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

Daniela Barreiro Claro

INTEGRAÇÃO DE BASES DE DADOS UTILIZANDO A MOBILIDADE DO CÓDIGO

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

Profº Dr. João Bosco Mangueira Sobral

Florianópolis, Setembro / 2000

INTEGRAÇÃO DE BASES DE DADOS UTILIZANDO A MOBILIDADE DO CÓDIGO

Daniela Barreiro Claro

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 da Universidade Federal de Santa Catarina.

Prof. João Bosco Mangueira Sobral, Dr.
(Orientador)

Prof. Fernando Álvaro Ostuni Gautier, Dr.

Banca Examinadora

Prof. Jorge Lopes de Souza Leão, Dr. Ing.

Prof. Luiz Carlos Zancanella, Dr.

Prof. Jorge Muniz Barreto, D.Sc.A

Prof. Ricardo Felipe Custódio, D.Sc.

"...e aprendi que se depende sempre de tanta, muita, diferente gente. Toda pessoa sempre é as marcas das lições diárias de outras tantas pessoas. E é tão bonito quando a gente entende que a gente é tanta gente onde quer que a gente vá. É tão bonito quando a gente sente que nunca está sozinho por mais que pense estar..."

Gonzaguinha

Agradecimento

Produzir este trabalho exigiu bastante tempo e esforço de muitas pessoas além de mim, por isso gostaria de começar agradecendo a minha família, em especial a D. Olga Claro e D. Olga Barreiro, minha mãe e minha avó, que me deram força e confiança nos momentos de desânimo e descrença, e me exaltaram nos momentos de sucesso absoluto, permitindo que neste período eu subisse mais um degrau na minha vida, além, é claro, dos telefonemas para recomposição de energias e encorajamento; a meu pai, pela coragem e esforço para trazer meu TheBest, e pelos telefonemas, quando eu realmente precisava falar com alguém; aos meus amigos baianos, que mesmo distante, marcaram sempre presença nos meus pensamentos, nas cartas e principalmente nos e-mails.

Não poderia deixar de agradecer aos meus mais novos amigos de Santa Catarina, principalmente ao Clube da Luluzinha, onde os incentivos, os planos futuros, o crescimento e o aprendizado foram fatos intermitentes nas nossas reuniões.

E gostaria de agradecer principalmente ao Profº Dr. João Bosco, meu orientador, não só pela oportunidade de compartilhar seus conhecimentos e crescer diante da sua orientação, mas pelos ensinamentos de vida, pela sua amizade, pelo seu acompanhamento.

Enfim, gostaria de agradecer a todas as pessoas que direta ou indiretamente contribuíram para a realização deste trabalho.

Resumo

Atualmente, diversas tecnologias de armazenamento de informações estão sendo desenvolvidas e utilizadas pelas organizações. Porém, os bancos de dados existentes nestas empresas não podem ser desprezados a curto prazo e, então, surge a necessidade de integrar diversas bases de dados utilizando a mobilidade do código.

A utilização de agentes móveis permite que a integração dessas bases de dados seja transparente para o usuário, visto que o agente é o responsável pela pesquisa de informações armazenadas nesses bancos. Além disso, os agentes móveis também são responsáveis pela comunicação com outros agentes, particionando as tarefas entre agentes estáticos com a finalidade de acesso aos dados em uma determinada base.

Assim, a utilização da mobilidade do código para a integração de bases de dados permite facilitar, principalmente, a recuperação das informações .

Abstract

Nowdays, different technologies of storing information are being developed and used by business corporations. However, databases already in use can not be prompt thrown away off in a short time. So, there is a growing necessity of integrating different databases using mobile code.

The use of mobile agent allows the database integration to be transparent for users, because agents are responsible for searching information. In additional, mobile agent can be also responsible for the communication with other agents, partitioning the tasks with static agents that have the main goal to access the information in a database

Thus, the use of mobile code for databases integration allow facilities specially in retrieval information.

Sumário

Índice de Figuras

1. Introdução

A evolução da Informática nos últimos anos, permitiu um crescimento tecnológico em relação ao *hardware* e *software*. Com isso, o desenvolvimento de novas aplicações, as constantes mutações de armazenamento dos dados, a diversidade dos ambientes utilizados foram os precursores dos requisitos básicos para a flexibilidade de integração. Diante deste contexto, observa-se, atualmente, a necessidade das organizações em integrar ambientes distintos, assim como armazenamentos de informações que divergem por seus tipos, com o intuito de prover uma maior interoperabilidade dentro e entre as organizações.

Com o advento da globalização e o surgimento da Internet, distribuindo assim, fisicamente, as corporações, as mesmas necessitam cooperar mutuamente para que possam atingir uma determinada demanda do mercado. Além disso, a grande rede é responsável por conectar diferentes tipos de computadores, redes distintas, além de garantir a transparência de localização em se tratando das informações pertinentes. Porém, a grande quantidade de usuários e informações que trafegam por esta rede estão culminando com uma utilização saturada da largura de banda. O problema ainda se agrava em se tratando da obtenção e o envio de informações que compõem os acessos às bases de dados remotas. Presenciando estes fatos, surge a tecnologia dos agentes móveis, que trata as questões relativas à integridade dos dados trafegados, visto que os mesmos serão tratados localmente em cada estação, assim como a questão da distribuição das bases de dados.

O problema da integração de bases de dados tem como primeira proposta de solução o trabalho realizado pela OMG (Object Management Group) dentro do contexto do CORBA (Common Object Request Broker Architecture), através do seu serviço de objetos persistentes (POS). O POS permite que o estado de um objeto seja salvo em um dispositivo de armazenamento persistente e restaurado quando for necessário. A idéia é criar uma implementação aberta que satisfaça aos requisitos de objetos de diferentes dispositivos de armazenamento persistente, assim podendo se construir uma única interface para múltiplas bases de dados(arquivos, bases de dados relacional, orientadas a objetos, e outras). Entretanto, a padronização vigente se utiliza do conceito de um elemento mediador que roteia as chamadas às bases de dados para um determinado banco de dados no âmbito do paradigma Cliente/Servidor[ORFA97].

Em uma organização, a crescente tecnologia para armazenamento de informações requer muitos gastos com treinamento de pessoal, investimento em software e hardware para suportarem as novas abordagens, e principalmente tempo do funcionário que será o responsável por realizar a migração dos sistemas antigos para novas arquiteturas. Além disso, o armazenamento e a obtenção coesa dos dados, outrora armazenados, permite ganhos em eficiência e integridade, garantindo a satisfação do cliente. Assim sendo, a flexibilidade no armazenamento de informações e a utilização de agentes móveis permitem à empresa galgar a caminho do seu diferencial de mercado. Diante desta visão comercial, a necessidade de integrar bases de dados distintas utilizando a mobilidade do código torna-se cada vez mais evidente. E é daí que surge a idéia de implementar um ambiente para atender a esta demanda do mercado.

Objetivamente, este trabalho busca responder a questões como, por exemplo, se é possível integrar em um único ambiente distintas bases de dados, como banco de dados relacional, objeto-relacional e orientado a objeto; configurar um ambiente heterogêneo como este e garantir a perenidade do mesmo de forma semelhante a um acesso comum a uma única base de dados; desenvolver uma interface gráfica responsável por disparar o agente de forma fácil e transparente para o usuário; garantir, caso haja falha na conexão da rede, que o agente irá concluir com êxito as tarefas a ele designadas; diminuir a utilização da rede, no momento que o agente se deslocará para a máquina onde se

encontra as informações, obtendo os dados localmente; obter, em uma única conexão, as informações requisitadas pelo usuário.

Enfim, este trabalho busca elucidar características de integração de bases de dados que se adeqüem à utilização de agentes móveis.

1.1-Trabalhos Existentes

Ao longo do período de pesquisa, alguns trabalhos foram selecionados por estarem relacionados com o tema proposto neste documento.

O artigo Compartilhamento de Módulos de Bases de Dados Heterogêneas através de Objetos Integradores[BUSH99], publicado no SBBD'99, autoria de Gisele Busichia e João Eduardo Ferreira, propõem a utilização de objetos que permitem a integração de módulos de bases de dados que estejam situados em localidades distintas, porém utilizando para isso SGBD's relacionais e realizando o acesso aos módulos usando as *stored procedures*. Além disso, a utilização do objeto integrador pressupõe o conceito de desenvolvimento de sistemas e acesso aos servidores de dados em n-camadas, ultrapassando a limitação imposta por algumas linguagens de programação que propõem esta integração no contexto do paradigma Cliente/Servidor.

Outro artigo ressaltado é o *Mobile Agent Technology Enabling The Virtual Enterprise: A Pattern for Database Query*[PAPA99], cujos autores são *Todd Papaioannou* e *John Edwards*, publicado no *Agent Based Manufacturing Workshop*, que traz como tema principal o desenvolvimento de um agente, utilizando a plataforma fornecida pela IBM, os Aglets, e como linguagem de desenvolvimento o Java, da Sun Microsystem. Este artigo propõe uma busca de informações utilizando banco de dados relacional, fazendo uso do JDBC (Java Databse Connection), uma API para conexão com o banco de dados utilizando o Java, além de propor uma ferramenta que permite a passagem dos parâmetros de conexão através de uma interface gráfica.

Ainda assim, o artigo *Agents and Databases*[SAHU97] tem como propósito realizar um levantamento inicial de como os agentes móveis podem resolver ou melhorar alguns problemas relacionados à base de dados e recuperação da informação. O autor verifica a origem dos problemas, delimitando-os aos agentes móveis, trazendo como características as definições dos agentes, aplicações com acesso às bases de dados, além de tratar questões como performance e segurança.

E por fim, como parte integrante do *Mobile Agent Computing, A White Paper*[HORI98], fornecido pela Mitsubishi Electric ITA em Janeiro de 1998, onde se descreve as funcionalidade do Concordia, sua infraestrutura para o desenvolvimento de agentes móveis, fornecendo exemplos de agentes colaborativos que fazem acesso à base de dados relacional, utilizando o banco de dados Access da Microsoft. Este *White Paper* exemplifica, através do Concordia, a utilização e manipulação dos agentes móveis, para que os mesmos se desloquem pela rede e obtenham a informação de um único banco implementado.

Diante destes artigos e publicações, este trabalho descreve a integração de diferentes tipos de bancos de dados: relacional, objeto-relacional e orientado a objeto. É utilizado para isso, a mobilidade do código de um agente móvel, a comunicação entre agentes móveis e agentes estáticos e a meta a alcançar do agente móvel. Este, por sua vez, irá percorrer cada base de dados, fornecendo os requisitos necessários e retornando os resultados, realizando assim a integração destas diferentes formas de armazenamento de dados, de uma forma transparente para o usuário.

1.2. – Trabalhos publicados

O presente trabalho foi publicado em dois eventos de renome, sendo um nacional e outro internacional. Além disso, o mesmo foi submetido para outro evento: CACIC 2000, na Argentina. Segue abaixo as publicações:

Claro, Daniela Barreiro; Sobral, João B.M.; **Integration of Databases using Mobile Code**, Agent-Based Simulation, Passau - Alemanha, 2000, promovido pela Society for Computer Simulation -SCS[CLAR00a].

Claro, Daniela Barreiro; Sobral, João B.M.; **Agentes Móveis aplicados à Integração de Bases de Dados**, Semish' 2000, Congresso Brasileiro de Computação, Curitiba - Brasil, 2000[CLAR00b].

1.3. – Organização do Trabalho

O presente trabalho está organizado em capítulos, sendo que os dois subsequentes contêm a fundamentação teórica. Os capítulos 4 e 5 fornecem abordagens sobre a integração das bases de dados e implementação do ambiente, sendo os demais sobre a conclusão e referências bibliográficas, todos detalhados em seguida:

Capítulo 2 – Acesso à Base de Dados

Este capítulo traz conceitos básicos sobre as diversas bases de dados: relacional, objeto-relacional e orientadas a objeto, assim como seus respectivos acessos, situando o leitor no tema abordado.

Capítulo 3 – Paradigma dos Agentes

Este fornece subsídios necessários ao leitor para uma posterior leitura das propostas que envolvem este novo paradigma. Neste capítulo será abordado desde o histórico e necessidade de surgimento dos agentes, concluindo com os agentes móveis e suas características.

Capítulo 4 – A Integração dos Bancos de Dados

Esta etapa é composta pela integração dos bancos de dados no que concerne à utilização das bases de dados, dos agentes móveis e suas interações com os agentes estáticos. Além disso, será explicitado o desenvolvimento dos agentes e das bases de dados, assim como suas características para a posterior implementação.

Capítulo 5 – Implementação do Ambiente

Esta fase terá como objetivo exemplificar a efetiva utilização dos agentes, assim como os requisitos necessários para a sua utilização, além do ambiente escolhido para esta implementação.

Capítulo 6 – Conclusão

Nesta fase final, será realizada uma avaliação do trabalho em questão, enumerando as principais conclusões, assim como, a indicação de futuras extensões ao mesmo.

2. Acesso à Base de Dados

2.1. - Histórico

Desde o surgimento dos primeiros sistemas de computação, a necessidade de armazenar e recuperar dados de forma confiável e segura vem se apresentando como um dos principais desafios para os especialistas.

O surgimento dos primeiros computadores comerciais reforçou essa demanda, principalmente porque começou a surgir a necessidade de armazenamento de grandes volumes de dados e principalmente sua recuperação de forma rápida e segura.

Os primeiros sistemas de arquivos, com a utilização de técnicas de organização e acesso, com destaque para o VSAM (*Virtual Storage Access Method*), ofereceram os primeiros recursos confiáveis para o manuseio dos computadores pelas grandes corporações[NAVA94].

Apesar do avanço alcançado por sistemas como o VSAM, esse não foi suficiente para suportar o aumento no volume de acessos concorrentes e do fluxo de usuários do sistema.

Até então, a área de banco de dados era caracterizada por estudos teóricos, quando os fabricantes começaram a implementar os primeiros Sistemas Gerenciadores de Banco de Dados (SGBD) para utilização em computadores de grande porte. Estes sistemas estavam baseados em modelos de dados hierárquico ou de redes. Tanto o modelo hierárquico quanto o modelo de redes eram orientados a registro, exigindo uma modelagem eficiente, porém, pouco estruturada das aplicações, além de certa complexidade no estabelecimento dos acessos aos dados[VASC98].

Muitos sistemas foram desenvolvidos utilizando estas plataformas e as empresas começaram a usufruir dos benefícios fornecidos pela utilização de um banco de dados.

As pesquisas em Banco de Dados prosseguiram, surgindo o modelo relacional, definido por E.F.Codd. O modelo relacional trouxe uma nova perspectiva de representação e recuperação dos dados: a orientação a conjuntos. Concomitante ao modelo relacional, surgiu a linguagem SQL, que se tornou a linguagem padrão de fato dos SGBDs relacionais. A linguagem SQL possui sentenças para definição e manipulação dos dados. Em 1976, o modelo de Entidades e Relacionamentos (E/R) foi introduzido por Peter Chen e passou a ser a principal proposta na área de modelagem semântica dos dados. O modelo E/R é geralmente utilizado no processo de estruturação e representação da estrutura lógica do banco de dados.

Embora o modelo relacional continue sendo dominante no mercado, aplicações mais complexas estão surgindo, incrementando assim a necessidade de um novo modelo que incorpore as requisições destas novas tecnologias. Dentre estes novos modelos surgem o Banco de Dados Orientado a Objetos e mais recentemente o Objeto-Relacional.

O desenvolvimento destes dois modelos, incluindo características como o armazenamento de dados persistentes, tipos de dados multimídia, acesso remoto aos dados, além das características da orientação a objetos, permitiram o surgimento de uma especificação para linguagens de acesso a banco de dados orientado a objetos e ainda para os que detém as características relacionais, combinados com o paradigma de

objetos. Assim, a extensão do SQL-2 (mais conhecido como SQL-92 da ANSI), permite uma utilização mais eficiente dos recursos fornecidos pelos bancos de dados que suportam objetos mais complexos integrados às características relacionais, tais como os bancos de dados objeto-relacionais.

2.2. – Tecnologia de Banco de Dados

Os sistemas de Bancos de Dados vêm sendo parte integrante do desenvolvimento das aplicações desde que os computadores passaram a ter grande cunho em âmbito comercial. Inicialmente, os repositórios de dados eram simples sistemas de arquivos, onde o dado que fosse necessário posteriormente em uma aplicação, tinha que ser armazenado fisicamente. Estes sistemas de arquivos atenderam aos requisitos dos computadores de grande porte, devido, principalmente, a execução dos seus programas (*jobs)* ser seqüencial e não exigirem compartilhamento em tempo de execução entre os mesmos.

Com o advento dos sistemas multiusuários e o acesso concorrente a estes sistemas de arquivos, o desenvolvimento de sistemas gerenciadores não tardou em emergir. Assim, segundo [FERR93], banco de dados é "um repositório eletrônico de dados com recursos de pesquisa às informações", assim como base de dados é "um conjunto de dados organizados de forma a facilitar a pesquisa às informações", sendo usada, neste trabalho, as duas nomenclaturas para armazenamento de informações.

Um sistema gerenciador de banco de dados consiste de três componentes que se sobressaem: o banco de dados ou repositório de armazenamento de dados, um software gerenciador deste banco e uma linguagem de interface.

Um repositório de dados contém quatro elementos essenciais, tais como os **itens de dados**, que são os responsáveis por representar as informações do mundo real no formato binário; os **relacionamentos**, que representam as correspondências entre esses

itens; as **restrições** do banco de dados que garantem a consistência e integridade do sistema; e por fim os **esquemas**, que definem a organização e os relacionamentos dos dados na base em questão. Os esquemas são responsáveis pela separação dos aspectos físicos no armazenamento de dados dos aspectos lógicos, explicitados na representação dos mesmos[DATE91].

Além disso, os sistemas gerenciadores de banco de dados suportam linguagens para a definição e manipulação dos dados. Neste caso, o esquema conceitual é especificado na linguagem de definição dos dados (DDL), assim como as restrições e os seus relacionamentos. Em relação à linguagem para manipulação de dados (DML), a mesma é utilizada para expressar as operações realizadas no banco de dados. Diante disso, às vezes a DML é referenciada como uma linguagem de consulta, sendo este apenas um dos mecanismos fornecidos pela mesma. A figura 2.1 ilustra as interações do SGBD (Sistema Gerenciador de Banco de Dados).

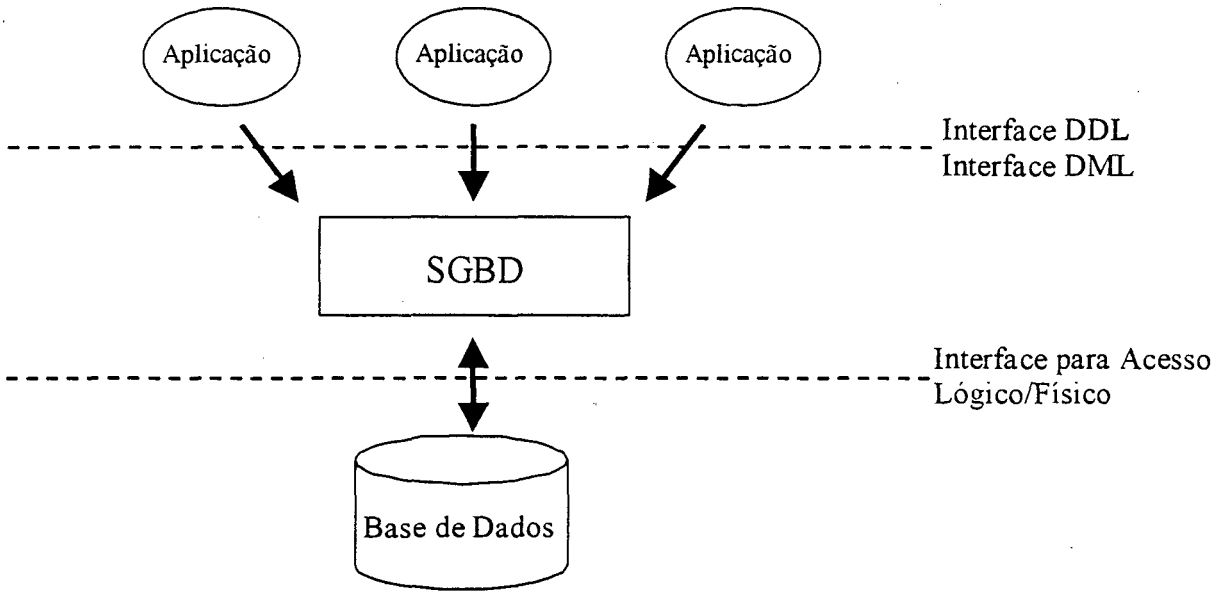

FIGURA 2.1 INTERAÇÕES DO SISTEMA GERENCIADOR DE BANCO DE DADOS

Assim, um sistema gerenciador de banco de dados provê serviços de acesso, manipulação e alteração da base de dados, ao mesmo tempo mantendo a integridade e consistência das informações armazenadas.

