Faculdade de Engenharia da Universidade do Porto

Mestrado em Tecnologia Multimédia

ADO, um novo método de acesso a fontes de informação

Jorge Alberto Canhoto Coutinho da Rocha

Licenciado em Informática Industrial
pelo Instituto Superior de Engenharia do Porto

Dissertação submetida para satisfação parcial dos
requisitos do grau de mestre em
Tecnologia Multimédia

Dissertação realizada sob a supervisão do

Professor Eurico Carrapatoso

do Departamento de Engenharia Electrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

Porto, 30 de Novembro de 1999

RESUMO

Hoje a multimédia integra um conjunto bastante vasto de tecnologias para alcançar os seus fins e recorre-se cada vez mais às aplicações multimédia nos mais variados cenários. Esta nova realidade para as aplicações significa que estas têm que conseguir manipular a informação que tradicionalmente tratavam e manipular um novo conjunto de fontes de informação para as quais não estavam preparadas. Por vezes, a informação que é necessária encontra-se espalhada por várias plataformas e, em muitos casos, o acesso é crítico devido ao facto das fontes de informação serem normalmente orientadas ao tipo de informação armazenado.

O trabalho descrito nesta tese teve como objectivo explorar uma tecnologia de acesso a fontes de informação para desenvolver um sistema de *Trouble Tickets* para o Centro de Informática Prof. Correia de Araújo, tirando partido das capacidades do método de acesso.

Nesta dissertação começa-se por analisar várias tecnologias de acesso a fontes de informação. Entre elas, o *ActiveX Data Objects* (ADO), devido ao seu modelo de objectos orientados para o acesso a informação, revelou-se numa primeira impressão bastante flexível e independente da fonte de informação.

Apresenta-se seguidamente o modelo de objectos do ADO, descrevem-se as características de cada objecto bem como as extensões do ADO para a manipulação da estrutura da fonte de informação e a possibilidade de acesso multidimensional.

Descreve-se depois os aspectos de implementação do sistema de *Trouble Tickets*, nomeadamente como é feito o acesso à fonte de informação, refere-se como é que o ADO pode ser integrado num sistema composto por vários níveis e são apresentados exemplos de aplicações que usam componentes multimédia e fazem acessos a fontes de informação.

Descreve-se também como é feita a troca de informação entre o interface ADO de alto nível e o nível da camada inferior, ou seja, qual o percurso desde o pedido inicial da conexão até à obtenção dos dados.

ABSTRACT

Today multimedia combines a rather set of technologies in order to achieve its goals as well as multimedia applications are increasingly used on many different areas. This new reality forces applications into handling both the usual information and a new set of information sources to which they weren't prepared. Sometimes, the necessary information is spread across several platforms. In most cases, what makes this access difficult is the fact that the source of this information is orientated to the type of information stored.

This work aims to exploit a technology that provides the access to a set of information and in order to develop an application of Trouble Tickets for the 'Centro de Informática Prof. Correia de Araújo' that uses its capabilities.

We started analysing several access technologies to information sources. Among these, the ActiveX Data Objects (ADO), due to its object oriented model to information access has revealed itself as a flexible and data independent platform.

Then, ADO object model is introduced, each object characteristics are described as well as ADO extensions to manipulate the structure of information source and the capability of multidimensional access.

Trouble Tickets implementation aspects are also given an outline, namely how the access to information source is achieved and how ADO can be integrated in a tiers system. Examples of applications that use multimedia and use data sources are included in this work.

As ADO is an application level interface, it's also mentioned how the connection between that level and the system level is done, from the initial request for a connection to the retrieval of data.

RESUME

Aujourd'hui la multimédia intègre un ensemble tout à fait vaste de technologies pour atteindre ses fins et il recours, de plus em plus, à des aplications multimédia les plus variés. Cette nouvelle réalité pour les aplications, signifies quelle doivent manipuler l'information que traditionnellement elles traiter et manipuler un nouveau ensemble de sources d'information pour laquelle na pas été préparé. Porfois, l'information qui est nécessaire se trouve dispersé pour plusieurs plate-formes et, dans beaucoup de cas, l'accès est restrie dû au fait que les sources d'information, habituellement som orientés au type d'informations entreposés.

