uma nova operação, ele deve também editar cada arquivo de *script* do banco de dados relacional, do objeto-relacional e do orientado a objetos, isto porque cada modelo de dados utiliza um tipo diferente de linguagem de definição de esquema de banco de dados e manutenção.

O trecho de código a seguir demonstra a configuração do Agente Facilitador e a criação do Agente Administrador Relacional.

```
Config c = new Config();
c.set("facilitator.host", "local");
saci.Agent agRelacional = new AgAdmBDR();
```

Logo abaixo, pode se observar um pedaço de código, onde o Agente Administrador Relacional pede permissão ao Agente Facilitador para entrar na sociedade. Após entrar na sociedade o Agente é inicializado e começa a executar.

```
if (agRelacional.enterSoc("AgAdmBDR",c)) {
        agRelacional.initAg(null);
        agRelacional.run();
}
```

O código abaixo mostra como o Agente Administrador Relacional anuncia sua habilidade dentro da sociedade. Pode-se notar também que a linguagem utilizada dentro do *content* do KQML será o XML.

```
public void initAg(String[] args) {
 try {
        mbox.advertise("ask-one", "XML", "administracao", "administraBDR");
 } catch (Exception e) {
        System.err.println("Error starting agent:"+e);
 }
}
```

6.3 Base de Dados

Foram utilizados neste trabalho sistemas gerenciadores de banco de dados para representar o modelo relacional, o modelo objeto-relacional e o modelo orientado a objetos.

6.3.1 Relacional

Para definição e alteração no banco de dados relacional, o DBA deve definir o script da operação requisitada no arquivo de configuração relacional. Uma vez que a operação estiver definida, o agente administrador relacional utilizará este *script* a cada nova solicitação. A linguagem a ser utilizada dentro do arquivo XML relacional deve ser o SQL92 padrão.

O código abaixo mostra um pedaço de um arquivo de *script* XML para o banco de dados relacional.

```
<?xml version="1.0"?>
<operacoes>
        <operacao nome="alter_add_atributo" descricao="Alterar tabela/objeto   para
        adicionar coluna/atributo">
                        <comando nome="ALTER TABLE">
                                <tabela>nm_tabela</tabela>
                        </comando>
                        <comando nome="ADD">
                                <atributo>nm_atributo</atributo>
                                <tipo>nm_tipo</tipo>
                        </comando>
        </operacao>
</operacoes>
```

O agente administrador relacional utiliza um driver proprietário do SQL Server para abrir a conexão com o banco de dados, como mostrado no trecho de código abaixo.

```
Class.forName(com.microsoft.jdbc.sqlserver.SQLServerDriver);
con = DriverManager.getConnection("jdbc:microsoft:sqlserver://local:1433;
                databasename="+ banco + ";user="+ user +";password="+ pass);
```

6.3.2 Objeto Relacional

No banco de dados objeto-relacional deve-se proceder de forma semelhante ao banco de dados relacional, com a diferença da linguagem utilizada que deve ser o SQL3.

O agente administrador objeto-relacional utiliza um driver proprietário do Oracle para abrir a conexão com o banco de dados, como mostrado no trecho de código abaixo.

```
Class.forName(oracle.jdbc.driver.OracleDriver);
con = DriverManager.getConnection("jdbc:oracle:thin:@local:1433:
        databasename="+ banco + ";user="+ user +";password="+ pass);
```

6.3.3 Orientado a Objetos

Os banco de dados orientados a objetos normalmente utilizam uma linguagem de programação para a definição e alteração. O SGBD utilizado usa a linguagem de programação Java para definição de esquema de banco de dados e também para a alteração do banco de dados.

O agente administrador orientado a objeto utiliza um driver proprietário do *Fast Object* para abrir a conexão com o banco de dados, como mostrado no trecho de código abaixo.

```
banco = new Database();
banco.open("poet://Local/Base", Database.OPEN_READ_WRITE);
```

6.4 Linguagens Utilizadas

A linguagem utilizada para desenvolver a aplicação foi Java Server Page (JSP), por todas as vantagens que a linguagem Java oferece (DEITEL, 2001). Porém para a comunicação dos agentes foram utilizados o KQML e o XML, além de se utilizar o XML na configuração do ambiente.