Os primeiros bancos de dados que surgiram com o conceito de sistemas gerenciadores foram os bancos de dados hierárquicos e de redes. Porém, suas estruturas complexas de indexação dos dados permitiram que, por volta dos anos 70, fossem gradativamente substituídos pelos bancos de dados relacionais, que a partir de então traziam o conceito da orientação a conjunto e não mais a registo, ou dado, facilitando a forma de acesso.

2.2.1. – Banco de Dados Relacional

O surgimento do modelo relacional trouxe à tona a abstração de dados e o desenvolvimento de aplicações não direcionadas a apontadores ou registros, mas a um conjunto de dados.

A estrutura de dados do modelo relacional é a relação ou, informalmente, a tabela. Um banco de dados relacional é representado como uma coleção de relações. Cada tabela é composta por um conjunto de linhas, onde cada linha é uma lista de valores de atributos. A estas linhas dá-se o nome de tuplas. Cada tipo de dado que pode aparecer nas colunas, determinadas como os atributos, fazem parte de um domínio de valores possíveis[NAVA94].

Um esquema de relação R, representado por R(A1,A2,A3...An) é formado pelo nome da relação R e uma lista de atributos A1, A2, A3...An, onde cada atributo Ai pertence a um domínio de valores[MELO98].

Na figura abaixo observa-se um exemplo onde PESSOA é o nome da relação que possui cinco atributos.

PESSOA(CPF, Nome, Endereço, Data_Nascimento, Renda)

PESSOA

CPF	Nome	Endereço	Data_Nascimento	Renda

FIGURA 2.2 ESQUEMAS NA FORMA DE NOTAÇÃO DE RELAÇÃO E REPRESENTAÇÃO COMO TABELA.

Uma relação ou a instância da relação é um conjunto de tuplas onde cada tupla é composta por uma lista de valores, sendo os mesmos pertencentes a um domínio. Assim, cada tupla terá o seu valor correspondente na relação, como segue o exemplo abaixo:

PESSOA

111.111.111-11	Daniela Claro	Rua A, nº 123 - SSA	14/10/1976	R$ 5000,00
222.222.222-22	Pepe Barreiro	Rua B, nº 456 – SSA	11/11/1999	R$2000,00

FIGURA 2.3 TABELA QUE REPRESENTA AS INSTÂNCIAS DA TABELA PESSOA.

2.2.1.1. – Integridade de Dados no Modelo Relacional

Um dos mais importantes princípios de um modelo de banco de dados é o princípio da integridade de dados, realizado através de restrições que são condições obrigatórias impostas por este modelo. Estas restrições incluem as restrições de domínio, de chaves, integridade de entidade e integridade referencial[MELO98].

Restrições de Domínio

As restrições de domínio são impostas em relação às colunas das tabelas. Elas especificam um conjunto de valores de dados válidos, um domínio, que possa ser atribuído à coluna em questão.

As regras de domínio são especialmente definidas para um atributo de uma tabela. Normalmente são criadas como máscaras para digitação de dados ou especificam uma faixa de valores numéricos, uma lista de itens permitidos em uma coluna, entre outros componentes.

Os tipos de dados associados aos domínios incluem tipos primitivos, como números inteiros e reais, tipos alfanuméricos, tipos especiais como data, moeda, objeto longo, dentre outros[NAVA94].

Restrições de Chaves

No esquema de uma relação, uma chave é "um atributo ou conjunto de atributos cujo valor ou combinação de valores deve ser distinto em qualquer instância da relação"[MELO98].

A existência de pelo menos uma chave é condição obrigatória, visto que uma tabela é definida como um conjunto de tuplas ou registros e, portanto, todos os registros devem ser distintos entre si.

Uma vez definida esta chave, denomina-a de chave primária (*primary key)*, onde a mesma é composta por um atributo ou um conjunto deles. No caso de haver mais de um atributo ou conjunto de atributos candidatos (chave candidata), as que não foram selecionadas passarão a se chamar de chaves alternativas. É importante observar que toda relação tem uma chave candidata trivial, composta pelo conjunto de todos os atributos da relação. Neste caso, a relação é conhecida como "relação toda chave". Para solucionar este problema, é criado um atributo seqüencial para servir como chave primária[NAVA94].

Integridade de Entidade

Esta restrição é decorrente da chave primária da tabela, visto que a mesma não pode ter nenhum valor nulo. Isso é uma condição obrigatória, em se tratando de chaves primárias, pois as mesmas têm como função identificar as tuplas na relação.

Integridade Referencial

As restrições citadas acima, se referem à integridade das tuplas em uma única relação, enquanto que a integridade referencial trata das restrições que correspondem às tuplas em duas ou mais relações.

O atributo de uma relação que referencia o atributo de outra relação é chamado de chave estrangeira ou chave externa (*foreign key*). As chaves estrangeiras podem possuir valores nulos e não precisam, necessariamente, fazer parte da chave primária da relação que a mesma faz referência[DATE91]. Desse modo, a regra de integridade referencial permite que o valor de uma chave estrangeira de uma relação R deve ser equivalente ao valor da chave primária de alguma linha da relação referenciada ou totalmente nula. Além disso, através da integridade referencial pode-se garantir que os relacionamentos especificados entre as relações permaneçam corretos, mesmo após ocorrer alterações em tuplas desta relação[VASC98].

Os relacionamentos estabelecidos entre chaves primárias e chaves estrangeiras podem ser de três tipos:

- Um para um (1-1) – um único valor de chave primária na tabela pai[1] referenciando a um único valor de chave estrangeira na tabela filho[2]. Nestes casos, é preferível torná-los atributos da relação.
- Um para muitos (1-n) – um único valor de chave primária na tabela pai e um ou muitos valores de chave estrangeira na tabela filho.

---

[1] Tabela Pai será a fornecedora de chaves primárias, a mestre no relacionamento, a tabela a qual será referenciada.

[2] Tabela Filho será a que referencia os atributos.

- Muitos para muitos (n-m) – relacionamento entre qualquer um dos registros na tabela pai, com um ou muitos na tabela filho.

Assim, a integridade referencial permite que os dados do banco de dados estejam sempre corretos e consistentes, garantindo assim a estabilidade do sistema.

2.2.2. – Banco de Dados Orientado a Objeto

A abordagem da orientação a objeto, em se tratando de linguagens de programação, se iniciou por volta dos anos 60. Porém, em relação à base de dados, estes conceitos só foram aparecer nos anos 80. Isso ocorreu pelo fato das aplicações atuais estarem se tornando cada vez mais complexas, tornando os bancos de dados relacionais inadequados ao armazenamento das informações.

Os bancos de dados orientados a objetos adotaram muitos dos conceitos implementados nas linguagens de programação com este mesmo paradigma. A principal diferença entre os mesmos é que, no banco de dados, os objetos têm características persistentes, enquanto que nas linguagens, os mesmos são transientes. Objetos persistentes trazem como característica básica sua existência permanente em uma base de dados, para que possam ser acessados por outros programas e aplicações.

O conceito fundamental de um banco de dados orientado a objetos é o próprio objeto. Um objeto é uma representação abstrata de uma entidade do mundo real. A estrutura interna de um objeto inclui a especificação de variáveis de instância, ou simplesmente atributos, onde estes são encapsulados com o objeto e não necessariamente podem ser atributos públicos. Além disso, estas variáveis de instâncias também podem ser outros objetos. Com o intuito de acrescer encapsulamento, uma operação ou método de um objeto é definida em duas partes. A primeira delas corresponde à interface, onde são definidos os nomes das operações e seus parâmetros. A segunda parte seria o método propriamente dito, onde é desenvolvida sua implementação. Estas operações podem ser invocadas passando uma mensagem para o

objeto que inclua o nome da operação e os seus parâmetros. Este encapsulamento permite a alteração da implementação dos métodos sem alterar as interfaces[BELL92].

Uma outra vantagem do banco de dados orientado a objeto é a correspondência direta entre o mundo real e os objetos do banco de dados, pois desta forma eles não perdem a integridade e identidade, além de serem facilmente identificados e manipulados.

Identidade do Objeto

Um banco de dados orientado a objeto (BDOO) provê uma identificação única para cada objeto armazenado. Esta identidade é implementada através do identificador do objeto (OID). O valor do OID não é visível ao usuário externo, mas é utilizado internamente pelo sistema para identificar o objeto, criar e gerenciar as referências entre eles.

O valor de um OID, uma vez definido nunca mais será alterado. Esta é a principal propriedade do OID. O mesmo preserva a identidade do mundo real que está sendo representada. Mesmo que o objeto seja removido da base de dados, este OID não deverá ser repassado para outro objeto [NAVA94].

Hierarquia de Tipos/Classes e Herança

Na maioria das aplicações que envolvem bases de dados há muitos objetos do mesmo tipo. Neste caso, BDOO deve prover um meio de classificar estes objetos de acordo com o seu tipo. Uma vantagem deste tipo de banco de dados é que o mesmo permite a definição de tipos baseados em outros tipos pré-existentes, criando assim um hierarquia de tipos. Um tipo é definido associando um nome, os atributos e suas operações. Os subtipos herdam todas as características dos tipos além das suas particularidades, como ilustra a figura 2.4:

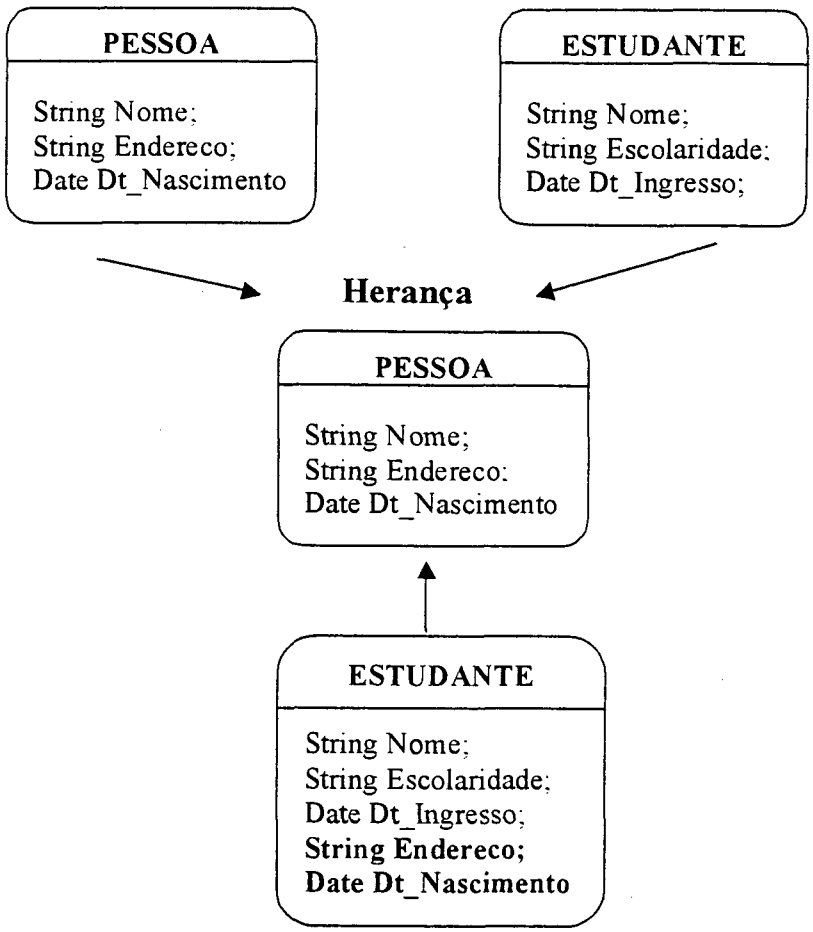

FIGURA 2.4 EXEMPLIFICAÇÃO DE UMA HIERARQUIA DE TIPOS OU CLASSES.

Em BDOO que são baseados em tipos, a hierarquia das classes tem uma correspondência na hierarquia de tipos, desde que todos os objetos da classe contenham o mesmo tipo. Não se pode generalizar que toda classe é uma coleção de objetos cujos tipos de dados são iguais. A linguagem *Smalltalk* não caracteriza esta premissa. Em relação às classes, também há a hierarquia, que pode ser observada quando se define subclasses para uma superclasse pré-definida. O mesmo exemplo se aplica ao citado anteriormente. Alguns sistemas orientados a objetos definem classes de sistemas, chamadas de ROOT, ou de OBJECT, como ocorre em *Smalltalk* e na linguagem Java, respectivamente.

Os bancos de dados orientado a objetos correspondem a cerca de 1 ou 2% do mercado, com relação ao mercado dos bancos de dados relacionais. Neste sentido,

devido a problemas até então comerciais e relativos aos custos necessários para a sua implantação, o mesmo ainda não atingiu a devida demanda no mercado comercial.

2.2.3. – Banco de Dados Objeto-Relacional

Apesar das vantagens citadas em relação aos BDOOs, estes representam uma mudança muito brusca em relação a tecnologia atualmente adotada dos banco de dados relacionais (BDR). Adicionando-se ao fato de que não reutilizam as estruturas das tecnologias anteriores (BDR), os mesmos ainda requerem novos conjuntos de habilidades que muitos profissionais não possuem no momento e que, para atingir tal ponto, necessitam de treinamento especializado.

Diante deste contexto surgem os Banco de Dados Objeto-Relacional (BDOR), que permitem uma integração das atuais tecnologias, com os conceitos de orientação a objeto. Neste sentido, os BDOR representam uma progressão natural baseada nos benefícios da combinação entre uma base de dados relacional com uma base de dados orientada a objetos[FORT99].

Nos BDOR as relações continuam sendo a base do modelo, porém com o acréscimo dos objetos, que poderão ser armazenados persistentemente em tabelas relacionais como tipos complexos ou tipo abstrato de dados.

Duas questões são essenciais em relação aos BDOR:

- Os BDOR devem suportar objetos complexos, tais como textos, gráficos, imagens, dados multimídia, etc.
- Devem suportar as características dos BDR, incluindo o SQL e a independência dos dados.

Algumas empresas já incorporaram estas características e lançaram seu produto no mercado, tais como: Oracle 8i (Oracle); o DB2 5.0 (IBM); o Informix (Informix).

2.3. - Manipulação de Dados

2.3.1. - A linguagem SQL

Originalmente, a SQL (Structure Query Language) era denominada de SEQUEL (Structured English QUEry Language), quando foi implementada no protótipo de pesquisa do SGBD relacional Sistem R, da IBM em meados dos anos 70. Na época, a SQL já tinha sido projetada como uma linguagem natural para ser utilizada em programas de aplicação e também diretamente por usuários finais, em programas em consultas ocasionais[NAVA94].

Impulsionados com os bons resultados obtidos nos testes da IBM, outros fabricantes decidiram produzir sistemas baseados na SQL, levando a se tornar, de fato, a linguagem padrão para banco de dados relacional. A partir daí a SQL passou por duas atualizações oficiais, em 1989 e outra em 1992[VASC98]. Assim, em um esforço do ANSI (American National Standard Institute), juntamente com a ISO (International Standards Organization), a SQL foi adotada como um padrão, atualmente em vigor, denominada de SQL-92 ou SQL2.

A SQL não é uma linguagem de programação, mas muitas implementações em SGBDs comerciais incluem extensões procedurais que a tornam pertencentes a esta categoria. Visto que, de fato, nenhum SGBD (Sistemas Gerenciadores de Banco de Dados) suporta todas as características da SQL-92, cada SGBD tem suas próprias extensões não previstas no documento padrão.

Assim como a maioria das sublinguagens de dados presentes nos bancos de dados, a SQL está disponível em duas interfaces: a interface interativa e a interface de programação da aplicação. No primeiro caso, o usuário envia comandos SQL através do terminal e obtém a resposta, caso exemplificado por clientes de bases de dados. No segundo caso, a instrução está embutida em um programa de aplicação, cuja declaração

será executada quando o programa estiver sendo manuseado e o resultado da operação retornará através de uma variável do programa.

A linguagem SQL possui um alto nível de abstração e propicia tanto funções de definição de dados, quanto funções de manipulação dos mesmos.

As principais funções de definições de dados incluem a criação e modificação de esquemas, domínios, tabelas e visões. E em relação às manipulações dos dados, as principais são as consultas (*select*), e as modificações dos dados (*insert, delete e update*).

2.3.1.1. - Linguagem de Definição dos Dados (DDL)

A DDL utilizada pela SQL permite a especificação não apenas de um conjunto de relações (tabelas), mas, informações sobre cada tabela que inclui o esquema para cada relação, o domínio de valores associados a cada atributo, o conjunto de índices a ser mantido pela relação, requisitos de segurança, integridade e estrutura física de armazenamento de cada relação no disco[MELO98].

A criação de uma tabela básica permite a designação de um nome e definição da estrutura física através da instrução CREATE TABLE. Esta instrução fornece o nome da tabela a ser criada, as colunas e seus tipos de dados. Além disso, pode fornecer também questões como definição de chaves primárias, além de identificação dos índices e valores nulos.

Uma vez criada, a tabela poderá ser removida do sistema através da cláusula DROP TABLE. Caso seja necessário algum tipo de alteração na estrutura da tabela, a instrução ALTER TABLE poderá ser utilizada.

2.3.1.2. - Linguagem de Manipulação dos Dados (DML)

A DML é uma linguagem de consulta aos dados de uma tabela baseadas na álgebra relacional e no cálculo relacional de uma tupla. As operações de recuperação e atualização dos dados são realizadas através das instruções DML.

A sintaxe básica da operação para recuperação de dados é

SELECT [DISTINCT] <lista de atributos>
FROM <lista de tabelas ou visões>
[**WHERE** <condição de seleção;junção>
[**GROUP BY** <lista de atributos>
    [**HAVING** <condição de seleção>]]
[**ORDER BY** <coluna> [(**ASC** | **DESC**)]
    {, <coluna> [(**ASC** | **DESC**)]}]

O comando SELECT permite que se tenha inúmeros tipos de consulta, inclusive comandos aninhados, isto é, comandos SELECT dentro de cláusulas WHERE.

Outra possibilidade de manipulação de dados é através das atualizações que ocorrem nos mesmos. Assim, o usuário poderá inserir registros nas tabelas através da cláusula INSERT, utilizada como segue abaixo:

INSERT INTO <tabela> [(lista de colunas)]
VALUES (<lista de valores>) | <comando select>

Enquanto o comando DELETE remove tuplas de uma relação. Este ainda pode incluir a cláusula WHERE, similarmente a utilizada na consulta, permitindo assim selecionar os registros para serem excluídos.

DELETE FROM <tabela>
    [**WHERE** <condição>]

E para finalizar, a cláusula UPDATE é utilizada para modificar valores de atributos para uma ou mais tuplas selecionadas. Neste caso, a cláusula WHERE também poderá ser empregada.

UPDATE <tabela>
SET <coluna> = <valor>

Assim, ainda há outras utilizações da SQL, confirmando sua presença inerente nos bancos de dados relacionais, afirmando assim uma hegemonia que subsidiará o desenvolvimento de novas linguagens baseadas neste conceito do SQL.

2.3.2. - A Linguagem SQL-3

A linguagem SQL-3 foi desenvolvida com o intuito de ser uma extensão do padrão SQL-92 (SQL-2). Isso significa que os bancos de dados e as aplicações que utilizam este padrão vão continuar funcionando, sem nenhuma alteração, com o modelo SQL-3.

O padrão SQL-2 permite ser utilizado em modelos de banco de dados puramente relacional, manipulando tabelas relacionais, assim como seus tipos de dados básicos: caracter, bit, string, número, inteiro, entre outros.

O modelo SQL-3 estende este padrão (SQL-2), incluindo tipos de dados como objetos, coleções que são os tipos abstratos de dados (ADT). Os componentes fundamentais do SQL-2 e SQL-3 não foram modificados. A tabela relacional continua sendo o único meio pelo qual os dados podem se tornar persistentes.

2.3.2.1. - Tipos abstratos de dados (ADT)

Os tipos abstratos de dados (ADT) representam uma das mais significantes mudanças dentro do SQL-3. A especificação do ADT consiste em duas partes: uma especificação dos atributos e uma especificação das funções ADT, que correspondem

aos métodos na orientação a objetos. Os comportamentos dos ADT são encapsulados dentro do mesmo, mas as interfaces das funções e a definição dos atributos são externos ao mesmo, onde os detalhes da implementação não são visíveis fora dos tipos.

As funções e atributos do ADT podem ter restrições definidas como públicas (*public*), privadas (*private*) e protegidas (*protected*), onde as públicas permitem que todos os usuários tenham acesso às mesmas, as privadas limitam o acesso ao código interno, e as protegidas limitam o acesso a todos os subtipos do ADT. Esta nomenclatura segue os mesmos padrões das linguagens C++ e Java.

Os objetos ADT detém um número de identificação único para cada instancia deste objeto. Assim, com esta identificação é que se faz o relacionamento dos mesmo com sua instância dentro da base de dados. Além disso o ADT também pode ser utilizado como tipos de colunas em uma tabela, atributos de outras definições de ADTs, parâmetros em procedimentos ou funções e como variáveis em sentenças SQL[FORT99].