Le travail décrit dans cette thèse a comme objectif explorer une technologie d'accès aux sources d'information pour développer un système de Troubles Tickets pour le Centre d'Informática Prof. Correia de Araújo, prenant partie des capacités de la méthode de l'accès.

Dans cette dissertation commence pour analyser plusieurs technologies accès aux sources d'information. Parmi eux, ActiveX Dates Objets (ADO), dû à son modèle de l'objets guidé pour l'accès de l'information, il a révélé dans une première impression tout à fait flexible et indépendante de la source d'information.

Aprés, on presente le modèle d'objectos de ADO, les caractéristiques de chaque objet sont décrites aussi bien que les extensions de ADO pour la manipulation de la structure de la source d'information et la possibilité de multidimensionnel de l'accès.

Aprés-sa , on décrit les aspects d'implementação du système de Trouble Tickets, comme il est fait l'accès à la source d'information, se reporte comme l´ ADO peut être intégré dans un système composé par les plusieurs niveaux et sont présentés exemples de aplications qui utilisent des composant de la multimédia et font des accès à sources d'information.

Il est aussi décrit comment est fait le changement d'information entre l'interface ADO de haut niveau et le niveau de la couche inférieure, c'est-à-dire, quel est le parcours du début de la demande de liaison à l'obtention des données.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ACRÓNIMOS

ADTG - Advanced Data TableGram

ADO – ActiveX Data Objects

ADO MD - ADO Multidimensional

ADOX - Data Objects Extensions for Data Definition Language and Security

API - Application Programming Interface

ASP - Active Server Page

CICS - Virtual Sequencial Access Method

CLI - X/Open SQL Access Group's Call Level Interface

COM - Component Object Model

DAO - Data Access Object

DCOM - Distributed COM

DDL - Dynamic Data Languages

DML - Data Manipulation Language

HTML – Hypertext Mark-up Language

IIS - Internet Information Server

IMS - Information Management System

ISAM - Indexed Sequential Access Method

JDBC - Java Database Connection

MDP - Multidimensional Data Provider

ODBC - Open Database Connectivity

OLAP - Online Analytical Processing

RAD - Rapid Application Development

RDO - Remote Data Object

RDS - Remote Data Service

TDP - Tabular Data Provider

VSAM - Virtual Sequencial Access Method

WFC - Windows Foundation Class

XML - Extensible Markup Language

I. INTRODUÇÃO

1.1. Motivação

Hoje a multimédia integra sem dúvida um conjunto bastante vasto de tecnologias para alcançar os seus fins e recorre-se cada vez mais às aplicações multimédia nos mais variados cenários. A qualidade visual, auditiva e interactiva, das aplicações, é um factor que pode ser determinante no sucesso de uma aplicação. Este nova realidade para as aplicações faz com que estas tenham que conseguir manipular a informação que já tratavam e manipular um novo conjunto de fontes informação para o qual não estavam preparadas.

Por vezes, a informação que é necessária encontra-se espalhada por várias plataformas e o acesso à informação numa grande parte dos casos é crítico devido ao facto das fontes de informação serem normalmente orientadas ao tipo de informação armazenado. Para uma determinada aplicação multimédia poder aceder a várias fontes de informação, deve usar métodos de acesso o mais genéricos possível e que suportem as especificações das fontes de informação.

Os métodos de acesso a bases de dados constituem um assunto quase inesgotável e de relevante interesse actual principalmente para as aplicações que necessitam de ter acesso a vários tipos de dados.