Para a definição e alteração do banco de dados relacional foi utilizado o SQL padrão, para o banco de dados objeto relacional foi utilizado o SQL3 e para o orientado a objetos foi usada a linguagem Java.

6.5 Ferramentas Utilizadas

Na implementação do trabalho foram utilizados três sistemas gerenciadores de banco de dados de modelos de dados distintos. Para representar o modelo relacional foi escolhido o SQL Server 2000 da Microsoft. Para representar o modelo objeto-relacional foi utilizado o Oracle 9i da Oracle. O modelo orientado a objeto foi representado pelo Fast Object da Poet.

A implementação dos agentes foi suportada pela plataforma SACI da USP.

7. CONCLUSÃO

Neste trabalho foi aplicado a tecnologia de agentes com o objetivo de realizar tarefas de definição e alteração de bases de dados de modelos heterogêneo. A utilização dos agentes vem a facilitar a manutenção dos bancos de dados para o Administrador de Banco de Dados (DBA). A transparência das ações do agente na rede, bem como a comunicação com os outros agentes, permite a manutenção de diversas bases de dados, o que torna o ambiente propício, principalmente, para um administrador de banco de dados de uma grande empresa.

A utilização de agentes em conjunto com a Internet para administrar banco de dados proporciona uma redução nos custos da organização, que não precisará arcar com despesas oriundas do transporte de pessoal qualificado, assim como, não terá a necessidade de utilizar diferentes ferramentas proprietárias para cada banco de dados existente.

Além de trazer estes benefícios, a utilização da mobilidade de código possibilita uma maior consistência no banco de dados, uma vez que os agentes se deslocam para as máquinas e executam suas tarefas localmente. Desta forma, as solicitações do DBA não ficam trafegando na rede, mas são armazenadas no formato XML no compartimento dos agentes móveis, fornecendo mais segurança.

A utilização dos agentes, em relação à arquitetura cliente-servidor, permite que as máquinas distribuídas na rede sejam cliente e ao mesmo tempo fornecedoras de serviços, balanceando assim, a carga dos servidores de dados. Além disso, se alguma máquina que contenha um banco de dados não esteja ativa, o agente prossegue o seu itinerário executando suas tarefas nas demais máquinas. Outra vantagem é a existência no ambiente, de um agente responsável por comunicar o DBA de possíveis falhas em algum *host* do itinerário dos agentes móveis.

A utilização de XML na configuração do ambiente desenvolvido proporcionou uma maior flexibilidade em termos de funções e bases de dados administradas pelo DBA. Assim, toda vez que o DBA precisar de uma nova função que ainda não foi configurada no ambiente, este simplesmente edita um arquivo XML e faz a nova configuração.

Finalmente, propõe-se como trabalho futuro, a avaliação de outras plataformas ou a extensão da plataforma utilizada, para que os agentes possam comunicar-se utilizando XML em vez de KQML, como foi implementado nesta dissertação.

A implementação integrada da definição e alteração de banco de dados, como apresentado no modelo deste trabalho. Propõe-se também a utilização de agentes inteligentes para administrar bancos de dados.

8. REFERÊNCIAS BIBLIOGRÁFICAS

AMANDI, A. A. **Programação de Agentes Orientada a Objetos**. Porto Alegre: Tese de Doutorado do CPGCC da UFRGS, 1997.

ASBURY, Stephen; RICART, Manuel Alberto. **ASP 3, Guia do Desenvolvedor**. Editora Bekeley, 2001.

BAUER, Bernhard. **Extending UML for the Specification of Interaction Protocols**, submitted for the 6th Call for Proposal of FIPA, 1999. Disponível em *http://www.auml.org.*

BAUER, Bernhard. **Extending UML for the Specification of Interaction Protocols**, submitted to ICMAS 2000, 2000. Disponível em http://www.auml.org.