Para criar um tipo de dados abstrato faz-se a utilização do seguinte comando:

CREATE TYPE <nome do ADT>
<atributos>

E em relação a manipulação das funções do ADT, a mesma ocorre da seguinte maneira:

FUNCTION <nome do atributo > (<nome do ADT>) [**RETURNS** <tipo retornado>]

2.3.2.2. -Subtabelas e Supertabelas

As subtabelas e as supertabelas são usadas no SQL-3 com o intuito de permitir a hierarquia de coleções. As subtabelas herdam todas as colunas e suas respectivas

definições das supertabelas. Se algum nome de coluna gerar algum tipo de conflito, este deve ser redefinido para retificar o conflito. Além disso, se algum item for atualizado, inserido ou deletado da supertabela, o efeito deverá ser propagado para as demais filhas da operação. Para trabalhar com estas características de herança deve-se utilizar a instrução UNDER, como mostrado a seguir:

CREATE TABLE <nome da tabela filha> **UNDER** <nome da tabela pai>

2.3.2.3 -Linguagem de Definição de Dados (DDL) e Linguagem de Manipulação de Dados (DML)

As manipulações de DDL e DML são semelhantes às utilizadas no SQL-2, porém, podem ser acrescidas de determinadas funcionalidades como em uma inserção, onde um dos atributos da tabela poderá ser uma linha a mais, fazendo referência ao mesmo. Segue um exemplo prático:

INSERT INTO pessoa **VALUES** (
'111.111.111-11', 'Daniela Claro',
ROW ('Rua A', 123, 'Fpolis', 'SC'),
5000,00)

2.3.2.3. - Tabelas Persistentes

As tabelas persistentes são tabelas básicas que são armazenadas persistentemente no banco de dados. Estas tabelas podem ser definidas utilizando a cláusula CREATE TABLE. Os dados inseridos neste tipo de tabelas serão armazenados em um banco de dados, situado em um dispositivo não volátil, enquanto as tabelas temporais são criadas localmente, diferenciado das demais de acordo com a sua criação e sua manipulação de instâncias. A principal diferença entre elas é que o conteúdo das tabelas temporais não podem ser compartilhados por sessões separadas de SQL. Segue abaixo a diferença na criação das mesmas[FORT99]:

CREATE TABLE <nome da tabela persistente>
<atributos>

DECLARE LOCAL TEMPORARY TABLE <nome da tabela temporal>
<atributos>

Os itens anteriormente definidos, assim como os pré-definidos na linguagem SQL-2 garantem que para a manutenção dos dados as informações devem ser enviadas para as tabelas persistentes. Neste sentido, as instâncias do ADT, assim como qualquer outro tipo de dados estendido, devem ser armazenadas em colunas das tabelas. Para tal, deve-se definir o tipo da coluna de acordo com o tipo do dado que se deseja armazenar, por exemplo, caso o dado seja um ADT, é necessário que se defina a tabela persistente com uma coluna do tipo ADT, para que o mesmo possa ser armazenado persistentemente. Para que este armazenamento seja efetivado, há a utilização dos comandos da SQL de inserção (INSERT) ou de atualizações (UPDATE) como vistos na sessão anterior. Segue abaixo um exemplo prático:

```
CREATE TYPE pessoa (
        nome        VARCHAR (40),
        endereco    VARCHAR (50),
        telefone    VARCHAR (15)
)

CREATE TABLE empresa (
        umaPessoa       pessoa,
        salario         NUMERIC,
        dt_Admissão     DATE
)

DECLARE :pessoa1 pessoa;
```

```
BEGIN
        NEW :pessoa1;
        SET :pessoa1.nome = 'Daniela Claro';
        SET :pessoa1.endereco = 'Rua A, 123, Fpolis SC';
        SET :pessoa1.telefone = '222-2222';
        RETURN :pessoa1;
END;
```

Diante disso, para realizar a inserção do ADT em uma tabela persistente utiliza-se, como citado anteriormente, o comando SQL INSERT.

```
INSERT INTO empresa VALUES
        ( pessoa1, 5000.00, 19991216);
```

Assim, a utilização do SQL-3 é similar a do SQL-2, sendo acrescido de alguns tipos de dados que permitem a manipulação mais próxima dos bancos de dados orientados a objetos. Mas, para a manipulação desta linguagens, grandes empresas como a Oracle e Informix desenvolveram os Banco de Dados Objeto-Relacional que permitem uma coesa manipulação e armazenamento dos dados através da SQL-3. Assim é que, o SQL-3 é uma linguagem que estende o SQL-2, permitindo a manipulação dos dados em banco de dados objeto-relacional.

2.3.3. - Persistência dos Objetos

Em qualquer sistema de informação, o armazenamento de dados é uma tarefa essencial. Sistemas orientados a objetos normalmente se referenciam ao armazenamento estático (no disco) como um armazenamento persistente. Assim, "persistência é a propriedade pela qual um objeto existe, independente do processo que o criou" [SETR94].

Para que um objeto se torne persistente é necessário que este guarde seu estado em um armazenamento de dados estático, por exemplo um banco de dados. Atualmente,

a maioria das informações existentes estão sendo armazenadas em um banco de dados relacional ou em sistemas de arquivos. Além disso, mesmo aplicações que trabalham com objetos, guardam estes objetos em arquivos comuns, como se fosse um *stream de bytes*. Porém, banco de dados orientado a objetos é a melhor maneira de se armazenar objetos, visto que os mesmos são mantidos no seu estado similar ao mundo real.

Assim, o ciclo de vida de um objeto persistente excede o tempo de vida da aplicação que o criou. O gerenciamento de objetos persistentes provê um mecanismo de armazenar o estado dos objetos em um ambiente não volátil, para que quando a aplicação termine sua execução, o objeto mantenha o seu valor, continuando a existir.

Existem, então, duas principais formas de persistir os objetos: uma é através da serialização dos dados e a outra através dos bancos de dados orientados a objetos.

2.3.3.1. - Serialização do Objeto

A serialização dos objetos permitida através do Java, fornece ao objeto uma forma básica de armazenamento persistente. Este mecanismo pode ser utilizado para o armazenamento de cópias dos objetos em arquivos comuns ou ainda enviando cópias destes objetos para serem executados em outra aplicação utilizando o JVM (Java Virtual Machine). Através da serialização dos objetos, os mesmos são convertidos em uma seqüência de *bytes* que poderá ser lida posteriormente eqüivalendo-se aos objetos, outrora escritos nesta seqüência (*stream*).

A serialização é um mecanismo simples de armazenamento de objetos persistentes. Esta é eficiente em se tratando de aplicações que operam com uma pequena quantidade de objetos, onde a confiabilidade do sistema não é necessariamente de suma importância.

É necessário realizar uma leitura e uma escrita do objeto por completo. Neste sentido, se a seqüência de bytes que se deseja armazenar tem alguns megabytes, o armazenamento através da serialização tornar-se-á lento, fato este se agravando quando a aplicação necessita de freqüentes atualizações.

Além disso, a serialização não provê um armazenamento de objetos confiável. Caso ocorra alguma falha no sistema, e a aplicação termine no momento em que os objetos estão sendo armazenados através da serialização, o conteúdo do arquivo será perdido.

Diante deste processo de armazenamento de objetos, surge a necessidade de se ter bases de dados que permitam persistir um objeto, porém de forma confiável e rápida.

2.3.3.2. - Acessando os Dados em BDOO

Um BDOO permite que se insira e consulte uma pequena quantidade de objetos por vez, não exigindo recursos desnecessários para a inserção e consulta dos objetos.

Além disso, caso ocorra alguma falha em uma aplicação ou no sistema, um BDOO é responsável por garantir a atomicidade da transação, ou seja, se tudo estiver coeso, em comum acordo, a transação será realizada, caso contrário, a mesma será totalmente abortada, não gerando assim nenhum tipo de inconsistência no banco.

Outra questão importante em relação ao armazenamento persistente em BDOO são as consultas à coleção de objetos. Uma consulta retorna um *subset* de objetos. E, assim como qualquer banco de dados, estas coleções podem ter índices que permitirão o aumento da performance, em se tratando de uma maior quantidade de objetos.

A linguagem de consulta aos objetos é denominada de OQL (Object Query Language) e permite o acesso aos objetos do banco de dados, absorção das informações e invocação de métodos. A OQL pode ser utilizada interativamente ou embutida no código da linguagem de programação, sendo utilizada para o desenvolvimento de aplicações. Esta linguagem foi adotada pela ODMG(Object Database Management Group) desde 1993, sendo uma extensão da linguagem de consulta SQL.

OQL fornece diversas facilidades para a recuperação de dados e informações, explorando, significativamente, os conceitos orientados a objetos. Esta linguagem, porém, somente oferece mecanismos de consulta aos dados, sendo as manipulações como inserção e exclusão dos mesmo ocorrendo de forma singular ou seja, um único objeto por vez, sem trabalhar com coleções.

Manipulando Objetos Persistentes

Para que se obtenha uma manipulação de objetos persistentes, ou seja, para que os mesmos sejam armazenados em BDOO, é necessário que se crie, a princípio, uma sessão, que pode ser definida como um contexto onde o banco de dados será criado. Em seguida, há a criação do banco e consequentemente da transação. Após o início de uma transação, faz-se a transferência do objeto para o banco de dados, a depender do banco de dados que se estiver utilizando.

Assim, a utilização de objetos persistentes com banco de Dados Orientado a Objeto permite um ganho de performance, se comparado com o armazenamento de objetos em BDR ou através da *serialização*, visto que os objetos serão armazenados de acordo com as suas atuais estruturas, sem serem modificados para executarem tal operação.

3. – Paradigma dos Agentes

3.1 Histórico

Durante os anos 80, a comunidade de Inteligência Artificial, desencorajada pela falta de progresso, após 20 anos de pesquisa em projetos de sistema, começou a explorar novas áreas, onde sistemas de Inteligência Artificial pudessem ter um domínio mais dinâmico. Ao invés de olhar para resultados simulados, simbólicos em mundos artificiais, começaram a explorar as possibilidades de interações complexas com o mundo físico, através de um mecanismo denominado **agentes**.

Vários pesquisadores como Marvim Minsk, Oliver Selfridge, Alan Kay, Nicholas Negroponte, Rodney Brooks e Pattie Maes, passaram a estudar problemas que pudessem demonstrar algum tipo de comportamento dos agentes.

Ao mesmo tempo, uma outra comunidade estava se formando usando também o termo **agente**. Era a comunidade de agentes de software, que estava explorando o desenvolvimento de partes de código menores e mais confiáveis. A idéia era desenvolver programas que pudessem agir separadamente, podendo mover informações entre dois ambientes distintos. Entretanto, a quantidade de termos usados para descrever agentes: *intelligent agents, intelligent interfaces, adaptive interfaces, knowbots,*

softbots, userbots, taskbots, personal agents, network agents tornou-se uma barreira para esta comunidade científica.

Pode-se definir um agente como sendo qualquer coisa que percebe seu ambiente através de sensores e atua neste ambiente através de reagentes[RUSS95]. A Figura 3.1 mostra a interação entre agentes e seu ambiente. Os agentes percebem o ambiente através dos seus sensores e executam suas ações no ambiente através do seus executores.

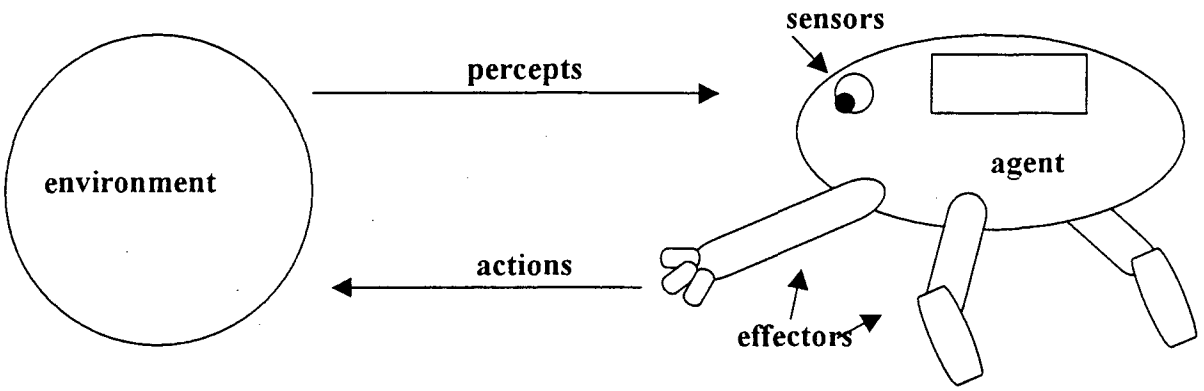

FIGURA 3.1 INTERAÇÃO DOS AGENTES COM O SEU AMBIENTE.

Recentemente, o explosivo crescimento da Internet e a disseminação dos sistemas distribuídos permitiu o surgimento da idéia de agentes que se movem na rede, interagindo com outros agentes e realizando tarefas para os usuários. Agentes inteligentes utilizam as novas tecnologias da Inteligência Artificial para provê autonomia, inteligência e mobilidade, estendendo a abrangência dos usuários por várias redes[BIGU98].

Diante do interesse comercial, a comunidade de inteligência artificial vem modificando a área de pesquisa para permitir a integração com sistemas distribuídos. Inicialmente, a área de agentes inteligentes estava limitada à pesquisa por palavras, absorção de informações e tarefas de filtragem. Porém, com o crescimento das

transações comerciais na rede, há um aumento no interesse de agentes inteligentes para desempenharem tarefas específicas.

3.2. - Agentes

Embora não haja ainda um consenso sobre uma definição formal do que seja um agente, tal que englobe todo o espectro possível, algumas características esperadas foram estabelecidas. Pesquisadores têm proposto uma variedade de definições, cada qual tentando explicar o uso da palavra "agente". Essas definições vão de um nível elementar até um nível mais elaborado.

"Um agente é um programa de computador que funciona em *background*, e desenvolve tarefas autônomas conforme delegadas pelo usuário".

"Agentes são programas que travam diálogos, negociam e coordenam transferência de informações".

"Os agentes apresentam conceitos de habilidade para execução autônoma e habilidade para executar raciocínio orientado ao domínio".

"Um agente pode ser definido como alguém ou alguma coisa que atua como um representante para seu usuário, com o propósito expresso de desempenhar ações que são benéficas para a parte representada. É diferenciado de outras aplicações por suas dimensões e capacidade de interagir independentemente da presença do usuário".

"Agentes são uma entidade real ou virtual que emerge num ambiente, onde pode tomar algumas ações, que é capaz de perceber e representar parcialmente esse ambiente, que é capaz de comunicar-se com outros agentes e que possui um comportamento autônomo que é uma conseqüência de sua observação, seu conhecimento e suas interações com outros agentes"[KOTA94].

"Agente é uma entidade persistente dedicada a um propósito específico". "Persistente" distingue agentes de subrotinas; agentes têm seu próprio conhecimento sobre como realizar tarefas, suas próprias agendas. "Propósito específico" distingue-se de toda aplicação multifunção; agentes são tipicamente mais inteligentes.

O termo agente é utilizado para representar dois conceitos ortogonais. O primeiro se refere à habilidade do agente de ser autônomo. O segundo, a habilidade do agente de ter um raciocínio orientado a um domínio.

Agentes autônomos são sistemas computacionais que habitam em ambientes dinâmicos complexos, percebe e atua neste ambiente e realiza as metas ou tarefas que foram para eles designados.

"*Software Agents* são programas que se engajam em diálogos ou negociações e coordenam a transferência de informações"[FRAN96].

As definições apresentadas tentam exprimir o que cada autor entende por agente, baseado nas características que seu agente possui. Pelo fato de cada autor, na maioria das definições encontradas, envolver o problema que o agente busca solucionar, encontram-se as diferentes interpretações do termo.

Assim, agente é um programa que detém autonomia, inteligência e desempenha suas tarefas de acordo com os interesses dos usuários. Para tal, um agente pode se comunicar com outros agentes e ambientes, sempre tendo como meta principal a realização da sua tarefa.

3.2.1. - Características dos Agentes

Abaixo segue uma lista das características gerais de um agente. Assim, os agentes devem deter estes atributos para que sejam distinguidos de programas normais ou subrotinas.

Autonomia: Os agentes operam sem a intervenção direta do usuário e possuem algum tipo de controle em cima de suas ações e do seu estado interno.

Habilidade Social: São os agentes que interagem com outros agentes e (possivelmente) como seres humanos, através de algum tipo de linguagem de comunicação.

Reatividade: Os agentes percebem seu ambiente (que pode ser o mundo físico, um usuário através de uma interface gráfica, outros agentes, a Internet ou talvez tudo isto combinado) e respondem a esses estímulos.

Iniciativa: Neste caso, os agentes além de serem reativos, devem ter um comportamento que satisfaça seus objetivos.

Continuidade Temporal: Os agentes executam continuamente processos que tanto podem estar ativos, (*foreground*), quanto inativos, (*background*).

Orientação a Objetivos: Nesta situação, 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 e em qual ordem e de que modo devem ser executadas, deve ser feita pelo próprio agente.

Além destas características, os agentes devem possuir significados ainda maiores com o intuito de implementar características mais próximas dos seres humanos. Para tal, o agente em questão deve possuir um ou mais destes conceitos abaixo enumerados:

Mobilidade: Habilidade de um agente mover-se em uma rede.

Benevolência: Suposição que os agentes não têm objetivos conflitantes e que todo agente consequentemente sempre tentará desempenhar a tarefa que lhe foi solicitada.

Racionalidade: Hipótese que um agente deve agir de forma a atingir seus objetivos e não ir contra eles, pelo menos dentro do alcance de suas crenças.

Adaptabilidade: Característica que um agente deve ser capaz de adaptar-se aos hábitos, métodos de trabalho e preferências de seus usuários.

Colaboração: O agente não deve aceitar (e executar) instruções sem considerações, mas deve levar em conta que o usuário humano comete erros, omite informações importantes e/ou fornece informações ambíguas. Neste caso, um agente deve checar estas ocorrências fazendo perguntas aos usuários. Deve ser permitido a um agente recusar a execução de certas tarefas que possam sobrecarregar a rede ou causar danos a outros usuários.

3.2.2. - Propriedades dos Agentes

Os agentes possuem propriedades que a *posteriori* permitirão uma classificação dos mesmos. Para que o sistema seja considerado agente, ele não necessita apresentar todas as propriedades, conforme visto nas definições de agentes, mas algumas delas são recomendáveis.

3.2.2.1. - Autonomia

Segundo *Castelfranchi in Wooldridge* e *Jennings* existe autonomia quando os agentes operam sem a intervenção direta de humanos ou outros, e têm alguma espécie de controle sobre suas ações e seus estados internos. Diante disso, uma analogia seria a percepção de um relógio, caso seu proprietário viajasse para a Austrália determinado dia, e um mecanismo, interno ao relógio, ajustasse automaticamente a hora[RUSS95].

O comportamento de um agente pode ser baseado tanto na sua experiência quanto no conhecimento utilizado para a construção do mesmo no ambiente que opera. Um sistema é dito autônomo quando seu comportamento é determinado pela sua própria experiência.

3.2.2.2. - Mobilidade

Mobilidade é a capacidade de transportar-se de uma máquina para outra, preservando seu estado interno. Assim, o agente ocupa diferentes nodos e recursos em uma rede ao longo do tempo.

3.2.2.3. - Comunicação

Os agentes, no curso da realização de seus objetivos, devem acessar informações da fonte sobre o estado atual do ambiente externo. Isso requer uma habilidade de comunicar-se com os repositórios dessa informação, que podem ser outros agentes. Essa comunicação pode ser na forma de um pedido/levantamento com um simples e conciso conjunto de respostas possíveis ou ele pode ser uma comunicação complexa com respostas variáveis.

Nas interfaces entre os agentes, é necessário decidir que tipo de declaração os agentes serão capazes de gerar e compreender. Obviamente, o projeto da interface é cuidadosamente relatado para o projeto da arquitetura de todo o sistema. Há ainda um outro problema na comunicação: a interpretação e o significado das declarações do agente. Por exemplo, um termo pode ter diferentes significados para diferentes agentes, o que é chamado de conflito. Diferentes termos podem, entretanto, ter significados similares, o que significa uma correspondência.

Assim, comunicabilidade é a capacidade que os agentes têm de se comunicar com outros agentes, ou outras entidades com o intuito de fornecer e obter informações. Para isso, os agentes devem se comunicar através da mesma linguagem, assim como os seres humanos.

3.2.2.4. - Aprendizagem

Um agente é capaz de aprender quando possui a capacidade de acumular conhecimento baseado em experiência passada e, consequentemente, modificar seu comportamento em resposta a novas situações. Neste sentido, para os agentes serem autônomos e demonstrarem raciocínio, é necessário que os mesmos avaliem o estado atual do ambiente no qual eles estão imersos e incorpore-os para que, em situações posteriores que os conduzam às situações similares, os mesmos adaptem as suas ações com o intuito de melhorar a eficiência na realização das suas tarefas.

Um agente, o qual é fornecido informações sobre o ambiente, domínio do conhecimento ou como realizar uma tarefa particular *on-line*, é dito como capaz de aprender por instrução.

3.2.2.5. - Reatividade

Agentes percebem seus ambientes e respondem de forma oportuna às mudanças que ocorrem nele. Assim, os agentes percebem e reagem às mudanças no seu ambiente.

3.2.2.6. - Pro-atividade, Iniciativa

Esta propriedade permite que um agente possua iniciativa, permitindo que o ambiente realize um comportamento baseado no seu objetivo, não simplesmente por reagir ao ambiente que está submetido.