Explorar novas funcionalidades e inovações, em áreas em que já se possui um conhecimento prévio, traduz-se, efectivamente, num enorme interesse e desafio, principalmente se atentarmos no facto de que o contacto diário com a tecnologia se perde e, por outro lado, a evolução tecnológica apresenta ritmos vertiginosos. Perante estes factos, surgiu a ideia de fazer uma tese centrada numa tecnologia de acesso a dados que, de certa forma, veio substituir outras por muitas e diversas razões.

O trabalho descrito nesta tese teve como objectivo explorar a tecnologia de acesso a bases de dados mais adequada para desenvolver um sistema de *Trouble Tickets* para o Centro de Informática Prof. Correia de Araújo.

Das várias tecnologias existentes, o *ActiveX Data Objects* (ADO) aparentava ser, numa primeira impressão, bastante flexível e independente das fontes de informação. O trabalho baseou-se na exploração desta tecnologia.

O ADO é uma *application programming interface* (API) que fornece um acesso a dados de uma forma consistente, com elevado desempenho e que responde a uma variedade de necessidades das equipas de desenvolvimento, nomeadamente a criação de aplicações clientes para acesso a fontes de informação, objectos de processamento ou objectos lógicos que são usados pelas aplicações, ferramentas, linguagens de programação ou *browsers* de acesso à *Internet*.

A principal razão da criação do ADO foi a reunião de um conjunto de funcionalidades habitualmente espalhadas por uma gama de tecnologias numa única, aproveitando as melhores capacidades de cada uma dessas tecnologias.

Depois da Microsoft ter disponibilizado a versão 1.0 do ADO, esta revelou-se pouco funcional para as intenções da empresa por ser pouco versátil e por estar longe dos objectivos inicialmente propostos para esta tecnologia. A versão 2.0 tinha que ser uma solução totalmente remodelada, cheia de novas potencialidades e com capacidade de integração de muitas fontes de informação, para poder ter sucesso entre as equipas de desenvolvimento.

Uma característica das fontes de informação é que podem ser perfeitamente distintas, tais como ficheiros pessoais completamente isolados, folhas de cálculo ou ficheiros de texto, para além das tradicionais bases de dados relacionais ou não relacionais. Todas estas são suportadas pelo ADO. Para além do acesso às fontes de informação, o ADO comporta duas novas facilidades: o acesso à informação multidimensional e a manipulação da estrutura da fonte. Através do ADO Multidimensional (ADO MD) é possível obter informação para navegar em estruturas multidimensionais, fazer perguntas e obter resultados. A manipulação de estruturas permite que uma aplicação

consiga gerir a estrutura ou o esquema da fonte de informação. Esta característica pode tornar-se particularmente útil quando se pretende criar novas fontes de informação a partir de fontes já existentes.

Outro factor importante para a escolha do ADO foi o facto deste pertencer ao *Universal Data Access,* que é a designação dada pela Microsoft a várias tecnologias que desenvolveu e que se relacionam entre si, nomeadamente ADO, OLE DB, e *Open Database Connectivity* (ODBC), e cuja característica é fornecer acesso a informação de um simples ficheiro até grandes bases de dados. Qualquer tecnologia pertencente ao grupo pode ser motivo para uma análise, tal como o OLE DB, mas na realidade a tecnologia para a grande maioria das equipas de programação actualmente é sem dúvida o ADO.

Em relação à plataforma escolhida, recorreu-se aos sistemas operativos da Microsoft, embora já várias empresas tenham feito o porte desta tecnologia para outros sistemas operativos, nomeadamente sistemas *Unix*, *Linux*, etc. Apesar de haver muitos argumentos contrários à sua utilização, os sistemas Windows são usados por milhões de utilizadores, principalmente utilizadores finais. Ora são exactamente estes os maiores consumidores de informação em aplicações cliente/servidor, onde o *software* cliente reside na máquina final e onde as fontes de dados podem ser diferentes e é para estes que o ADO se apresenta como uma boa solução para a manipulação da informação.