BAUER, Bernhard. **UML Class Diagrams Revisited in the Context of Agent-Based Systems**, 2001. Disponível em http://www.auml.org.

BIGUS, Jennifer; BIGUS, Joseph**; Constructing Intelligent Agents with Java**; Willey Computer Publishing, 1998

BOBROWSKI, Steve. **Oracle 8i for Windows NT Starter Kit**, Oracle Press, 2000

BRENNER, Walter; ZARNEKOW, Rüdiger; WITTIG, Hartmut. **Intelligent Software Agents, Springer**, Germany, 1998.

BROWN, Simon; BURDICK, Robert; et al. **Professional JSP**, Wrox Press, 2001.

CLARO, Daniela B. **Integração de Bases de Dados utilizando a Mobilidade de Código**. Dissertação de mestrado, CPGCC/UFSC, Florianópolis, setembro de 2000.

COLEMAN, Derek. **Object Oriented Development**: The Fusion Method. Prentice Hall, 1994.

DATE, C.J. **Uma Introdução a Sistemas de Banco de Dados**. Editora Edgard Blücher LTDA, 1995.

DATE, C.J. **Introdução a Sistemas de Banco de Dados**, Editora Campus LTDA, 2000.

DEITEL, H.M; DEITEL, P.J. **Java**, Como Programar.Bookman, Porto Alegre, 2001.

FININ, Tim; LABROU, Yannis. **A Proposal for a new KQML Specification**. University of Maryland Beltimore Count(UMBC), 1997.

FININ, Tim; FRITZON, Rich; MCKAY, Don; MCENTIRE, Robin. **KQML - A Language and Protocol for Knowledge and Information Exchange**. Computer Science Department. Universit of Maryland Baltimore, 1994.

FININ, Tim; WEBER, Jay. **Specification of the KQML Agent-Communication Language**. The DARPA Knowledge Sharing Initiative, 1993.

FORTIER, Paul J. **SQL3 Implementing the Object-Relational Database**, McGraw-Hill, 1999.

FOWLER, Martin; SCOTT, Kendall. **UML Essencial**, Bookman, São Paulo, 2000.

FUNG, Khun Yee. **XSLT interagindo com XML e HTML**, Editora Ciência Moderna, Rio de Janeiro, 2001.

FURLAN, José D. **Modelagem de Objetos através da UML**, Makron Books, São Paulo, Brasil, 1998.

GARCIA, Alessandro F; SILVA, Viviane T; LUCENA, Carlos J. P; et al. **An Aspect Based Approach for Developing Multi-Agent Object-Oriented Systems**. Em: XV Simpósio Brasileiro de Engenharia de Software. Rio de Janeiro, Brasil, 2001. p.177.

GUTTA, Rajendra. **Oracle DBA, Scripts de Automação**. Editora Campus Ltda, Rio de Janeiro, 2002.

HÓLM, Bjarki et al. **Oracle 9i Java Programando**, Editora Alta Books Ltda, Rio de Janeiro, 2002.

HÜBNER, Jomi F; SICHMAN, Jaime S; BECERRA, Jorge R.**Aplicação da Arquitetura de Objetos Distribuídos em Sistemas Multi-Agentes**, 2000.

IGLESIAS, Carlos A; GARIJO, M.; GONZALEZ, J. C.; VELASCO, J. R. **Analysis and design of multiagent systems using MAS-CommonKADS**, 1998.

IGLESIAS, Carlos A. **Definición de uma metodología para el desarrollo de sistemas multiagente**, Tesis Doctoral,1998.

INGRES, **Data Management & Application Development**. Disponível em http://www3.ca.com/Solutions/Product.asp?ID=1013

JASMINE, **Jasmine Solution**. Disponível em http://www.cavo.com/solutions/jasmine.asp

JENNINGS, Nicholas R.; FARATIN, P.; JOHNSON, M. J.; O'BRIEN, P.; WIEGAND,M. E. **Using intelligent agents to manage business processes**, 1996.