3.2.2.7. - Cooperação

Os agentes devem ter a capacidade de cooperação, para que trabalhem juntos com outros agentes e realizem tarefas complexas, aproveitando as particularidades de cada um no intuito de atingir seus respectivos objetivos.

3.2.2.8. - Raciocínio

A capacidade de raciocinar é, talvez, o aspecto mais importante que distingue um agente de outros programas. Assim, um agente que detém esta propriedade possui a habilidade de inferir e extrapolar baseado no conhecimento atual e experiências. Este raciocínio pode ser divido em três tipos:

Baseado em regras, onde utilizam um conjunto de condições prévias para avaliar as condições do ambiente externo.

Baseado em conhecimento, onde os agentes detém conjuntos de dados sobre cenários anteriores e ações resultantes, dos quais eles deduzem seus movimentos futuros.

Baseado na evolução artificial, onde estes agentes criam novos agentes com capacidades de raciocínios crescendo de acordo com a proliferação.

3.2.3. - Classificação dos Agentes

Apesar dos agentes terem sido integrados aos ambientes comerciais recentemente, os estudos dos mesmos já tinham sido iniciados há muitos anos. Desde então, os agentes foram classificados de acordo com suas características ou de acordo com sua estratégia de processamento, ou pela função que os mesmos realizam ou ainda de acordo com a localização onde são executados.

3.2.3.1 - Agentes Inteligentes

Agentes inteligentes podem ser descritos em termos de espaço definido por três dimensões: agência, inteligência e mobilidade, conforme apresentado por O'Connor et al. da IBM.

Agência – É o grau de autonomia que um agente detém quando representa um usuário para outros agentes, aplicações e ambientes.

Inteligência – É a capacidade de absorver conhecimento específico, além de processar a resolução de problemas, ou seja, aceitar a declaração de objetivos do usuário

e realizar tarefas delegadas a ele. Níveis altos de inteligência incluem um modelo de usuário ou alguma outra forma de compreender e raciocinar sobre o que o usuário deseja fazer, além de planejar o meio para atingir os objetivos.

Mobilidade – Determina a capacidade do agente de se mover em uma rede. Neste sentido, um agente móvel introduz maior complexidade a um agente inteligente, visto que no agente móvel há o conceito de segurança que deve ser relevado.

Todo agente deve ter um certo grau mínimo de inteligência, no sentido de poder ser considerado um agente. Contudo, uma ampla variação no aspecto da inteligência pode ser considerada, que vai desde agentes simples com inteligência limitada até a complexidade dos agentes altamente inteligentes. A inteligência de um agente é formada por três componentes principais: sua base de conhecimento interna, a capacidade de raciocínio baseada no conteúdo dessa base de conhecimento e a habilidade para aprender ou adaptar-se às mudanças no ambiente[BREN98].

3.2.3.2 - Agentes Autônomos

Estes agentes trazem consigo uma característica peculiar, a autonomia. Assim, um agente autônomo é um sistema situado dentro de um ambiente, sentindo e agindo sobre o mesmo através do tempo e esforçando-se para realizar suas tarefas.

Há agentes que podem ser autônomos, mas não são inteligentes. Estes sempre atuam como simples monitores no gerenciamento de aplicações em sistemas distribuídos. O *Simple Network Management Protocol* advindo da Internet, é um exemplo de agente deste tipo. Por outro lado, há programas ou aplicações que se utilizam da inteligência artificial, como aprendizado e reação, porém, tem pouca autonomia. Sistemas especialistas clássicos exemplificam estes tipos.

3.2.3.3 - Agentes Coordenados

Estes agentes são capazes de gerenciar interdependências entre atividades. Os agentes podem ser coordenados com o intuito de alcançar um objetivo maior ou ainda

completar suas atividades. A coordenação pode ser dividida três conceitos de comportamentos:

Comportamento de especificação, onde há a criação de objetivos compartilhados.

Comportamento de planejamento, onde os agentes expressam um conjunto de tarefas ou estratégias para acompanhamento dos objetivos.

Comportamento seqüencial, onde há uma distribuição das tarefas para os grupos e indivíduos, criando planos e seqüências compartilhadas, alocando recursos, entre outros. Um exemplo de aplicação de comportamento seqüencial é uma seqüência de manufatura ágil. Atualmente, sistemas de manufatura estão mudando de flexíveis para ágeis, onde estes sistemas seqüenciais que tenham fábricas geograficamente distribuídas podem operar em dois níveis: entre companhias ou internamente na empresa.

3.2.3.4 - Agentes Aprendizes e Adaptáveis

Estes agentes observam as ações dos usuários em *background*, analisam os comportamentos dos usuários e de outros programas, encontram padrões repetitivos, otimizando-os e por fim, automatizando-os da melhor forma possível. Assim, este agente necessita ser treinado antes para depois executar suas tarefas sem a assistência dos usuários.

3.2.3.5 - Agentes Reativos

Agentes Reativos não possuem modelos do mundo internamente. Estes respondem somente à estímulos externos. Assim como as redes neurais, os agentes reativos exibem comportamentos emergentes, os quais são resultados das suas próprias interações. Assim, estes agentes não têm previsão de nenhuma ação a ser realizada no futuro. Exemplos de aplicações para estes agentes seriam robôs limitados à percepção do mundo através dos seus sensores.

3.2.3.6 - Agentes Cognitivos

Estes agentes são baseados em modelos de organizações sociais, tais como grupos, hierarquias e comércios. Além disso, estes agentes representam explicitamente um ambiente ou membros da sociedade. Eles podem raciocinar sobre as ações tomadas no passado e planejar as ações a serem tomadas no futuro.

3.2.3.7 - Agentes Estáticos

Este tipo de agente também chamado de fixo ou estacionário é caracterizado por se manter em um mesmo *host*, ou seja, em um único nó da rede durante toda a sua existência. Eles são utilizados para provê serviços relativos aos recursos disponíveis no *host* em que eles residem. Eles também são utilizados em processos cooperativos com os usuários, provendo uma assistência pessoal para uma variedade de tarefas.

3.2.3.8 - Agentes Móveis

Este tipo de agente pode ser definido como processos de software que podem migrar entre sistemas computacionais dentro de uma rede, com o intuito de realizar tarefas para seus proprietários[BIGU98]. Algumas vezes faz sentido ter agentes trafegando através da rede. Outras vezes, esta tarefa pode causar muitos problemas. Um exemplo disso seria o caso de um agente conter algum conhecimento ou algoritmo exclusivo. Quando este for enviado para outros *hosts* na rede, isso divulgará as informações do mesmo. Neste sentido, manter este agente em um lugar seguro é a melhor das opções. Em se tratando de agentes móveis, é necessário ter a segurança que agentes maliciosos não invadirão o *host* em questão.

3.3. - Agentes Móveis

Agentes móveis são capazes de se mover em uma rede de computadores heterogênea, visando a execução progressiva das tarefas que foram desempenhadas para eles por seus proprietários.

A idéia de despachar o programa para uma execução remota é antiga. A submissão de esquemas pode ser exemplificada com o processamento em lote (*batch job*) enviados para os computadores de grande porte (*mainframes*) em meados dos anos 60[CARD99].

Assim como o termo **agente**, não há também um consenso em relação à definição do termo **agente móvel**. O fato de concordância é que um agente móvel se move dentro de uma rede de computadores carregando consigo o código e os dados. Segue abaixo algumas definições sobre agentes móveis:

"Agentes móveis são programas, tipicamente escritos utilizando *scripts*, que podem ser despachados do computador cliente e transportado para o servidor remoto para ser executado."

"Agentes itinerantes são programas que são despachados de um computador de origem, viajam entre servidores em uma rede até que se tornem hábeis para completar a sua tarefa; eles movem processos que progressivamente executam tarefas se movendo de um lugar para outro"[CHES95].

"Agentes transportáveis são capazes de suspender sua execução, se transportando para outro host na rede e inicializando a execução do ponto onde ele tinha suspendido"[KOTA94].

"Agente transportável é o nome de um programa que pode migrar de uma máquina para outra em uma rede heterogênea. O programa escolhe quando e para onde migrar. Ele pode suspender sua execução em um ponto arbitrário, se transportar para outra máquina e voltar a executar no novo *host*".

3.3.1 Os paradigmas Cliente/Servidor e Agentes Móveis

As figuras 3.2 e 3.3 a seguir mostram os conceitos básicos entre os paradigmas Cliente/Servidor e Agentes Móveis.

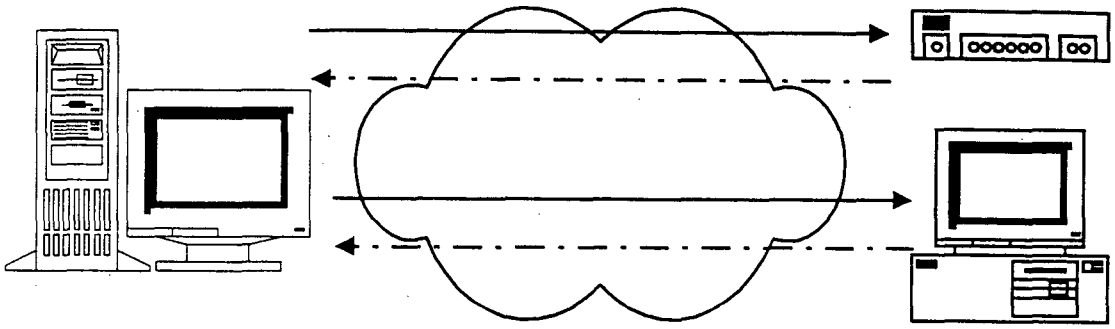

FIGURA 3.2 CLIENTE / SERVIDOR

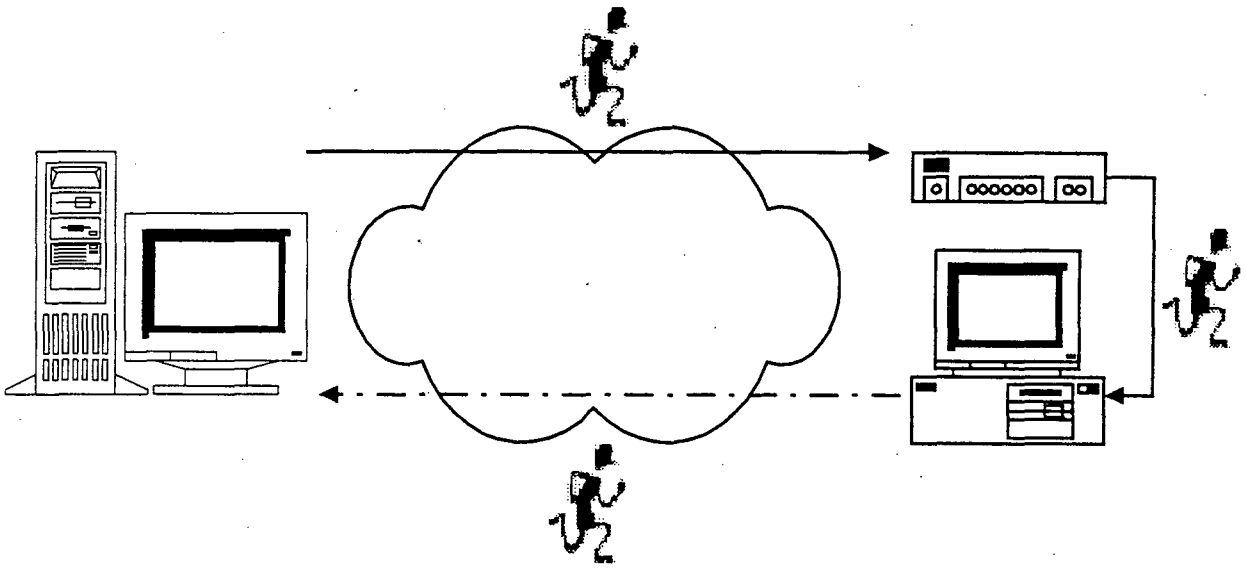

FIGURA 3.3 AGENTES MÓVEIS

Segundo Aridor & Lange, a tecnologia dos agentes móveis é uma tecnologia emergente que promete tornar sistemas distribuídos mais fáceis de projetar, implementar e manter. Agentes móveis não estão restritos aos sistemas que iniciaram a sua execução, pois eles têm a habilidade de se transportar de um sistema para outro através de uma rede. Este recurso permite um agente móvel se transportar para o

sistema que possui um objeto com o qual o agente deseja interagir, obtendo a vantagem de residir na mesma máquina ou rede do objeto.

3.3.2. - Vantagens dos Agentes Móveis

Estas vantagens são de acordo com Lange e Oshima [OSHI].

3.3.2.1 - Redução do tráfego de rede

Sistemas distribuídos demandam em grande volume de comunicação (interação) para realizar determinada tarefa, principalmente quando há restrições de segurança envolvida. Agentes móveis permitem reduzir o tráfego da rede, visto que despacham tarefas para que as mesmas possam ser executadas na máquina destino. 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".

3.3.2.2 - Ocultação da latência da rede

Sistemas críticos necessitam de respostas em tempo real para mudanças no ambiente. O controle deste sistema em uma rede substancialmente grande ocasiona uma latência inaceitável. Agentes móveis oferecem uma solução visto que são despachados pelo controlador central para executarem suas tarefas localmente.

3.3.2.3 - 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 nos protocolos 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.

3.3.2.4 - Execução assíncrona e autônoma

As tarefas podem ser embutidas em agentes móveis que são despachados pela rede. Após serem despachados os agentes são autônomos e independente da criação do processo, podendo executar assincronamente. Assim, os agentes podem se deslocar para outros nós da rede, executar suas tarefas, enquanto outras tarefas são desempenhadas pelos seus proprietários. Além disso, computadores móveis e dispositivos leves, tais como PDA (Assistentes Digitais Pessoais), normalmente se conectam de forma intermitente a uma rede. Aplicações-clientes em execução nestes equipamentos podem iniciar um agente móvel contendo a requisição de uma tarefa, despachar o mesmo pela rede durante uma rápida sessão de conexão e logo em seguida desconectar-se da mesma. A resposta pode vir em uma conexão realizada em um momento posterior.

3.3.2.5 - Adaptação dinâmica

Agentes móveis possuem a habilidade de perceberem mudanças no ambiente de execução e reagirem autonomamente. Múltiplos agentes podem interagir entre si e se distribuir pela rede, de modo a manter uma configuração satisfatória para resolver um problema em particular.

3.3.2.6 - Independência de plataforma

Redes de computadores geralmente são heterogêneas, tanto na perspectiva de hardware como 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. Para tal, os agentes móveis utilizam linguagens interpretadas, pois neste sentido permitem a interoperabilidade de múltiplas plataformas.

3.3.2.7 - Robustez e tolerância à falhas

A habilidade de agentes móveis reagirem dinamicamente às situações desfavoráveis, torna fácil a construção de sistemas distribuídos robustos e tolerantes à falhas. Se uma máquina será desligada, todos os agentes em execução na mesma serão

advertidos para que possam ser despachados e continuar suas tarefas em outras máquinas da rede.

3.3.3 - Infra-estrutura para Agentes Móveis

Os agentes móveis, para garantirem a mobilidade na rede e a realização das suas tarefas, necessitam ter uma infra-estrutura estabelecida. Neste sentido, esta infra-estrutura será composta de delegação, comunicação, mobilidade, gerenciamento dos recursos da máquina, gerenciamento de agentes, ambientes de execução, segurança, tolerância à falhas e interoperabilidade.

3.3.3.1 - Delegação

Os agentes são entidades que realizam tarefas para os usuários com o intuito de aumentar a performance do mesmo em relação aos sistemas computacionais. Porém, os usuários podem não confiar nos seus agentes. Com as pesquisas realizadas descobriu-se que associando cada vez mais inteligência a um agente, torna-se mais fácil dos usuários confiarem e delegarem suas tarefas aos mesmos. Alem disso, justamente por agentes atuarem baseado no interesse dos seus proprietários, eles carregam consigo a identificação de quem os criou.

3.3.3.2 - Comunicação

Os agentes devem se comunicar para que possam realizar as suas tarefas. A comunicação engloba três aspectos: comunicação entre agentes, com os ambientes de execução e com os usuários.

A comunicação entre agentes pode ser local ou remota, ou ainda, síncrona ou assíncrona. A comunicação entre agentes têm caráter de cooperação, ou seja, os agentes se comunicam com o objetivo de atingir metas comuns. O grupo DARPA *Knowledge Sharing Effort*, 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 Comunicação de Agentes (ACL), que satisfaz estas necessidades. A ACL pode ser melhor imaginada como sendo constituída de três partes - seu vocabulário, uma linguagem interna chamada KIF (Knowledge Interchange Format) e uma linguagem externa chamada KQML(Knowledge Query and Manipulation Language). Uma mensagem ACL é um expressão KQML na qual os argumentos são termos ou sentenças em KIF, formadas por palavras encontradas no vocabulário da ACL. O vocabulário da ACL é armazenado em um dicionário com as palavras apropriadas às áreas comuns das aplicações. Cada palavra tem notações formais (em KIF) e descrição informal em inglês. É um dicionário aberto, permitindo acréscimo de palavras nas áreas já existentes ou então de novas áreas que possam surgir. Tem-se então uma linguagem padrão de comunicação entre agentes [MILO99].

Em relação à KIF e KQML, enquanto KIF negocia com a representação do conhecimento, KQML se detém aos formatos das mensagens e as negociações entre os protocolos das mensagens enquanto os agentes estiverem em execução. Assim, KQML define as operações que os agentes irão obter das bases de conhecimento dos outros agentes, além de prover uma arquitetura básica com o intuito de compartilhar conhecimento e informações com agentes especiais, chamados *facilitadores*.

A utilização de KQML permite que os agentes se comuniquem através de mensagens. Estas mensagens realizam, implicitamente, alguma ação. Segue abaixo um exemplo da utilização da linguagem KQML para passagem de mensagens, onde requisita a um agente o preço de uma estação de trabalho da SUN.

```
( ask-one
  :sender      agenteOrigem
  :content     (real preço = sun.preço())
  :receiver    agenteDestino
  :reply-with  estoque-sun
  :language java
```

:ontology CHECAGEM)

Os agentes se comunicam com os seus ambientes para utilizarem serviços e recursos que estes possuem. Acesso à base de dados corporativas e ao Object Request Broker (ORB) baseado no padrão CORBA são exemplos importantes deste tipo de interação[MILO99].

Em relação à comunicação dos agentes com os seus proprietários e vice-versa, os mesmos podem usar mecanismos básicos como mensagens eletrônicas, para uma interação assíncrona e interfaces gráficas, por exemplo, perguntando ao usuário se ele confirma determinada ação[CARD99].

3.3.3.3 - Mobilidade

Um agente migra com o intuito de realizar a tarefa que foi a ele designada. A primeira medida que o agente deve tomar é selecionar a máquina destino que provê o serviço que ele necessita. Embora os agentes carreguem consigo seu itinerário fixo, eles, na maioria dos casos, devem decidir para onde vão. Mecanismos como pesquisas por palavras chaves, ou ainda rotas semânticas, que tentam encontrar o computador destino de um agente baseado em que tarefa ele deseja executar, podem auxiliar os agentes a encontrarem seus provedores de serviços.

O processo de mobilidade envolve dois passos. O primeiro, seria mandar o agente da origem para o destino e o outro seria receber o agente no destino. Para trafegar o agente da máquina origem para a máquina destino, é necessário suspender a execução do agente, no sentido de interromper a ação que o mesmo esteja desempenhando, codificar o agente para que seja transmitido e por fim liberar os recursos que o agente estava utilizando na origem, tornando estes disponíveis para outros usuários, agentes ou ambientes. Quando o mesmo chega no destino é necessário que, em primeiro lugar, o agente seja decodificado, verificando logo em seguida se o recurso que o agente necessita está disponível naquela máquina. Em caso positivo, o agente é excluído efetivamente do seu ambiente de execução original. Caso contrário se, por algum motivo, o agente não conseguir atingir a máquina desejado ou sofrer alguma

modificação externa, o mesmo retornará ao seu ambiente original, recompondo-se. Neste sentido, aspectos de segurança são envolvidos com o cuidado de garantir a integridade tanto do ambiente quanto do agente.

Para transmissão dos agentes, protocolos como TCP/IP, HTTP, X.25 e SMTP (protocolo de *mail*) podem ser utilizados como mecanismos de transporte dos agentes entre as máquinas com seu ambiente de execução.

3.3.2.4 - Gerenciamento de recursos das máquinas

A infra-estrutura de agentes móveis devem prover mecanismos de controle dos recursos disponíveis. Este tipo de controle permite uma utilização eficiente dos recursos e evitam que sejam manipulados inadequadamente pelos agentes. Os agentes podem conter informações relativas às necessidades dos seus recursos e a capacidade de utilização dos mesmos. Quando o agente é despachado para outro ambiente, os recursos por ele outrora utilizados, serão liberados.

3.3.2.5 - Gerenciamento de agentes

A infra- estrutura de um agente móvel deve controlar o ciclo de vida do mesmo dentro de um ambiente de execução. Cada ambiente deve ter controle desde a instanciação de um agente, até o momento que o mesmo é removido.