1.2. Objectivos

Pretendeu-se que o trabalho que é descrição nesta dissertação, se enquadrasse no âmbito do conhecimento da tecnologia ADO e na experimentação da ferramenta para simplificar o trabalho daqueles que vivem no dia a dia com os problemas de ter de implementar funcionalidades para o acesso à informação.

Para melhor conseguir esse objectivo, era preciso compreender o estado dessa e as suas tecnologia e as suas origens, a metodologia e era necessário fazer a sua

comparação com outros métodos tais como *Remote Data Objects* (RDO), *Data Access Objects* (DAO), *ODBCDirect.*

O passo seguinte seria analisar detalhadamente os objectos que a tecnologia possui para a comunicação com a fonte de dados bem como o modo como é feito o acesso a dados, o acesso à estrutura da fonte de dados e o acesso a dados já tratados para consultas.

O último objectivo era desenvolver exemplos que mostrassem como a tecnologia pode ser aplicada nomeadamente como pode ser feita a comunicação entre os objectos do ADO e como é que estes comunicam com as fontes de dados. Pretendia-se ainda verificar a capacidade de integração que o ADO e as linguagens de programação (como o *Visual Basic* e páginas Web) e como se podem construir automatismos para facilitar o trabalho aos programadores.

1.3. Estrutura da Dissertação

A presente dissertação encontra-se organizada em quatro capítulos, para além desta Introdução, na qual são dados a conhecer os objectivos da dissertação, a razão de escolha do método, o enquadramento e a importância que este têm para a multimédia.

No capítulo II, é feita uma apresentação de vários métodos desenvolvidos pela Microsoft para acesso a fontes de informação, descrição dos modelos, dos objectos de cada modelo e de como é possível aceder à informação usando os modelos e quando é que se usam as metodologias. É feita uma comparação entre estes métodos, indicando os melhores e piores pontos de cada um dos modelos apresentados. Também é descrito o que é o *Universal Data Access* no qual o ADO se enquadra. Outra tecnologia abordada e comparada é o JDBC que tem objectivos semelhantes ao ADO.

No capítulo III, é explicado mais pormenorizadamente o ADO. Na primeira parte apresenta-se como se pode manipular dados, na segunda como manipular a estrutura

da fonte de informação e na terceira como consultar dados via *Online Analytical Processing* (OLAP).

No capítulo IV, são apresentados dois exemplos distintos e que usam o ADO para aceder a uma fonte de informação. O primeiro exemplo pretende demonstrar as principais características do método bem como a sua aplicabilidade de duas formas diferentes. A primeira mostra através de uma aplicação muito simples como se pode utilizar os objectos do método para aceder aos dados através de uma perspectiva cliente/servidor e mostra também algumas características inovadoras que o método possui. A segunda mostra como é possível utilizar o ADO para criar páginas Web dinâmicas que necessitam de dados provenientes de fontes de informação. É demonstrado também como o ADO pode ser usado para aceder a um *stream* de som e como este pode ser disponibilizado para a aplicação.

O segundo exemplo mostra como é possível utilizar o ADO numa perspectiva diferente do primeiro exemplo. Como o ADO é uma forma de acesso a dados, é possível diferenciar numa aplicação os níveis de apresentação, lógico e de dados. O ADO pode ser usado como o mecanismo de acesso onde se situa fundamentalmente no nível de dados.

No final do capítulo IV é referida a relação que existe entre o ADO e a multimédia e são mencionados exemplos onde esta relação é importante. É também demonstrado como se pode extrair de uma base de dados relacional uma imagem e apresenta-la na Web.

No último capítulo, Conclusões, são feitas algumas considerações sobre o método de acesso utilizado, algumas referências sobre métodos concorrentes e tira-se algumas conclusões sobre a metodologia a partir dos exemplos desenvolvidos. Perspectiva-se a evolução do método no imediato e a contribuição que os fornecedores de fontes de informação podem ter para melhorar este ou outro mecanismo com características semelhantes e tecem-se algumas considerações sobre futuros desenvolvimentos.