KATTAN, Rami; WILKINSON, Jeff. **TableEditorR**. Disponível em http://www.2enetworx.com/dev/projects/tableeditor.asp

KENDALL, Elizabeth **A. Role Modelling for Agent System Analysis, Design, and Implementation**, 1999.

KHOSHAFIAN, Setrag. **Banco de Dados Orientado a Objeto**, IBPI Press, 1994.

LONEY, Kevin; THERIAULT, Marlene. **Oracle 8i, O Manual do DBA**, Editora Campus Ltda, 2000.

MACÊDO, José Antonio Fernandes. **Um Estudo de SGBDs Baseados em Agentes**, Pontifícia Universidade Católica do Rio de Janeiro, 2000.

MARTIN, Didier; BIRBECK, Mark, et al. **Profesional XML**, Editora Ciencia Moderna, 2001.

MATTOSO, Marta L. Q. **Mini-Curso sobre Banco de Dados**, UFRJ, 1998.

MATTOSO, Marta L. Q.; MAURO, Renato C. **Modelagem e implementação de Banco de Dados Orientado a Objetos baseados em Padrões**, UFRJ, 1999.

MELLO, Ronaldo dos S. **Avaliação de Sistemas de Gerência de Banco de Dados Orientado a Objetos**, UFSC, 1997.

MOURA, Ana M. de C. **ODMG2.0**: Modelo e Linguagem: ODL/OQL, IME, 2000.

MOURA, Ana M. de C. **SGBD Relacional Objeto**, IME, 2000.

MUNDO OO. http://www.mundooo.com.br

NASSU, Eugenio A.; SETZER, Valdemar W. **Banco de Dados Orientados a Objetos**, Editora Edgard Blücher LTDA, 1999.

ODMG, http://www.odmg.org/index.htm, 2000.

ODELL, James. **Objects and agents - how do they differ?** (draft 2.2) Disponível em http://www.jamesodell.com., 1999

ODELL, James; BAUER, Bernhard; MULLER, Jorg P. Agent UML: **A Formalism for Specifying Multiagent Interaction**, 2001. Disponível em http://www.auml.org.

ODELL, James; PARUNAK, H.Van Dyke; BAUER, Bernhard. **Extending UML for Agents**. Disponível em http://www.auml.org., 2000

OLIVEIRA, F. M. **Inteligência Artificial Distribuída**. In: IV Escola Regional de Informática. Canoas, Brasil: Sociedade Brasileira de Computação, 1996. p.239.

OMG Unified Modeling Language Specification. Disponível em, http://www.omg.org/index.htm, 2000.

ORACLE Corporation, http://www.oracle.com/index.html, 2000.

POET Software, http://www.poet.com/index.html, 1999.

PRESSMAN, Roger S. **Engenharia de Software**, Makron Books, São Paulo, 1995.

RUSSEL, S; NORVIG, P. **Artificial Intelligence**: A Modern Approach; New Jersey, Pretice Hall, 1995.

SALGADO, Ana C.; FONSECA, Fernando. **Banco de Dados Objeto-Relacional**, UFPE, 2000.

SACI Software, http://www.lti.pcs.usp.br/saci/, 2002.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistemas de Banco de Dados**, Makron Books, 1999.

SHAPIRO, Jeffrey R., SQL SERVER 2000: Completo e Total.
Editora Makron Books, 2002.

STONEBRAKER, Michael. **Object- Relational DBMSs** - The Next Great Wave, Morgan Kauffmann Pub., 1996.

UML 1.3 Notation Guide. Disponível por WWW em http://www.rational.com/uml (jun 1999).

VOYAGER, Recursion Software: http://www.recursionsw.com/
Disponível em 17 de junho de 2003.

WOOLDRIDGE, Michael; JENNINGS, Nicholas R.; et. al. **The Gaia Methodology for Agent-Oriented Analysis and Design**, 2000.