3.3.2.6 - Ambientes de execução baseado nas linguagens

Os ambientes de execução dos agentes permitem a execução de um agente escrito em uma linguagem de programação particularmente baseada em agentes. Compiladores e interpretadores compõem o núcleo dos ambientes de execução. Estes ambientes devem suportar várias linguagens de programação com o intuito de provê a interoperabilidade dos agentes.

Em relação à estrutura, as linguagens de programação baseadas em agentes podem ser classificadas em imperativas ou declarativas. As linguagens imperativas podem ser

procedurais ou orientadas à objetos. Enquanto que as linguagens declarativas são baseadas em declarações e mais utilizadas em agentes inteligentes.

Em relação ao modelo de execução, as linguagens podem ser compiladas, interpretadas ou ainda compiladas até um determinado ponto e depois interpretadas. Neste sentido, as linguagens compiladas podem ser utilizadas principalmente para agentes estáticos, visto que seus códigos binários são específicos para determinadas plataformas. Já os agentes móveis devem compilar e interpretar o código, pois devem ser independente de plataforma.

3.3.2.7 - Segurança

Em se tratando de agentes móveis, é necessário que políticas de segurança sejam impostas aos ambientes com o intuito de proteger os mesmos de ataques maliciosos de agentes. Isso pode ocorrer pois os agentes móveis têm autonomia suficiente para migrar para outras máquinas dentro de uma rede. É necessário verificar se, na entrada de um agente, o mesmo não é um vírus de computador. Para isso, algumas funções de segurança devem ser estabelecidas, como autenticação da origem do agente, checagem da autorização de execução do mesmo, controle de acesso aos recursos do ambiente, arquivamento em um *log* das ações dos agentes e checagem da integridade das informações. A autenticação da origem do agente permite que se evite identidades ambíguas da origem. Isso pode ser solucionado baseado na inclusão de um certificado de chave pública.

3.3.2.8 - Tolerância à falha

Os ambientes de execução dos agentes têm que prover algum mecanismo de recuperação do agente e da informação que este carrega no caso de ocorrer alguma falha no computador ou na rede. Este mecanismo deve reestabelecer o agente o mais rápido possível para que o mesmo possa voltar a desempenhar suas tarefas. Persistência é uma estratégia para suprir os requerimentos de tolerância à falha, visto que os agentes podem ser armazenados persistentemente e não na memória volátil.

3.3.2.9 - Interoperabilidade

Permite que se observe a interoperabilidade de um sistema, isto é um ambiente interagir com outro ambiente. Sistemas baseados em agentes estão crescendo rápidamente e com eles surgindo padrões tais como Foundation for Intelligent Physical Agents (FIPA), Agent Society e OMG's Mobile Agent Faciliy.

Alguns autores tratam esta infra-estrutura como ambientes de execução dos agentes ou chamam-no de agência. A característica principal de um agente móvel é a sua capacidade de transitar pela rede, e isso somente é possível na presença de uma infra-estrutura acima descrita.

3.4 - Linguagens dos Agentes

O desenvolvimento de agentes requer a utilização de dois tipos de linguagens. As linguagens de comunicação, que servem para garantir a comunicação entre os mesmos e as linguagens de programação, que permitirão o desenvolvimento dos agentes.

3.4.1 - Linguagens de Comunicação

Através destas linguagens, os agentes podem trocar conhecimentos através de mensagens. Atualmente, existem várias linguagens que se propõem a realizar esta tarefa. Dentre elas estão:

AgentTalk

Esta linguagem permite definir protocolos de coordenação incrementalmente e serem facilmente customizados para ajustar-se ao domínio da aplicação incorporando um mecanismo de herança.

ACL – Agent Comunication Language [MILO99]

Esta linguagem baseia-se na idéia da comunicação modelada através das trocas de sentenças declarativas (definições, suposições e gosto).

KIF – Knowledge Interchange Format

O KIF é caracterizada pòr uma linguagem formal para a troca de conhecimentos de programas que estejam em execução. Além disso, possui uma semântica declarativa, utilizada para representar o conhecimento sobre as diversas formas, dentre elas a representação de conhecimento, regras de raciocínio não monotônicas, dentre outras.

KQML – Knowledge Query Manipulation Language

A KQML permite a troca de informações e conhecimento. Esta linguagem pode ser utilizada por um agente para interagir com sistemas inteligentes ou ainda para a cooperação entre agentes. É caracterizada, como citada anteriormente, como uma linguagem com características externas de interação entre agentes.

3.4.2 - Linguagens de Programação

As linguagens de programação permitem que se desenvolvam os agentes que desejam criar e manipular. A seguir são relacionadas algumas destas linguagens.

Java

É uma linguagem orientada a objeto que tem características herdadas do C++ e do Smalltalk. Além disso, o sistema é composto por um compilador e um interpretador. A linguagem é compilada até gerar os *bytes codes*. Logo em seguida, a mesma é interpretada no sentido de garantir a independência de plataforma.

LALO

Esta linguagem é orientada a agentes, além de ser um ambiente para desenvolvimento de sistemas multiagentes. Um programa desenvolvido em LALO é gerado em código C++, sendo depois compilado e utiliza como linguagem de comunicação, KQML.

OBLIO

Esta é linguagem interpretada e orientada a objetos, tendo suporte à computação móvel, tal como agentes móveis.

PHANTOM

Esta também é uma linguagem interpretada, projetada para aplicações distribuídas, interativas e em larga escala.

TELESCRIPT[RICC99]

Esta linguagem orientada à objetos permite que se divida os processos em agentes locais, permitindo assim a comunicação entre os processos.

3.4.2.1 - Plataformas baseadas em Java

A linguagem Java viabilizou a concepção de diversos sistemas, também denominados plataformas ou ambientes de execução, baseados em agentes móveis. Segue abaixo algumas citações.

Aglets

Aglets são agentes autônomos baseados em Java e que foram desenvolvidos pela IBM. 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 a recepção do aglet, como citado anteriormente. Esta aplicação aglet, também denominado de servidor e nomeado Thaithi, permitirá a criação de uma plataforma neutra para os aglets.

FTP Software Agent Tecnology

Este é designado para gerenciar uma rede heterogênea através da Internet. Os agentes são autônomos e móveis, e podem se mover para qualquer sistema que tenha um agente correspondente instalado. Assim, na medida que o agente migra de sistema para sistema, as tarefas podem ser modificadas de acordo com o sistema visitado.

Este agente não interage com o usuário no processo de modificar ou confirmar as suas tarefas. Os FTPs podem se mover de sistemas para sistemas, realizando as tarefas que foram previamente definidas pelo usuário. Duas aplicações estão disponíveis utilizando esta tecnologia de agentes: IP Auditor e IP Distributor. A aplicação do IP Auditor utiliza agentes para coletar informações sobre hardware, software e configurações de sistemas em uma rede. A aplicação IP Distributor tem por finalidade distribuir software e pode também executar funções dentro de uma rede.

VOYAGER

O VOYAGER, da *ObjectSpace Inc.* é um agente que foi desenvolvido totalmente em Java. Utiliza o ORB (*Object Request Broker*), acrescentando as capacidades dos agentes as suas características. Os agentes Voyager têm mobilidade e autonomia, o que é disponibilizado em uma classe básica *Agent*.

Um agente pode se mover de uma localidade para outra, além de previamente redirecionar as futuras mensagens para as localidades que eles irão a posteriori. Assim

como os aglets, estes agentes também têm o itinerário a ser seguido assim como as ações que serão executadas em cada máquina.

ODYSSEY

Este agente, desenvolvido pela General Magic, foi criado baseado em subclasses da classe Agent do Java. Cada agente executa sua *thread* particular e pode se transportar de um lugar para outro. Em cada lugar novo que o agente chega, a *thread* deve ser reinicializada. Uma instância da classe *Place* permite que se defina o ambiente de execução deste agente. A classe *Worker* é uma especialização da classe *Agent*, sendo a mesma responsável pelo armazenamento do itinerário, que contém as tarefas a serem executadas, assim como os hosts destinos. Há também a classe *Petition*, que é responsável pela identificação de outros agentes com os quais o Odyssey irá se comunicar.

O Odyssey foi desenvolvido inteiramente em Java. Utiliza o RMI como transporte, mas pode ser reconfigurado para utilizar o DCOM (Distributed Component Model) da Microsoft ou ainda o IIOP (Internet InterORB Protocol), protocolo este que permite a comunicação inter ORBs.

JATLite – Java Agent Template Lite

Desenvolvido pela Universidade de Stanford com o intuito de minimizar a utilização dos pacotes Java, também pode ser utilizado para a construção de sistemas multiagentes. É composto por uma arquitetura em camadas, na qual a mais inferior provê somente classes abstratas que podem suportar qualquer protocolo de comunicação. A camada básica utiliza o TCP/IP como mecanismo de comunicação e permite que o desenvolvedor do agente defina qual será o protocolo para a troca de mensagens. A camada KQML trata o suporte aos agentes que utilizam as mensagens do KQML. A camada de roteamento adiciona mensagens que chegam para agentes que não estão disponíveis em uma fila. Este ambiente do JATLite pretende desenvolver agentes

autônomos que se comunicam usando protocolos ponto-a-ponto. Além disso, suporta também a passagem de mensagens tanto síncronas quanto as assíncronas.

Utilizando todos estes conceitos absorvidos anteriormente, surgem motivações baseadas nos agentes móveis para a aplicação dos mesmos. Atualmente, estes tipos de agentes têm motivado muitas áreas de pesquisa devido a suas vantagens, anteriormente abordadas. Diante deste contexto, a aplicação de agentes móveis no comércio eletrônico, na coleta de dados em locais remotos, na monitoração de informações remotas, na gerência de redes e sistemas distribuídos, além de ser empregado na computação móvel está sendo cada vez mais pesquisada despertando o interesse em novas implementações no mercado.

4. A Integração dos Bancos de Dados

Este capítulo aborda a integração dos bancos de dados, caracterizando os aspectos de implementação de um ambiente a seguir elucidado.

Devido ao constante crescimento da informática no mundo, diversos ambientes que outrora estavam ativos, absorvem uma nova conotação. Com isso, novas aplicações, novas bases de dados, novos ambientes de desenvolvimento vão crescendo comercialmente e mudando o paradigma do desenvolvimento e acesso à base de dados.

A dificuldade deste processo se encontra principalmente nos custos que a organização deve despender com o treinamento de pessoal, permitindo que os antigos empregados possam usufruir das novas tecnologias; desperdício de tempo, no sentido da migração dos sistemas antigos para os mais atualizados; além da reestruturação e modelagem que os analistas de sistemas devem realizar, garantindo assim uma migração satisfatória no final do processo.

Além disso, com o advento da Internet e o crescimento irredutível dos seus usuários, torna-se clara a necessidade de desenvolver aplicações para esta grande rede. Diante deste contexto, surgem as aplicações distribuídas se utilizando da arquitetura de *n-camadas*. Aplicações distribuídas permitem que se distribua objetos facilitando a sua utilização e reutilização, absorvendo as características essenciais da orientação a objetos. Além disso, estas aplicações permitem que um mesmo objeto seja um servidor e cliente, sendo assim caracterizado por múltiplas camadas de acordo com os requisitos

dos objetos em uma aplicação. Perante esse novo mundo da distribuição e utilização da Internet como meio de comunicação, surgem os agentes móveis, como um novo paradigma para aplicações distribuídas, que permitem utilizar tecnologias orientadas a objetos para migrar na rede em busca de informação relevante.

Assim sendo, o ambiente desenvolvido objetiva a integração de bases de dados distintas, utilizando para isso a abordagem de agentes móveis, onde o mesmo obtém a informação, transpondo o código localmente. Com a utilização de agentes móveis, a busca de informação em base de dados distintas é transparente para o usuário, o que facilita a absorção da informação necessária.

4.1 – O Ambiente da Mobilidade

O ambiente é constituído por uma rede, onde a mesma pode ser local ou a Internet, fator irrelevante em se tratando de aplicações distribuídas, composta por três máquinas que contém as bases de dados e uma máquina responsável pela inicialização do agente móvel. Desta forma, um único cliente é o responsável pelo encaminhamento do agente móvel, sendo este o gerenciador das demais tarefas realizadas. Com este ambiente, é mais fácil garantir a interoperabilidade das bases de dados, independente do usuário estar a par da situação ou mesmo ciente da máquina onde o agente se encontra, sendo importante para o mesmo, o resultado da pesquisa requisitada.

O ambiente tem a seguinte estrutura, como mostra Figura 4.1.

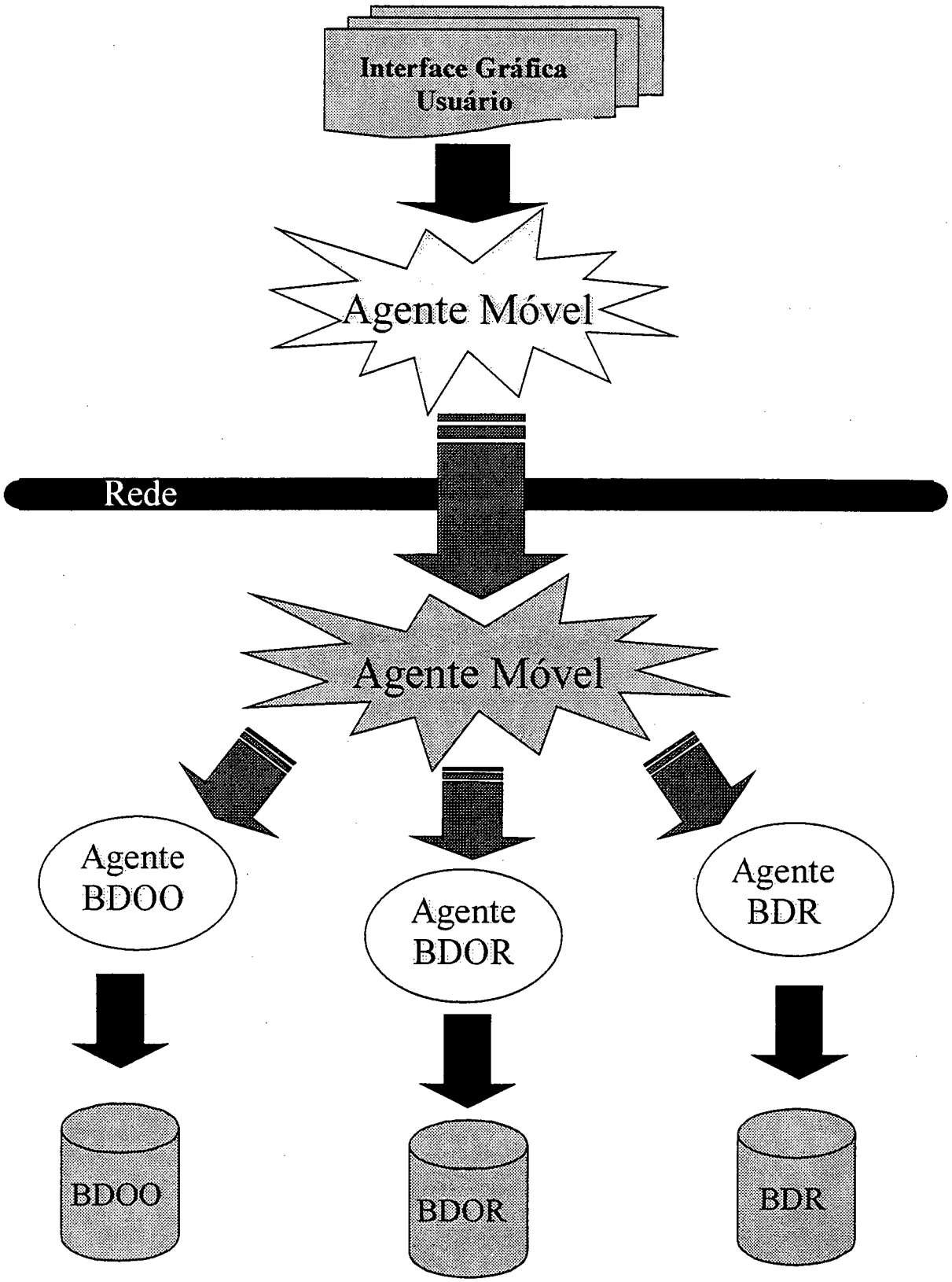

FIGURA 4.1 O AMBIENTE DA MOBILIDADE

As máquinas que compõem este ambiente são caracterizadas por computadores pessoais (PC) interconectados por uma rede, na qual é realizado o movimento do agente móvel.

O usuário tem interação com este ambiente através de uma interface gráfica, que é responsável por disponibilizar os campos por onde o usuário realiza a pesquisa desejada. Uma vez determinado os campos para seleção, o usuário dispara o agente móvel através da interface gráfica, permitindo a inicialização do mesmo, que pode ser local ou remotamente. Neste sentido, a interação do usuário com o agente móvel ocorre através da interface gráfica, sendo transparente as ações do agente dentro da rede. Além disso, na máquina do usuário somente tem a presença da interface gráfica, minimizando assim a utilização dos recursos localmente.

Uma vez inicializado, o agente móvel é deslocado para uma máquina, onde ocorre uma publicação do mesmo e, consequentemente, dos seus métodos. Dentro desta máquina, o agente móvel inicia a criação dos agentes estáticos responsáveis pelo acesso à base de dados nas máquinas presentes no itinerário. Essas bases de dados, por sua vez, podem estar situadas em máquinas distintas, ou seja, uma máquina servidora para cada banco de dados, assim como podem estar situadas em uma única máquina contendo as três bases de dados. Situadas em uma única máquina servidora, o agente móvel efetiva a instanciação do primeiro agente estático presente no itinerário, enquanto que os demais seriam criados e inicializados localmente, não sendo mais necessário outra conexão com a rede. Neste caso, a conexão com a máquina servidora seria singular, minimizando o tráfego da rede e o deslocamento do agente seria duplo, no sentido que o mesmo se deslocaria da máquina onde foi criado para esta estação-destino, e retornaria para a sua máquina gestora. Porém, nem sempre os bancos de dados de uma organização estão situados em uma única máquina. Neste caso, o agente deve criar uma conexão para cada servidor contendo o agente estático e se desloca para a primeira máquina presente no seu itinerário, comunicando-se com o agente estático local. A partir desta comunicação, o agente móvel realiza a tarefa que outrora fora determinada e se encaminha para a próxima máquina presente no seu itinerário, restabelecendo uma nova conexão com a máquina destino.

Com essa estrutura, o acesso dos usuários às bases de dados é restrito, visto que as estações-clientes não têm acesso direto aos banco de dados, prevenindo a danificação de informações relevantes, e ou a má manipulação das mesmas, evitando, consequentemente, a inconsistência do ambiente. Além disso, os agentes foram particionados em agente móvel e agentes estáticos, com o intuito de aumentar a performance do ambiente, visto que esta separação modificou o código fonte dos agentes que trafegam pela rede. Caso contrário, o agente móvel carregaria consigo todo o código de acesso às bases de dados, aumentando o seu conteúdo e, consequentemente, o tráfego das informações que passariam na rede. Assim, neste ambiente o agente móvel é responsável por enviar as requisições dos usuários obtidas através da interface gráfica e receber as respostas dos agentes estáticos, estes responsáveis por realizar o acesso às respectivas bases de dados.

Outra vantagem verificada nesta organização ocorre no caso de haver falha na conexão com a rede ou erro no acesso a uma determinada base de dados. Neste caso, o resultado das informações não será afetado, pois o agente móvel só trará as informações onde a comunicação com o agente estático for bem sucedida. Porém, se o acesso a uma das três máquinas não tenha ocorrido de forma satisfatória e os dados não possam ser obtidos, o agente móvel traz as informações das outras bases de dados, realizando assim a sua meta, apesar da mesma ter sido parcial.

4.2. – A Plataforma de Execução dos Agentes

O plataforma de execução é o principal responsável pela mobilidade do agente dentro da rede. É necessário que o ambiente seja propício para o agente para que o mesmo possa desempenhar suas tarefas. Nas diversas plataformas analisadas, estes ambientes são chamados de Servidor. Assim, o Concordia tem o servidor Concordia, o Voyager tem seu servidor que é executado escutando em uma porta pré-definida. O

Aglets da IBM, tem o Thaithi, que também é responsável por prover os serviços necessários para a execução de um agente.

A plataforma utilizada neste trabalho foi o Voyager da *ObjectSpace* devido as suas características de manipulação dos agentes, facilidade de integração com a linguagem Java e difusão das funções para os desenvolvedores. O servidor do Voyager provê os serviços necessários para as manipulações dos agentes, bem como o controle do seu ciclo de vida, dos relacionamentos e principalmente da sua execução. Neste caso, é necessário executar este servidor em cada máquina presente no itinerário do agente móvel. Assim, na estação onde o agente for criado deve ser inicializado o ambiente de execução, além dos respectivos servidores em cada máquina que contiver os bancos de dados. Dessa forma. o agente é direcionado de acordo com o endereço IP da máquina em questão e com a porta correspondente, visto que em uma única máquina podem ter dois servidores escutando portas distintas.