II. TECNOLOGIAS DE ACESSO A FONTES DE INFORMAÇÃO

2.1. Introdução

A grande maioria das aplicações actuais necessita de acessos a conjuntos de dados. Normalmente, implementar o acesso a esses dados para aplicações onde os dados estão residentes na própria máquina não exige um esforço muito elevado já que as ferramentas oferecem normalmente um conjunto de possibilidades de acesso que praticamente não exige qualquer tipo de código para conseguir o acesso aos dados. Para aplicações de grande dimensão, o acesso a dados é bastante mais complexo, normalmente envolvendo acesso a dados remotos e a fontes de dados de tipos completamente distintos.

Neste caso, as equipas de desenvolvimento deparam-se com o dilema de ter de escolher a melhor tecnologia para aceder aos dados. Para responder a este dilema as equipas de desenvolvimento devem considerar dois pontos essenciais: a importância da reutilização do código e a capacidade de desenvolvimento do interface escolhido. Muitas vezes, os programadores implementam soluções exóticas para acessos a dados simplesmente por questões de desempenho ou para ter melhor controlo sobre os dados. Só que esta solução fica bastante dispendiosa para a manutenção desses dados. As novas tecnologias de acesso a dados reduzem normalmente o tempo de desenvolvimento, simplificam o código e conseguem ter um elevado desempenho, sem prejuízo das funcionalidades inerentes às tecnologias escolhidas.

Na maioria dos casos em que existe a necessidade de acessos a dados. É possível usar vários métodos de acesso no entanto cada tecnologia de acesso tem os seus pontos fortes e para poder escolher qual a melhor tecnologia de acesso, é necessário conhecer as especificações das diferentes abordagens.

Este capítulo vai estar organizado na seguinte perspectiva:

1. Métodos de acesso usando tecnologia Microsoft, onde são comparados os principais métodos de acesso a dados, com descrição dos seus modelos, qual a sua abordagem em relação à manipulação dos dados e em que situações é que devem ser usados;
2. Como e porque é que o ADO pertence ao *Universal Data Access* e a razão da criação deste conceito;
3. Uma comparação entre duas tecnologias rivais em alguns domínios: JDBC versus ADO.

2.2. Métodos de acesso usando tecnologia Microsoft

Os diferentes métodos de acesso vão ser descritos para melhor se perceber qual a razão da sua existência. Embora alguns deles usem modelos muito parecidos, o único que consegue abranger quase todas as funcionalidades é o ADO.

As diferentes tecnologias vão ser abordadas pela seguinte ordem:

1. Acesso a informação usando *Open Database Connectivity* (ODBC)
2. Acesso a informação usando *Data Access Objects* (DAO)
3. Acesso a informação usando *Remote Data Objects* (RDO)
4. Acesso a informação usando ODBCDirect
5. Acesso a informação usando *ActiveX Data Objects* (ADO)

2.2.1. *Open Database Connectivity* (ODBC)

O ODBC fornece um interface de programação (API) que permite o acesso de aplicações a um conjunto numeroso de bases de dados. Baseado na especificação *X/Open SQL Access Group's Call Level Interface* (CLI), o ODBC é um método para uniformizar o acesso a fontes de informação de diferentes formatos e a motores de bases de dados.

O ODBC é o interface mais usado para dados relacionais e, além disso, é relativamente rápido, mas é penalizado por este rápido acesso com um código complexo.

As características gerais do ODBC são:

- Elevado desempenho;
- Código complexo e difícil;
- Razoáveis requisitos de memória;
- Compatibilidade com as tecnologias de bases de dados existentes;
- Portabilidade para muitas plataformas e sistemas operativos;
- Um modelo de conexão que permite diferentes redes, sistemas de segurança e opções de bases de dados.

Com um *driver* normalizado de bases de dados relacionais, uma aplicação pode aceder a muita informação usando ODBC. Mas o ODBC não requer que a informação se pareça com uma base de dados relacional, logo não será a melhor forma de extrair informação. Por outro lado, se a base de dados não for relacional é bastante difícil escrever o *driver* ODBC porque basicamente é necessário codificar um motor relacional como parte da estrutura do *driver*.

2.2.1.1. Modelo de Objectos

A arquitectura consiste em quatro componentes, que são:

- API - chama as funções do ODBC para efectuar a conexão com a fonte de informação, receber e enviar dados e fazer a desconexão;
- *Driver Manager* - fornece informação à aplicação (como por exemplo a lista de fontes de informação), carrega os *drivers* dinamicamente à medida que for necessitando e informa do estado das transacções;
- *Driver* - processa as chamadas às funções do ODBC e controla todas as trocas entre a aplicação e a base de dados específica. Se necessário, o *driver* pode ainda fazer a tradução entre o SQL nativo e o SQL suportado pela fonte de informação;
- *Data Source* - interpreta a informação e associa-a ao motor da base de dados.

A aplicação usa a API do ODBC para se conectar à fonte de informação, enviar expressões SQL, receber informação e desconectar-se. O *driver manager* situa-se entre a aplicação e o *driver* ODBC, decide qual o *driver* a carregar e controla as comunicações entre o *driver* e as funções que a aplicação invoca. Finalmente, os *drivers* implementam as funções da API do ODBC para a base de dados em questão. A figura seguinte mostra como é que estas funções interagem.

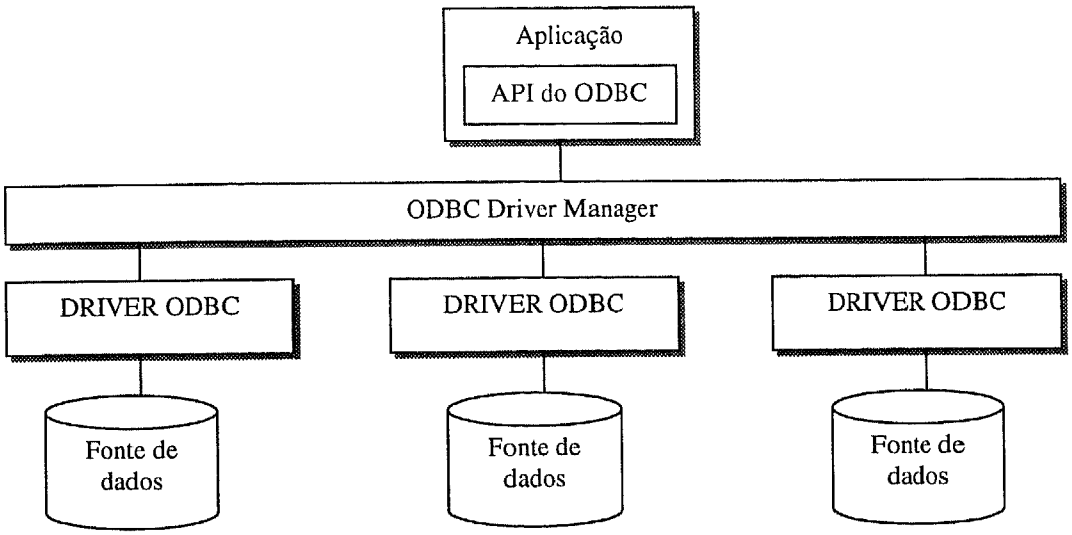

Figura II-1 - Modelo de Objectos do ODBC

A arquitectura do ODBC permite que uma aplicação possa aceder a diferentes fontes de informação, em locais diferentes, usando as mesmas funções disponíveis na API do ODBC. Uma vez construído o código de acesso a uma fonte de dados relacional, o código é facilmente usado com outra fonte de informação relacional.