Além disso, torna-se necessário ressaltar que servidores de diferentes plataformas ainda não permitem a interoperabilidade. Neste caso, não é possível, inicializar um agente utilizando o Thaithi da IBM, e destiná-lo para outra estação que esteja sendo executado o servidor Concordia.

Os servidores do Voyager não necessariamente precisam ser executados manualmente nas máquinas presentes no itinerário. Os mesmos podem ser inicializados no próprio agente móvel antes de iniciar a criação dos agentes estáticos. Porém, estes servidores devem fazer referência às máquinas que os agentes estáticos irão ser executados, sendo fornecidos para tal o endereço IP da máquina e a porta de comunicação.

Assim, a plataforma de execução simula, tipicamente, uma sub-camada, permitindo a mobilidade dos agentes, que juntamente com a linguagem Java, garantem a independência do ambiente.

4.3. - Agente Móvel

A utilização de um agente com características de mobilidade permite uma melhor utilização do tempo disponível do usuário para realizar a pesquisa, visto que o mesmo pode disparar o agente e somente avaliar os resultados, sem se preocupar com as conexões ás bases de dados, nem com o acesso aos dados, sendo estes designados pelos agentes estáticos. Além disso, o agente móvel torna transparente todo o caminho percorrido pela rede, assim como a sua migração e comunicação, sendo somente revelado o cumprimento da meta designada pelo usuário.

Uma vez instanciado local ou remotamente, e realizado suas tarefas na máquina específica, o agente móvel é publicado para que o mesmo possa se comunicar com os agentes estáticos posteriormente. A partir de então, o agente móvel prepara-se para o início do seu itinerário, caminho entre estações que o mesmo irá passar, desempenhando suas funções com o objetivo de atingir a meta a ele designada. De posse do itinerário, o agente instanciará os agentes estáticos, disparando-os nas respectivas máquinas. Com intuito de desempenhar as suas funções de pesquisa, o agente móvel deve carregar consigo dois objetos. O primeiro que é responsável por enviar aos agentes estáticos os requisitos necessários para a execução da pesquisa e o segundo que é responsável por armazenar os resultados da mesma. O objeto que conterá os requisitos será fornecido pelo usuário, após a criação do agente, e terá como conteúdo os dados dos campos presentes na interface gráfica, pelo qual o cliente deseja que a pesquisa seja realizada. Assim, após a criação do agente, a primeira tarefa que o mesmo irá desempenhar será buscar da interface gráfica os dados para atingir seus objetivos. Para isso, o agente deverá invocar um método presente na sua interface que realize esta tarefa. Com a posse deste objeto, o agente invoca outro método que será responsável pela sua publicação, permitindo assim a comunicação do agente móvel com os agentes estáticos. Uma vez publicado, o agente verifica as instâncias dos agentes estáticos e efetiva as criações

destes nas respectivas máquinas que contém os bancos de dados. Após a comunicação com o agente estático, que terá como objetivo passar este objeto contendo as requisições obtidas no cliente, o agente móvel aguarda um comunicado do estático para obter os resultados. No momento que o agente estático contiver o resultado do seu processamento, o mesmo invoca uma referência do agente móvel que está publicado e envia o resultado. Este resultado fornecido como retorno é um novo objeto responsável pelo armazenamento do resultado da pesquisa em questão. Uma vez obtido o resultado de uma base de dados, o agente móvel continua aguardando os novos envios dos resultados referentes aos outros bancos de dados. Caso o resultado obtido seja vazio ou ainda tenha ocorrido algum erro no processo de obter as informações, o agente móvel despreza este resultado e continua esperando uma comunicação do próximo agente estático. Assim, no final do itinerário, o agente móvel tem todos os dados resultantes da pesquisa requisitada. Cada resultado é obtido através da comunicação com os agentes estáticos que realizam os acessos às bases de dados relacional, objeto-relacional e orientadas a objetos, sendo finalmente adicionados a um objeto comum. Chegando na máquina de origem, o agente móvel é responsável por enviar as informações obtidas ao cliente, completando assim sua meta de execução.

O agente móvel foi desenvolvido utilizando o Voyager juntamente com a linguagem Java e o seu itinerário foi pré fixado de acordo com as máquinas que foram disponíveis para a execução dos bancos de dados relacional, objeto-relacional e orientado a objeto. Além disso, a comunicação do agente móvel com o agente estático é baseada na linguagem Java e o protocolo de rede utilizado é o TCP/IP, sendo que a comunicação ocorre através da invocação dos métodos presente no agente estático e no agente móvel. Esta invocação dos métodos só é possível devido a publicação do agente estático em uma área reservada e da referencia por parte do agente estático ao agente móvel mediante a necessidade de executar um método, por exemplo, o método de retorno das informações obtidas através do banco de dados.

4.4. – Agentes Estáticos

Os agentes caracterizados como estáticos, são em número de três, de acordo com a quantidade de acesso à base de dados existente. Para cada banco de dados específico é instanciado um novo agente estático, que contém as funções necessárias para o acesso aos mesmos.

No momento de instanciação do agente estático, fato este que ocorre depois da publicação do agente móvel, este envia para o agente estático o objeto contendo as requisições da pesquisa do usuário. De posse das requisições, os agentes estáticos realizam uma consulta à base de dados utilizando para isso as funções referentes ao banco em questão. Como estas funções são divergentes, torna-se necessário a implementação de um agente estático para cada tipo de banco de dados.

Cada agente estático é responsável por realizar a conexão com a base de dados, seja ela relacional, objeto-relacional ou orientada a objeto. Após esta conexão, os agentes estáticos utilizam classes específicas que permitem a consulta às tabelas ou aos objetos, em se tratando do banco de dados orientado a objetos. Em se tratando do resultado desta consulta, os agentes estáticos utilizam novas classes que permitem a manipulação do conteúdo encontrado. Estas classes divergem de acordo com o banco de dados utilizado, assim como os comandos utilizados pelas linguagens de consulta . Os três bancos trabalham com linguagens de consultas, o relacional utiliza o SQL-92, o objeto-relacional utiliza o SQL-3 e o banco de dados orientado a objetos utiliza a OQL.

A distinção dos agentes em móvel e estáticos, permite uma separação das metas dos mesmos, sendo os móveis responsáveis pela comunicação e delimitação do ambiente e os agentes estáticos pela consulta e obtenção efetiva dos resultados diretamente das bases de dados. Os agentes estáticos ainda foram separados em três de

acordo com as bases de dados, o que especifica seus objetivos, permitindo assim uma distinção dos acessos a cada uma das bases de dados.

4.5. – Bases de Dados

A utilização de três tipos distintos de banco de dados ocorre pelo fato de que, em média 75% das empresas do mercado comercial ainda trabalham com banco de dados relacional, utilizando assim o padrão SQL-92 definido pela ANSI como principal meio de acesso aos dados. Outra pequena porção do mercado está migrando seus sistemas para a utilização de um banco de dados objeto-relacional, o qual incorpora conceitos orientados a objetos, armazenando-os relacionalmente, ou seja, utilizando tabelas. Cerca de 2% das empresas que utilizam banco de dados relacional estão fazendo uso da orientação a objetos como forma de armazenamento das informações. Assim, devido ao crescimento do paradigma orientado a objeto e novas formas de acesso a estas bases de dados, tanto o objeto-relacional, quanto o orientado a objeto, torna-se necessária a integração de bases de dados distintas.

Os bancos de dados foram projetados contendo estruturas semelhantes, diferindo apenas no tocante a sua forma de organização. Em relação ao relacional, são analisadas e estruturadas tabelas, contendo as chaves primárias e chaves estrangeiras, quando houver necessidade. Em relação ao banco de dados objeto-relacional, sua forma de armazenamento se diferencia um pouco do relacional, visto que há o conceito dos tipos abstratos de dados. Assim, as tabelas são criadas baseadas em objetos, que são os tipos abstratos. Além disso, quando há um relacionamento, ou seja, uma referência a uma coluna de outra tabela, a referência ocorre baseado no objeto e não em uma coluna específica como se observa no relacional. A consulta aos dados de uma tabela que não envolve referência a outros objetos ocorre de forma semelhante ao banco de dados relacional. Porém, caso seja necessário alguma consulta com uma referência a um outro objeto, utiliza-se a própria nomenclatura da orientação a objeto, podendo ainda ser

invocados métodos de acesso. O banco de dados orientado a objeto é estruturalmente semelhante aos tipos abstratos de dados, porém com uma única ocorrência do objeto. Assim, caso haja a necessidade de obter um conjunto de dados, é necessário especificar que aquele objeto pode ser manipulado como conjunto, caso contrário o mesmo só poderá ser recuperado individualmente. Além disso, não é necessário nenhum atributo referente a identificação do objeto, pois o mesmo será realizado através do OID (Object ID), criado no momento da instanciação do classe. Ainda assim, neste tipo de banco de dados, será necessário definir os métodos inerentes a cada classe que for desenvolvida.

A princípio, foi designado que o Jasmine seria o banco de dados orientado a objeto, porém, devido a sua completa integração com a linguagem C++ e ainda não permitir a interoperabilidade com *front-ends* separados, este banco foi descartado. Diante deste problema, e uma posterior averiguação do mercado americano, o próximo banco de dados escolhido foi o ObjectStore, da ODI. Mas, a não utilização de uma interface gráfica para a manipulação dos objetos e classes, assim como negligências quanto a administração dos usuários e o seu gerenciamento resultou no descarte desta opção. Assim sendo, o POET foi o banco de dados escolhido para a utilização do conceito da orientação a objeto, visto atender as citadas necessidades e seguir o padrão estabelecido pela ODMG (*Databse Object Management Group*) em relação aos bancos de dados.

Em relação a base de dados objeto-relacional, o Oracle8i foi o utilizado para o desenvolvimento deste ambiente, por presenciar experiências satisfatórias com o mesmo, utilizando o SQL3. E em se tratando de banco de dados relacional, o SQL Server 7.0, da Microsoft foi utilizado pela experiência adquirida no uso deste banco de dados.

5. Implementação do Ambiente

5.1 – Plataforma utilizada

A implementação tem como objetivo principal exemplificar as características da mobilidade do código, elucidando as etapas essenciais para desenvolvimento do trabalho. Diante disso, foi necessária a escolha de um ambiente de execução, como anteriormente citado, para que o agente possa desempenhar as suas tarefas. Após a avaliação de diversas plataformas, dentre elas os Aglets, Grasshopper, Concordia e Voyager, este último foi o escolhido devido, principalmente, a disponibilidade dos recursos e as características de manipulação dos agentes. A facilidade de integração com a linguagem Java foi outra característica importante em relação ao Voyager, pois a utilização deste ambiente com agentes que podem transitar pela rede, torna-se evidente, devido à independência de plataforma inerente à linguagem.

5.2 – Configuração das máquinas

Uma vez escolhido o ambiente de execução do agente, é necessário preparar a máquina para que o mesmo possa ser executado. A princípio é necessário que se tenha instalado o JDK (Java Developers Kit) na versão superior ao 1.2. A versão superior ao 1.2 é exigida devido a integração desta linguagem com o POET (banco de dados orientado a objetos) e com o próprio Voyager, na versão 3.2 que permite utilizar mais recursos e aprimorar o desenvolvimento. Além disso, é necessário que o Voyager esteja

instalado em todas as máquinas que o agente irá percorrer, visto que o mesmo é o responsável por iniciar o ambiente de execução.

Após esta etapa de instalação, o Voyager deverá ter alterado o arquivo *autoexec.bat*, em se tratando de máquinas com o sistema operacional Windows95 ou as variáveis de sistema, quando o sistema operacional for o WindowsNT. Assim, duas novas linhas deverão ter sido acrescentadas:

SET PATH=%PATH%;C:\voyager\bin;

SET CLASSPATH= %CLASSPATH%;.;C:\voyager\lib;

Além destes caminhos padrões que a própria plataforma altera no autoexec.bat ou nas variáveis de sistema, há a necessidade de se criar um diretório onde as classes que o ORB(*Object Request Broker*) do Voyager referencia estejam contidas. Isso ocorre pelo fato de se estar utilizando uma tecnologia do Voyager denominada de *NameSpace*. Neste caso, este diretório com as classes que se referenciam ao *NameSpace* devem também estar presente no CLASSPATH como a linha que segue abaixo:

SET CLASSPATH= %CLASSPATH%;C:\mestrado\drone;

Quando a plataforma de execução é inicializada manualmente, é necessário executar o seguinte comando dentro da tela do DOS:

voyager 7000

Sendo 7000 o número da porta de comunicação dos agentes. Após realizada esta tarefa, o ambiente de execução estará em funcionamento, assim como demonstrado na figura 5.1.

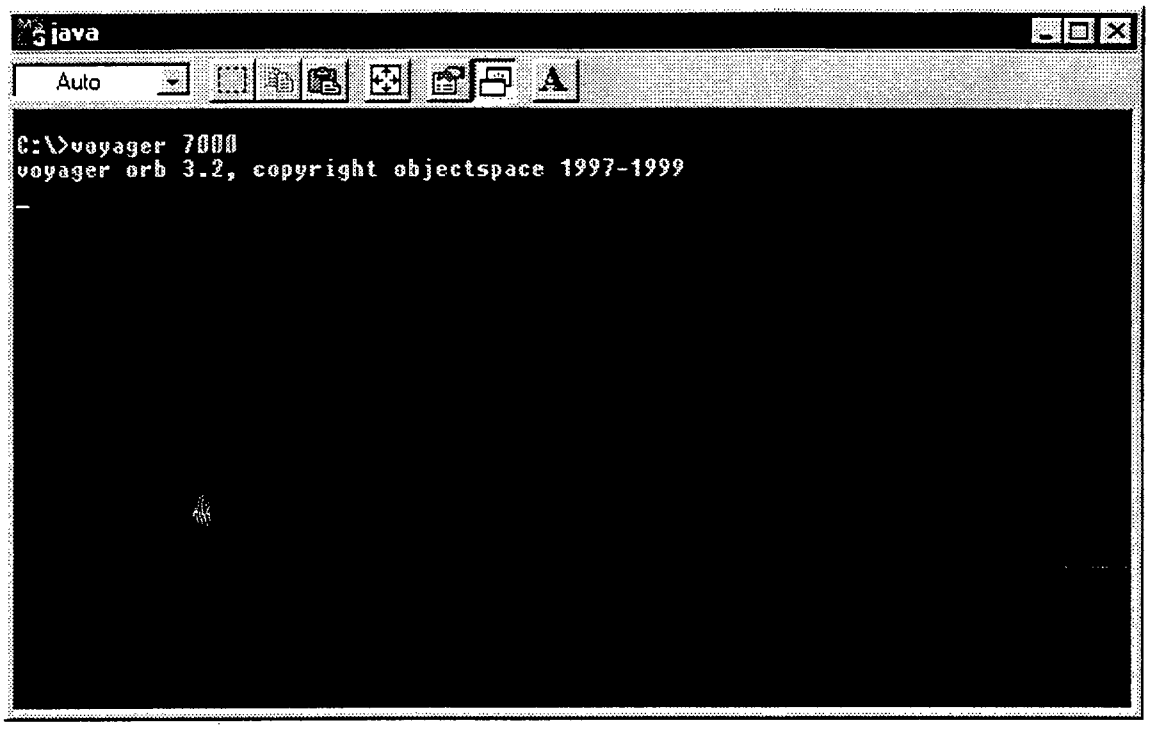

FIGURA 5.1 AMBIENTE DE EXECUÇÃO DO VOYAGER

5.3 - O Usuário

O usuário deste ambiente é o principal responsável por disparar o agente móvel para realizar a pesquisa de acordo com as necessidades. Para tal, é apresentada uma interface gráfica, onde o mesmo seleciona a informação que deseja pesquisar, disparando logo em seguida o agente para o seu itinerário. A partir dos dados selecionados pelo usuário como requisitos para a pesquisa das informações, os agentes fazem a conexão com as bases de dados e obtém os resultados. A interface gráfica está ilustrada na figura 5.2.

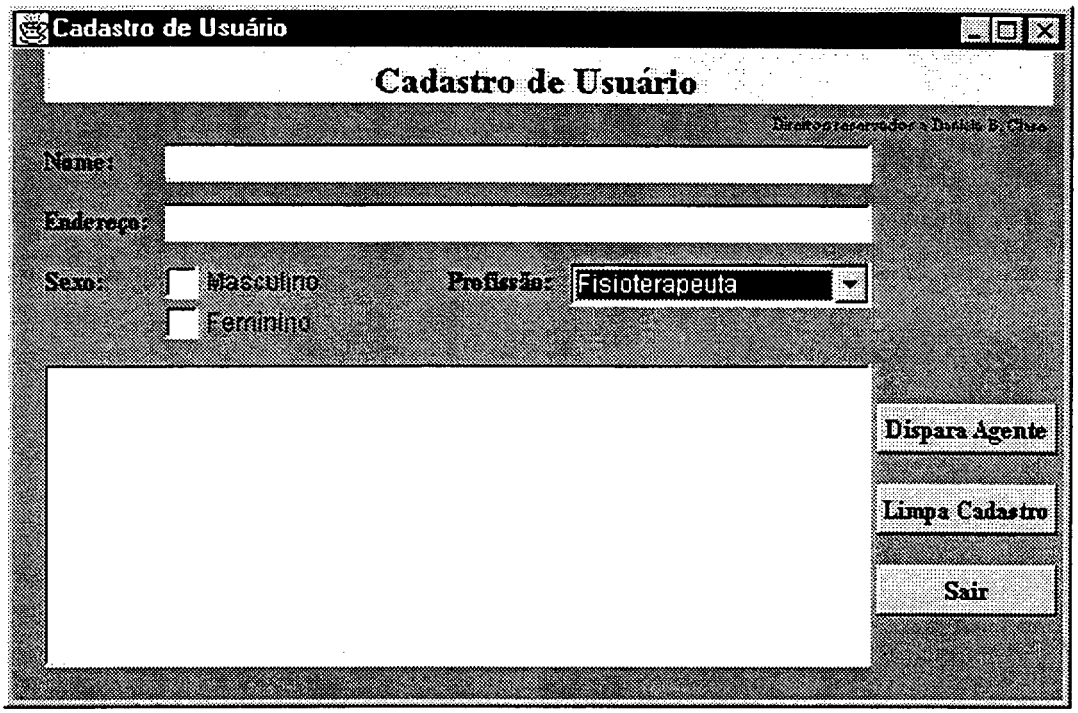

FIGURA 5.2 INTERFACE GRÁFICA PARA REALIZAR A PESQUISA COM O AGENTE

No exemplo elucidado acima, pode-se observar que o agente móvel é inicializado e tem como requisito buscar todos os usuários que tenham como profissão a fisioterapia.

Internamente, para o agente móvel ser criado em uma máquina remota ou em uma máquina local, é necessário que antes seja definida a sua interface de comunicação, ou seja, os métodos que podem ser executados por objetos remotos. Esta interface é mostrada na figura 5.3.

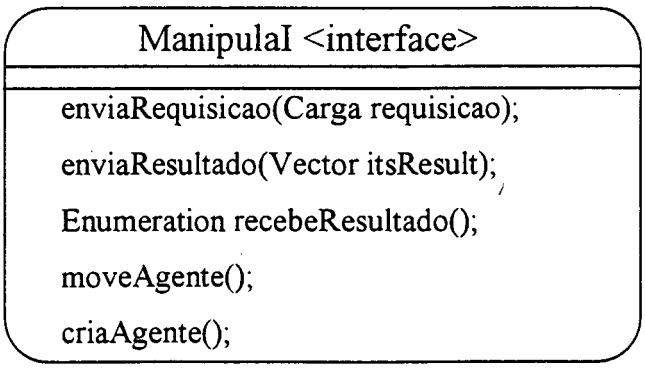

FIGURA 5.3 INTERFACE MANIPULAI

Com a utilização desta interface, a criação do agente ocorre da seguinte forma:

Voyager.startup(); //Estabelece a conexão com ambiente
ManipulaI agMovel = (ManipulaI) Proxy.export(new AgenteMovel(), "//192.168.0.2:6000");

No momento de instanciação do agente, o mesmo já é exportado para ser publicado, sendo logo em seguida associado a um nome, pelo qual posteriormente será referenciado.

Namespace.bind("//192.168.0.2:6000/NASDAQ", agMovel);

O agente móvel será referenciado pelo agente estático quando houver necessidade, pelo nome de "NASDAQ". Uma vez instanciado, o agente móvel obtém as requisições da interface gráfica do usuário e as publica no agente móvel em uma área reservada denominada de *NameSpace*. O envio das requisições dos usuários para o agente móvel ocorre através da invocação do método abaixo, tendo como parâmetro o objeto requisição da classe Carga, elucidada logo a seguir.

agMovel.enviaRequisicao(requisicao);

Uma vez de posse das requisições, o agente móvel está apto para iniciar o seu itinerário, sendo assim invocado o método que permite o seu deslocamento.

agMovel.moveAgente();

Após todas as pesquisas nas bases de dados e comunicações entre os agentes, há uma desvinculação da plataforma de execução do Voyager, sendo para isso necessário a execução do seguinte comando:

Voyager.shutdown();

A classe ***Carga,*** mostrada na figura 5.4, é utilizada pelo objeto requisição passando todos os parâmetros contidos na interface gráfica, independente de serem nulos ou não. Assim, o objeto *Requisicao* contém os dados fornecidos para a pesquisa no bancos de dados.

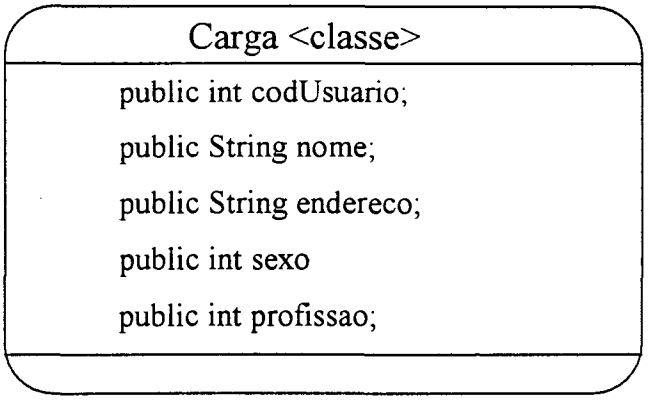

FIGURA 5.4 CLASSE CARGA

5.4. – O Agente Móvel

A classe ***AgenteMovel*** é o efetivo agente que trafega através da rede em busca da informação que o usuário requisitou. Para isso, a mesma herda características de uma classe do Voyager denominada *com.objectspace.voyager.agent.Agent*, fornecendo assim funções inerentes a um agente, como sua mobilidade.

Além disso, esta classe ***AgenteMovel***, deve implementar duas outras interfaces, a *Manipulal*, previamente caracterizada, e a *Serializable*, que fornece os subsídios necessários para o transporte dos agentes entre máquinas na rede.

Uma vez instanciado, o agente móvel designa a criação dos agentes estáticos nas respectivas máquinas que os mesmos irão executar os acessos às bases de dados. Logo em seguida, é enviado para os agentes estáticos o conteúdo das requisições dos usuários, para que os mesmos possam desempenhar as suas funções. Isso ocorre para todos os

agentes estáticos que estiverem presentes no ambiente e cuja objetivo seja realizar a consulta a uma base de dados. Uma vez realizada as tarefas dos agentes estáticos, estes retornam um resultado para o agente móvel, que o recebe em forma de vetor, armazenando as informações obtidas depois do acesso ao banco de dados.

Após a passagem do AgenteMovel pelo banco de dados relacional, o mesmo se desloca para a próxima máquina contida no seu itinerário. Uma vez que haja a instanciação dos agentes estáticos nas respectivas máquinas, ocorre o mesmo processo de envio dos requisitos e obtenção dos resultados por parte do agente móvel.

De posse dos resultados provindos dos agentes estáticos, o agente móvel retorna a sua máquina de origem, apresentando os resultados na interface gráfica. Isso ocorre mediante a execução do método *recebeResultado()* que retorna um vetor contendo todas as informações obtidas, tendo o agente assim cumprido a sua meta. Segue abaixo a figura 5.5 com os resultados.

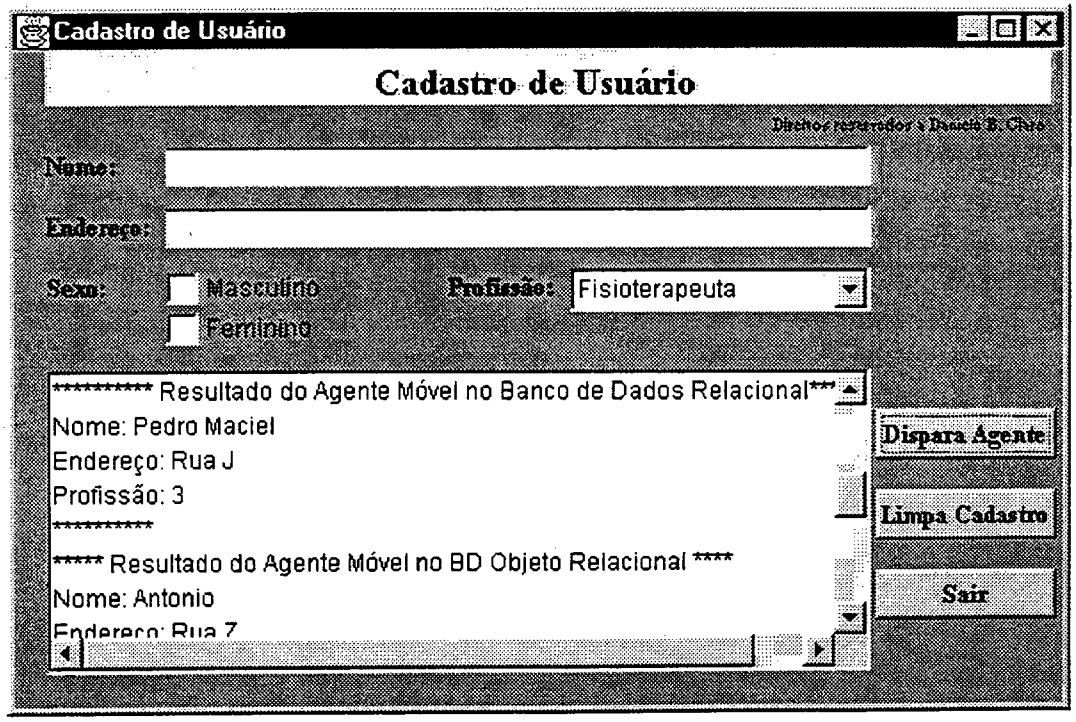

FIGURA 5.5 INTERFACE GRÁFICA CONTENDO OS RESULTADOS

5.5 – Banco de Dados

Os bancos de dados utilizados nesta implementação requerem uma estrutura individual na exposição dos seus dados. Diante disso, segue abaixo detalhes da administração e recuperação de dados referente aos bancos específicos. Ainda assim, foram inseridos dados e criados tipos distintos que permitem a manipulação dos mesmos de acordo com o tipo de banco.

5.5.1 - Banco de Dados Relacional

Este banco de dados é o mais comum atualmente no meio comercial, permite os relacionamentos entre as relações, também chamadas de tabelas e a inserção de dados através das cláusulas SQL-92, manipulando os dados através de conjuntos.

Neste banco de dados foram criadas três tabelas: eUsuario, eEmprestimo e eLivro, como demonstradas na figura 5.6.

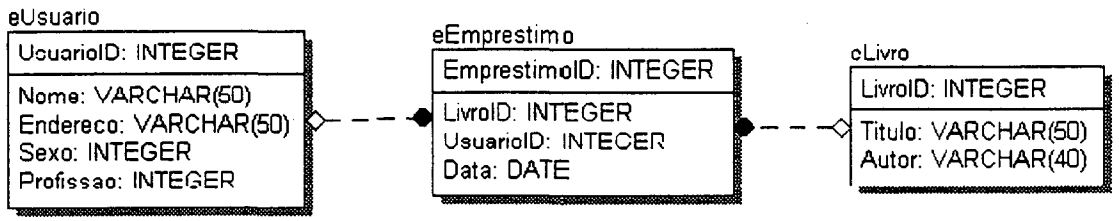

FIGURA 5.6 DIAGRAMA DE ENTIDADE-RELACIONAMENTO DO BD RELACIONAL

Após realizada a etapa da análise estruturada dos dados, foi realizada a conexão com o Banco de Dados relacional, o SQL Server 7.0 da Microsoft, sendo criadas as tabelas acima mencionadas. Para tal utilizou-se o seguinte *script* de criação:

CREATE TABLE [dbo].[eEmprestimo] (

```
  [EmprestimoID] [int] NOT NULL ,
  [LivroID] [int] NOT NULL ,
  [UsuarioID] [int] NOT NULL ,
  [Data] [timestamp] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[eLivro] (
  [LivroID] [int] NOT NULL ,
  [Titulo] [int] NULL ,
  [Autor] [int] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[eUsuario] (
  [UsuarioID] [int] NOT NULL ,
  [Nome] [varchar] (50) NULL ,
  [Endereco] [varchar] (50) NULL ,
  [Sexo] [int] NULL ,
  [Profissao] [int] NULL
) ON [PRIMARY]
GO
```

Após a criação das tabelas, há uma alteração das mesmas no sentido de definir as chaves primárias.

```
ALTER TABLE [dbo].[eEmprestimo] WITH NOCHECK ADD
  CONSTRAINT [PK_eEmprestimo] PRIMARY KEY  NONCLUSTERED
  ( [EmprestimoID] )  ON [PRIMARY]
GO

ALTER TABLE [dbo].[eLivro] WITH NOCHECK ADD
  CONSTRAINT [PK_eLivro] PRIMARY KEY  NONCLUSTERED
  ( [LivroID] )  ON [PRIMARY]
GO

ALTER TABLE [dbo].[eUsuario] WITH NOCHECK ADD
  CONSTRAINT [PK_eUsuario] PRIMARY KEY  NONCLUSTERED
```

```
( [UsuarioID] ) ON [PRIMARY]
GO
```

As chaves estrangeiras também foram definidas permitindo assim um relacionamento da tabela de ***eEmprestimo*** com ***eLivro*** e ***eUsuario***.

```
ALTER TABLE [dbo].[eEmprestimo] ADD
  CONSTRAINT [FK_eEmprestimo_eLivro] FOREIGN KEY
        ( [LivroID] )
        REFERENCES [dbo].[eLivro] ( [LivroID] ),
  CONSTRAINT [FK_eEmprestimo_eUsuario] FOREIGN KEY
        ( [LivroID] )
        REFERENCES [dbo].[eUsuario] ( [UsuarioID] )
GO
```

Após a criação da estrutura do banco de dados relacional, os dados foram inseridos através de cláusulas SQL, semelhante a estrutura abaixo:

```
INSERT INTO eUsuario
VALUES (1, 'PedroMaciel', 'Rua J', 1, 3)
```

Neste caso, foi inserido um registro na tabela de ***eUsuario*** assim como pode ser realizada a mesma operação para inserir dados nas demais relações.

5.5.2 - Banco de Dados Objeto - Relacional

O banco de dados objeto-relacional requer algumas modificações em relação a estrutura dos dados. Os dados continuam sendo armazenadas em tabelas ou relações, mas a forma como estas tabelas são desenvolvidas é baseada em objetos que são os tipos abstratos de dados. A utilização do SQL-3 como forma de consulta aos dados e inserção dos valores permite que se obtenha uma maior proximidade com o mundo relacional.

No Oracle8i, banco de dados objeto-relacional trabalhado nesta implementação, as tabelas são compostas por um conjunto de objetos. Assim, as tuplas ou registros são instâncias dos tipos abstratos, ou seja destes objetos.

```
CREATE OR REPLACE TYPE objUsuario AS OBJECT (
  UsuarioID  Integer,
  Nome       Varchar(35),
  Endereco   Varchar(50),
  Sexo       Integer,
  Profissao  Integer
);

CREATE OR REPLACE TYPE objEmprestimo AS OBJECT (
  EmprestimoID  Integer,
  LivroID       REF objLivro,
  UsuarioID     REF objUsuario,
  Data          Integer,
);

CREATE OR REPLACE TYPE objUsuario AS OBJECT (
  LivroID    Integer,
  Titulo     Varchar(35),
  Autor      Varchar(50) );
```

O objeto ***Emprestimo*** contém uma referência ao tipo de dados ***Usuario*** e ***Livro***. Neste caso, o Oracle trata esta referência pelo OID do objeto que foi determinado internamente.

A criação, propriamente dita, da tabela ocorre com a instanciação destes objetos que foram criados. Assim, a tabela no Objeto-relacional é um conjunto de tipos de dados ou de objetos na sua estrutura singular.

```
CREATE TABLE Usuario OF objUsuario
  ( UsuarioID PRIMARY KEY);
```

Neste caso, há a definição da chave primária do objeto como sendo o ***UsuarioID***. As chaves estrangeiras, por sua vez, são referenciadas no momento de criação do tipo do objeto, assim a chave estrangeira desta relação seria a ***UsuarioID*** e ***LivroID*** no objeto ***objEmprestimo***, que faz referência ao objetos ***Usuario*** e ***Livro*** respectivamente.

A seleção dos dados das tabelas que não contém referência ocorre de forma semelhante ao banco de dados relacional, porém os objetos que permitem a referência a outros objetos, devem ser tratados como na orientação a objeto, colocando o nome do mesmo seguido do seu atributo, como segue abaixo:

```
SELECT Emp.UsuarioID.Nome as Nome, Emp.LivroID.Titulo as Titulo
FROM Emprestimo Emp;
```

Neste caso, o retorno desta consulta será os nomes dos usuários e o título dos livros que foram realizados empréstimos. A inserção dos dados nas tabelas que não contém referência aos outros objetos ocorre de forma semelhante ao SQL-92. No caso da tabela de empréstimo, que contém uma referência, a inserção ocorre seguindo os critérios de utilização da linguagem SQL-3. Assim, quando houver uma referência a determinado tipo abstrato de dado, a inserção da tabela ocorre através de uma subconsulta como segue abaixo:

```
INSERT INTO Emprestimo (EmprestimoID, LivroID, UsuarioID, Data)
   SELECT 2, REF(refLivro), REF(refUsuario), '11/07/2000'
   FROM Livro refLivro, Usuario refUsuario
   WHERE LivroId=2 AND UsuarioID = 2;
```

O exemplo acima está inserindo um registro na tabela de Empréstimo, fazendo referência a tabela de Livro e Usuário.

5.5.3 - Banco de Dados Orientado a Objeto

O banco de dados orientado a objeto tem como principal objetivo armazenar os estados dos objetos, ou seja, persistir os objetos para que seus estados possam ser recuperados em outro momento. O POET na versão 6.1 permite a visualização gráfica

dos objetos assim como o controle de instâncias e administração do banco de dados. Para tal possui duas ferramentas: *POET Administrator* e o *POET Developer*. A administração permite gerenciar o controle dos usuários e as permissões aos objetos e classes, assim como a definição do *workspace* como sendo a área de trabalho onde os objetos residem.

Uma aplicação orientada a objeto utiliza várias classes em que não há a necessidade de armazenar os seus valores. Neste caso, estas classes são denominadas de transientes e assumem o valor padrão do comportamento. Porém, caso haja a necessidade de persistir uma classe, é necessário que, explicitamente, defina-a como uma classe persistente. Nestes casos, o POET fornece um arquivo de configuração das classes que permite, além de outras funções, definir a persistência das classes. Este arquivo é denominado ***ptj.opt*** e permite configurar as opções do POET.

```
[schemata\AgentesDict]
oneFile = true

[databases\AgentesDB]
schema = AgentesDict
oneFile = true
```

Nestes dois primeiros parâmetros estão sendo definidos o dicionário de dados e o banco de dados. O parâmetro *oneFile* se refere a quantidade de arquivos que serão gerados em se tratando de índices e estrutura dos dados. Neste caso, os índices e as estruturas ficarão em um único arquivo denominado ***AgentesDB***.

```
[classes\Usuario]
persistent = true
hasExtent = true
useIndexes = UsuarioNomeIndex


[indexes\UsuarioNomeIndex]
class = Usuario
members = nome_
unique = true
```

A definição acima da classe **Usuario** segue as mesmas nomenclaturas para as demais classes **Emprestimo** e **Livro**. Nas cláusulas referente a classe usuário, há os parâmetros referente aos índices e persistências. O parâmetro *hasExtent* é utilizado para permitir que através da linguagem OQL (*Object Query Language*) obtenha um conjunto de objetos baseado na consulta. Caso este parâmetro fosse definido como falso, a recuperação dos dados dos objetos só poderia ser realizada individualmente, através da cláusula *Lookup*. Os índices são definidos de acordo com um atributo presente na classe, se referenciando à denominação do mesmo no banco de dados, com o acréscimo de um "_" (sublinhado).

As classes, por sua vez, são definidas utilizando a própria linguagem Java, mapeando os métodos e os atributos para a persistência predefinida no arquivo ***ptj***.

```
class Usuario {
  private String nome_;
  private String endereco_;
  private int sexo_;
  private int profissao_;
  public Usuario( String nome, String endereco, int sexo, int profissao ) {
          nome_ = nome;
          endereco_ = endereco;
          sexo_ = sexo;
          profissao_ = profissao;
  }
  public Usuario( String nome, String endereco ) {
          nome_ = nome;
          endereco_ = endereco;
          sexo_ = 0;
          profissao_ = 0;
  }
     public Usuario() {
          nome_ = "Sem nome";
          endereco_ = "Sem endereco";
          sexo_ = 0;
```

```
            profissao_ = 0;
    }
    public String getNome()  {
            return nome_;
    }
       public String getEndereco() {
            return endereco_;
    }
       public int getProfissao() {
            return profissao_;
    }
       public int getSexo() {
            return sexo_;
    }
    public void setNome( String nome ) {
            nome_ = nome;
    }
}
```

Em relação à classe ***Emprestimo***, a mesma sofre algumas modificações por trazer referências às classes ***Livro*** e ***Usuario***.

```
class Emprestimo {
    private Usuario usuario_;
    private Livro livro_;
    private String data_;
    public Emprestimo( Usuario usuario, Livro livro,  String data) {
            usuario_ = usuario;
            livro_ = livro;
            data_ = data;
    }
    public String getData(Usuario usuario, Livro livro)  {
            return data_;
    }
    public void setData( String data )  {
            data_ = data;
    }
}
```

Para a completa manipulação do banco de dados, é necessário que se compile as classes geradas utilizando o comando do Java:

*javac *.java;*

Após obtida todas as classes que irão ser referenciadas pelo arquivo ***ptj***, há a necessidade de registrar estas classes no dicionário de dados do POET. Isso ocorre baseado no seguinte comando:

ptj -enhance -inplace -create

Este comando faz referência ao arquivo ***ptj*** e realiza as publicações das classes uma vez ditas persistentes. Há ainda a possibilidade de somente atualizar o banco, sendo a última cláusula do comando acima substituída por *-update*.

A inserção de dados utilizando o POET pode ser realizada de duas maneiras: através da interface gráfica do *POET Developer* ou através da aplicação utilizando o comando *bind:*

AgentesDB.bind('objUsuario');

Não há a possibilidade de inserir um conjunto de dados, como ocorre com o comando *INSERT* do SQL-2 e SQL-3. A linguagem OQL não permite a manipulação de dados no sentido de inserção e exclusão de objetos, neste caso, estas manipulações dos objetos só podem ocorrer individualmente.

5.6 – Os Agentes Estáticos

Os agentes estáticos são em número de três, de acordo com as bases de dados contidas neste ambiente. A instanciação dos mesmos ocorre no agente móvel, onde este é responsável por designar a máquina que o agente estático será desenvolvido. Para tal, faz-se necessário a utilização da linha abaixo mencionada:

AgenteEstaticoI agBDOO =
(AgenteEstaticoI)Factory.create("AgenteBDOO","tcp://192.168.0.4:9000");

Neste caso, há uma ordem para se criar o agente estático na máquina cujo endereço IP é *192.168.0.4* e se comunicar com o ambiente de execução através da porta 9000. Além disso, há a presença da interface dos agentes estáticos que permite facilitar a criação dos métodos e do próprio agente. A estrutura da interface *AgenteEstaticoI* segue abaixo como mostrado na figura 5.7.

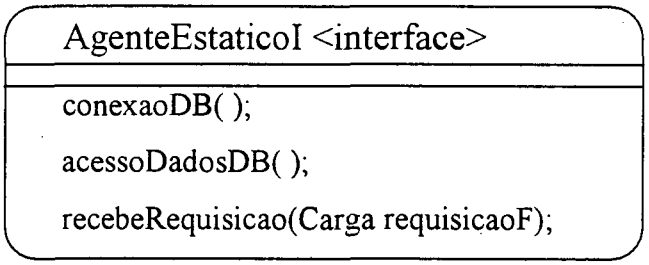

FIGURA 5.7. INTERFACE AGENTEESTATICOI

Os dados dos requisitos para a pesquisa do usuário no agente estático são obtidos através da invocação do método citado abaixo:

agBDOO.recebeRequisicao(requisicao);

O retorno gerado pelo AgenteBDOO, denominação concedida por se tratar do agente estático que realiza a pesquisa na base de dados orientada a objeto, é armazenado em um objeto denominado *Resultado* da classe ***Carga***. A classe Carga, por sua vez, é

organizada em um vetor e passada para o **AgenteMovel**, como sendo sua carga de retorno.

As conexões e os acessos às bases de dados diferem em relação ao banco de dados em questão. Assim, surge a necessidade de serem analisados isoladamente.

5.6.1 - AgenteBDR

O AgenteBDR, agente que manipula as informações relativas ao banco de dados relacional, é responsável pela efetiva conexão, utilizando assim o JDBC. Este , por sua vez, utiliza o ODBC para realizar a comunicação efetiva com o banco de dados.

A conexão com o banco de dados relacional utilizando o JDBC-ODBC é realizada através da detecção dos drivers. De posse do driver, a conexão é efetivada com a base de dados definida na folha de dados do ODBC.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con= DriverManager.getConnection("jdbc:odbc:AgentesDB", "", "");
```

O primeiro parâmetro do método *getConnection* se refere ao nome fornecido na folha de dados do ODBC para o acesso a base de dados e o segundo e terceiro parâmetro se referem ao *login* e senha do usuário, respectivamente.

Uma vez realizada a conexão com o banco de dados torna-se necessário a criação de três classes inerentes ao JDBC: *Connection*, *Statement* e *ResultSet*. Através destas classes é que ocorre a conexão do driver específico para o JDBC com a fonte de dados definida através do ODBC, que permitirá o acesso a base de dados relacional.

A classe *Statement* utiliza-se da classe *Connection* para realizar a execução das sentenças SQL de acesso aos dados presentes no banco. Uma vez executada uma sentença contendo uma consulta SQL, os resultados obtidos são armazenados em uma outra classe denominada de *ResultSet*. Esta classe permite que se navegue no conjunto de dados obtidos através da consulta à base de dados. Segue um exemplo abaixo:

Statement stmt = con.createStatement();

*ResultSet rs = stmt.executeQuery("SELECT * FROM eUsuario WHERE Profissao = 3");*

No exemplo acima o valor da profissão está sendo passado como um parâmetro constante para a consulta a base, sendo o valor 3 correspondente a fisioterapia.

Uma vez realizada a consulta à base de dados e os resultados armazenados no objeto Resultado da classe Carga, o agente estático obtém uma referência do agente móvel, sendo o mesmo instanciado localmente na máquina onde o agente estático se encontra. Isso ocorre através do comando abaixo:

ManipulaI agMovel=(ManipulaI) Namespace.lookup("//192.168.0.2:6000/NASDAQ");

Obtida a referência do agente móvel, um método de envio do resultado obtido nesta consulta é executado, sendo os dados repassados para o *NameSpaces*, como segue abaixo:

agMovel.enviaResultado(itsResult);

5.6.2 - AgenteBDOR

O AgenteBDOR, agente que manipula as informações relativas ao banco de dados objeto-relacional, segue as mesmas estruturas do *AgenteBDR*, divergindo apenas na conexão que o mesmo não utiliza o ODBC, realizando a conexão com acesso nativo ao banco de dados, no caso o Oracle8i.

Class.forName("oracle.jdbc.driver.OracleDriver");
DriverManager.getConnection("jdbc:oracle:thin:@agentemovel:1521:AMORIns","dan claro","nt");

Neste caso acima, o nome após o simbolo @ (arroba) é o nome da máquina onde está o banco de dados, o número 1521, é a porta default onde este banco de dados está escutando as requisições, e AMORIns, se refere a instância do agente móvel em se tratando do banco de dados objeto-relacional. Uma instância no Oracle pode ser entendida como o "um conjunto de processos e área de memória que o Oracle utiliza para gerenciar o acesso ao banco de dados"[BOBR00]. Além disso os demais parâmetros se referem ao *login* e senha do usuário, respectivamente.

O modo de acesso aos dados, em se tratando de uma tabela que não tenha referência a tipos abstratos de dados, ocorre similarmente ao banco de dados relacional. Além disso, também há a utilização das três classes de manipulação presente no relacional e o envio do resultado, sendo também obtida a referência do agente móvel para a máquina que o *AgenteBDOR* se encontra. Segue abaixo a figura 5.8 com os passos realizados pelo agente estático para obter os acessos aos dados no banco de dados objeto relacional.

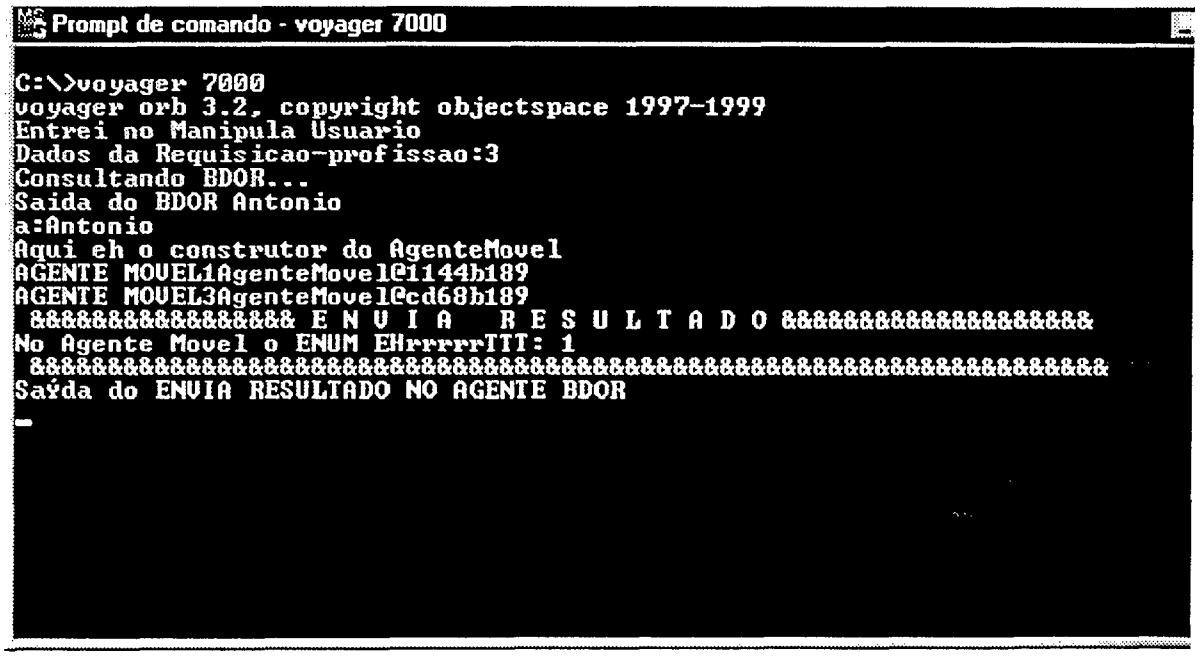

FIGURA 5.8 SAÍDA NA TELA DO DOS DOS DADOS PROCESSADOS PELO AGENTEBDOR

5.6.3 - AgenteBDOO

O **AgenteBDOO**, é responsável por manipular as informações relativas ao banco de dados orientado a objetos. Este agente, no tocante a conexão com o banco de dados, se difere do relacional quanto do objeto relacional. Não há utilização de drivers de conexão, mas é realizada uma instância da classe ***Database*** do POET e estabelecida uma conexão com esta base de dados, como segue abaixo:

```
Database agenteDB = new Database();
agenteDB.open("poet://DANCLARO:nt@LOCAL/AgentesDB",Database.OPEN_
READ_WRITE);
```

Os parâmetros passados para o método *open* do banco de dados se referem ao login e senha do usuário, tais como exemplificado com *DANCLARO:nt*. Logo após o @ (arroba) é o local onde o banco de dados está sendo executado, neste ambiente o mesmo está sendo executado localmente. O parâmetro seguinte se refere ao nome do banco de dados propriamente dito. E por fim, os acessos permitidos, neste caso de abertura da conexão, leitura e escrita.

Uma vez estabelecida a conexão ao banco de dados, pode-se realizar uma consulta aos objetos situados no mesmo. Esta consulta envolve a abertura de uma transação, garantindo assim a integridade das informações, sendo seguida por uma *string* que contém a sentença OQL. No caso do banco de dados orientado a objeto, o mesmo utiliza a OQL (Object Query Language) como forma de manipulação de conjunto de objetos.

```
String queryString ="select c from UsuarioExtent as c where c.profissao_ 3"
```

Em relação à sentença montada acima, o *UsuarioExtent* se refere a um conjunto de objetos da classe **Usuario**. No momento da criação da classe persistente **Usuario**, há a possibilidade de permitir se aquele objeto poderá ser manipulado com um conjunto de objetos ou somente isolado. No caso de se tratar da manipulação utilizando a OQL, define-se que aquela classe terá um *Extent*. Além disso, as manipulações dos objetos

através de conjuntos só podem ocorrer por intermédio de um *alias* para a classe *Extent*. No exemplo elucidado, o *alias* é observado através do nome "*c*". Outra consideração a ser observada é que as manipulações dos atributos no banco de dados é normalmente referenciada por um "_" após o nome do atributo definido pela classe.

De posse da sentença de consulta a base de dados, deve-se executá-la e obter o resultado do banco. Para tal, há a necessidade de manipular uma classe fornecida pelo POET denominada de *OQLQuery*.

```
OQLQuery query = new OQLQuery( queryString );
Object result = query.execute();
```

O resultado da consulta é armazenado em um objeto comum, sendo realizado posteriormente um *cast* para converter o resultado para o tipo de objeto requerido, no caso um usuário. Os dados relativos à consulta são obtidos através da invocação dos métodos presente na classe ***Usuario***. Um exemplo destes métodos é a recuperação do nome do mesmo, quando a consulta trouxer os dados desejados, como segue o exemplo abaixo:

```
Usuario umUsuario = Usuario( );
umUsuario.getNome();
```

Assim como os demais agentes estáticos, o **AgenteBDOO** envia o resultado obtido da consulta ao ***AgenteMovel*** através da referência obtida do mesmo, sendo logo em seguida executado o método de *enviaResultado()*, como explicado anteriormente. Segue abaixo a figura 5.9 que retrata a consulta do **AgenteBDOO** no banco de dados Orientado a Objetos.

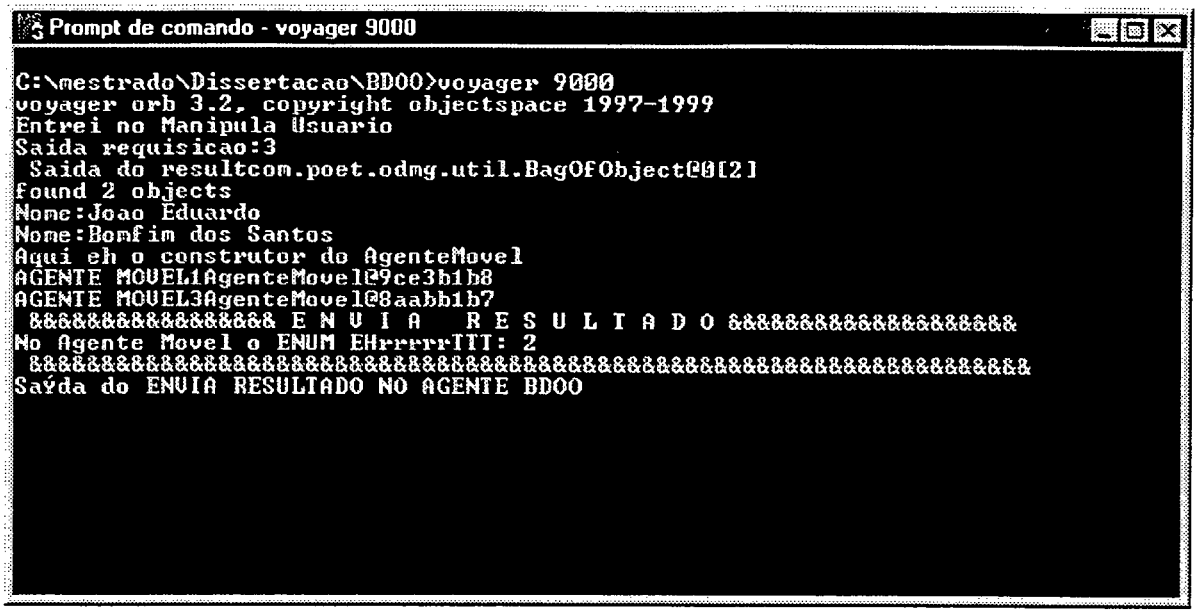

FIGURA 5.9 SAÍDA NA TELA DO DOS DOS DADOS PROCESSADOS PELO AGENTEBDOO.

6. Conclusão

A integração de bases de dados distintas utilizando a tecnologia de agentes que trafegam pela rede em busca das informações facilita a manutenção e a obtenção dos dados requeridos pelo usuário. A transparência das ações do agente na rede, assim como a comunicação com outros agentes permitindo o acesso às diversas bases de dados, torna este ambiente propício para a interação com um usuário.

Ainda assim, a integração destas bases de dados permite uma redução nos custos de uma organização, onde os sistemas legados e as novas tecnologias de armazenamentos de dados podem se inter-relacionar, fornecendo aos usuários informações mais consistentes e provindas de uma gama maior de base de dados.

Além de trazer estes benefícios, a utilização da mobilidade de código deve permitir uma maior consistência nos dados, visto que os agentes se deslocam para as máquinas e executam suas tarefas localmente. Assim, os dados não ficam trafegando pela rede, mas ficam armazenados no compartimento dos agentes móveis, fornecendo mais um nível de segurança.

Em relação a arquitetura Cliente/Servidor, este ambiente permite que os objetos distribuídos pela rede em diversas máquinas contendo os bancos de dados sejam cliente e ao mesmo tempo servidores de serviços, balanceando assim, a carga dos servidores de dados. Além disso, uma vez que uma máquina que contenha um banco de dados não esteja ativa ou o acesso aos dados tenha sido danificado, o agente móvel continua a

pesquisa nas outras máquinas disponíveis, trazendo os resultados, embora parciais. Em uma arquitetura Cliente/Servidor, caso o servidor não esteja disponível no momento, a obtenção dos resultados não ocorre, prejudicando assim o trabalho do usuário.

A consulta às bases de dados relacional, orientada a objetos e objeto-relacional ocorreu de forma satisfatória, sendo o resultado obtido com integridade e rapidez. Porém, a criação remota dos agentes estáticos ainda deve sofrer melhorias tanto por parte da plataforma utilizada quanto por parte dos mecanismos de criação.

Assim, em uma extensão deste trabalho, propõe-se a avaliação dos mecanismos utilizados para a criação remota dos agentes assim como modificações e testes com outras plataformas que permitem a utilização dos agentes móveis. Dentre as plataformas pesquisadas, deve-se avaliar o comportamento deste ambiente com o Aglets da IBM. Além disso, aspectos específicos de segurança dos agentes móveis podem beneficiar, assim como as questões relativas às transações necessárias no processo do agente estático também podem ser observadas.

Outrossim, seria a continuação das manipulações dos dados nas diversas bases de dados, o que permitiria o agente móvel distinguir entre consultas às bases de dados, inclusão e exclusão das informações. Além disso, a formalização deste trabalho facilitando a análise do desempenho, assim como o controle das transações em relação a plataforma utilizada permitiria um gerenciamento mais efetivo dos agentes em questão.

7. Referências Bibliográficas

[BELL92] BELL, David; GRIMSON, Jane; **Distributed Database System**, Addison-Wesley, 1992.

[BIGU98] BIGUS, Jennifer; BIGUS, Joseph; **Constructing Intelligent Agents with Java**;Willey Computer Publishing,1998

[BOBR00] BOBROWSKI, Steve; **Oracle 8i for Windows NT Starter kit;**Oracle Press, 2000.

[BREN98] BRENNER, Walter; ZARNEKOW, Rudiger; WITTIG, Hartmut; **Intelligent Software Agents.** Springer-Verlag Press, 1998.

[BUSH99] BUSHIA, Gisele; Ferreira, João Eduardo. **Compartilhamento de Módulos de Bases de Dados Heterogêneas através de Objetos Integradores**; Publicado no SBBD, 1999.

[CARD99] CARDOZO, Eleri; KARMOUTH, A . **Mobile Agent-based System: an Alternative Paradigm for Distributed Systems Development**,1999.

[CAST92] CASTELLANOS, M.;**A Cononical Model for the Interoperability amog Object and Relational Databases;** International Workshop on Distributed Object Management (IWDOM),1992.

[CHES95] CHESS, D.M.; HARRISON, C. G.;Lebine, D;.**Itininerant Agents for Mobile Computing** IBM Reserach Report RC 20010, IBM Research Division, 1995.

[CHOI99] CHOI, InJun **Object database & ODMG-93** http://batman.postech.ac.kr/ seminar/odmg/index.html, 1999.

[CLAR00a] CLARO, Daniela B.;SOBRAL,João B.M.; **Integration of Databases using Mobile Code**, Proceeding of Workshop 2000, p.227-232, Agent-Based Simulation, Passau, Germany.

[CLAR00b] CLARO, Daniela B.;SOBRAL,João B.M.; **Agentes Móveis aplicados à Integração de Banco de Dados**, SEMISH 2000,p.21, Anais do XX Congresso Nacional da Sociedade Brasileira de Computação,Curitiba -PR.

[COND99] CONDE, Javier **Mobile Agents in Java** http://wwwinfo.cern.ch/asd/rd45/ white-papers/9812/agents2.html,1999.

[CLUR97] CLURE, Steve Mc. **Object Database vs. Object-Relational Database**. Agosto, 1997. http://www.cai.com/products/jasmine/analyst/idc/14821E.htm

[DATE91] DATE, C.J. **Introdução à Sistemas de Banco de Dados**, Rio de Janeiro, editora Campus, 1991.

[DEBO99] DEBONI, José E.Z. **Traduzindo Objetos em Bancos de Dados Relacionais**, Eng.da Computação – USP,1999.

[DELG99] DELGADO, Armando L.N. **Agentes Móveis em Java**, 1999.

[FERR93] FERREIRA, Aurelio B.H. **Minidicionário da língua portuguesa**, Nova Fronteira, 3ª Edição, 9ª Impressão, 1993.

[FORD99] FORD, B; Karmouth, A . **An Architectural model for mobile agent-based multimedia apllications,**1999.

[FORT99] FORTIER, Paul J. **SQL-3 Implementing the Object-Relational Database**, McGraw-Hill, 1999.

[FRAN96] FRANKLIN, Stan; GRAESSER; **Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents** Institute for Intelligent Systems, University of Memphis, 1996.

[HORI98] HORIZON SYSTEMS LABORATORY **Mobile agent computing: A white paper**. Technical report, Mitsubishi Electric ITA (http://www.meitca.com/HSL/Projects/ Concordia), January 1998.

[KHOS94] KHOSHAFIAN, Setrag **Banco de Dados Orientado a Objeto**;IBPI Press,1994

[KING97] KING, Nelson. **The Object Database Goes Online** , Janeiro, Internet Systems, 1997 http://www.dbmsmag.com/9701i07.html

[KOTA94] KOTAY, K; KOTZ, D; **Transportable Agents** Departament of Computer Science – Darmouth – 1994.

[KWAN99]KWAN, W; KARMOUTH, A . **Multimedia Agents in a Distributed Broadband Environment**, 1999.

[LITW90] LITWIN, W; **Interoperability of Multiple Autonomous Databases,** ACM Computing Surveys, 22(3):267-298,1990.

[MELO98] MELO, Rubens N.; Silva, Sidney D.; Tanaka, Asterio K. **Banco de Dados em Aplicações Cliente-Servidor** Infobook S.A , 1998.

[MELL97] MELLO, Ronaldo dos Santos; **Avaliação de Sistemas de Gerência de Banco de Dados Orientado a Objetos** UFSC, 1997.

[MILO99] MILOJICIC, Dejan; DOUGLIS, Frederic; **Mobility: Process, Computer and Agents** ACM Press, 1999.

[NAVA94] NAVATHE, Shamkant B; ELMASRI, Ramez. **Fundamentals of Database Systems**, Addison-Wesley, 2ª. Edição, 1994.

[ORFA97] ORFALI, Robert; HARKEY, Dan; EDWARDS, Jeri. **Instant CORBA**, Wiley Computer Publishing, 1997.

[OSHI98] OSHIMA, Mitsuru; LANGE, Danny B. **Programing and Deploying Java Mobile Agents with Aglets**, Addison-Wesley Publishing Company, 1998.

[OZSU91] OZSU, M. Tamer; VALDURIEZ, Patrick. **Principles of Distributed Database Systems**. Prentice Hall, 1991.

[PAPA99] PAPAIONNOU, Todd; EDWARDS, John. **Mobile Agent Tecnology Enabling The Virtual Enterprise: A Pattern for Database Query** Loughborough University, UK,1999.

[PONS99] PONS, Anne; KERHERVÉ **A Maintenance Process for Object Database Application.** http://www.cs.umd.edu/~sharip/wess/papers/pons.html, 1999.

[REES97] REESE, George. **Database Programming with JDBC and Java**, O'Reilly, 1997.

[RICC99]RICCIONI, Paulo Roberto **Agentes Móveis**,1999.

[RIOR99] RIORDAN, Rebecca M. **Desingning Relational Database Systems;** Microsoft Press, 1999.

[RUSS95] RUSSEL, S; NORVIG,P. **Artificial Intelligence: A Modern Approach**; New Jersey, Pretice Hall, 1995.

[SAHU97] SAHUGUET, Arnaud; **Agents and Databases;** CIS-650, May,1997.

[SOLE99] SOLER, Ortin; PRIETO, F.Martinez; GUTIÉRREZ, A.B.A. **A Reflective Persistence Middleware over na Object Oriented Database Engine**, Publicado no SBBD, 1999.

[VASC98] VASCONCELOS, Lara L.; COSTA, Flavia V. **Técnicas de Gerência de Banco de Dados**, Unifacs, 1998.

[VOYA97] VOYAGER CORE TECHNOLOGY GROUP - ObjectSpace, Inc. **Agent-enhanced distributed computing for java**. Technical report, Março, 1997 http://www.objectspace.com/ developers/voyager/white/index.html

[ZIMM96] ZIMMERMANN, Chris. **Advances in Object-Oriented Metalevel Architectures and Reflection**, CRC Press, 1996.