2.2.1.2. Acesso a dados

O ODBC disponibiliza uma única plataforma para o acesso a todas as fonte de dados relacionais. Porque o ODBC oferece um bom suporte às aplicações e para os produtores das bases de dados relacionais, o resultado é uma única API que proporciona todas as funcionalidades de que os programadores necessitam para projectar e desenvolver aplicações. Esta arquitectura uniforme assegura

interoperacionalidade e uma única forma de abordagem para aceder às muitas e diferentes bases de dados relacionais (antes de aparecer o OLE DB).

Uma aplicação usa as seguintes funções e processos para aceder a uma fonte de dados usando a API do ODBC:

- Criar o ambiente de funcionamento - cria áreas de memória para variáveis globais e de ambiente e informação com as definições das conexões;
- Especificar a conexão - identifica o espaço de memória para guardar informação sobre uma conexão;
- Conectar - especifica uma expressão para a conexão, com a identificação do utilizador, da palavra-chave e da fonte de informação;
- Criar a expressão SQL - associa a expressão SQL com a conexão, podendo diferentes expressões de SQL ser associadas à mesma conexão, mas só uma de cada vez;
- Executar a expressão SQL - processa a expressão SQL usando o motor da base de dados;
- Buscar o resultado - recebe o resultado da expressão SQL (todas as linhas, a primeira, a última, a próxima ou a anterior) e busca informação sobre os resultados (número de linhas, ou numero de colunas);
- Libertar a expressão - liberta a expressão da conexão e assim é possível repetir o processo com a mesma expressão SQL ou outra, para a mesma conexão;
- Desconectar - retira a informação da desconexão;
- Libertar a conexão;
- Libertar ambiente - liberta a memória usada para o ambiente, incluindo variáveis, etc.

Com a utilização da API ODBC, é possível criar código independente e este pode comunicar com as várias bases de dados relacionais. Contudo, existe uma consideração importante na adopção deste método: enquanto que qualquer *driver* específico ODBC usa sempre as mesmas funções, estas podem não ser suportadas por outros *drivers*. Se a aplicação for construída para várias bases de dados algumas funções devem ser usadas com algum cuidado.

2.2.1.3. Quando é que se usa

Existem vários motivos para usar esta tecnologia e, entre eles, podem-se considerar o elevado desempenho, o total controlo sobre o interface e pouco *overhead* como factores relevantes para a escolha desta tecnologia.

A API do ODBC é considerada de muito difícil implementação em relação a outros interfaces que usam modelos de programação orientado aos objectos, mas oferece um elevado grau de controlo sobre a fonte de dados. Como é bastante fácil cometer erros durante o desenvolvimento, a API do ODBC fornece um excelente controlo de erros bem como bastantes detalhes nas mensagens dos erros. Na realidade, desenvolver, detectar e corrigir os erros (*debugger*) e fazer a manutenção de aplicações que usam a API do ODBC requer bastantes conhecimentos, experiência e muitas linhas de código. Regra geral os programadores preferem o acesso à informação usando um conjunto de objectos de mais alto nível tal como o ADO.

O ODBC não é aplicável a bases de dados não relacionais tais como as que usam o *indexed sequential access method* (ISAM), porque não há interfaces para fazer o posicionamento *(seek)* dos registos ou pesquisar os índices que não existem. O ODBC não foi projectado para aceder a dados do tipo ISAM.

Se a aplicação necessitar de acessos via ODBC bastante rápidos, mesmo que isso exija escrever bastante código (embora na documentação específica do ODBC existam bastantes exemplos e muito código que pode ser utilizado), então o ODBC é uma boa escolha.

2.2.2. *Data Access Objects* (DAO)

O DAO fornece acesso a bases de dados do tipo Microsoft *Jet Engine* (nativos), algumas bases de dados ISAM e qualquer fonte de dados ODBC. Historicamente, o

DAO é uma solução bastante popular quando é utilizado com fontes de dados do tipo Microsoft Access e ISAM tal como *Btrive*, *FoxPro*, *Paradox* e *dBase*.

As principais características do DAO são:

- Flexibilidade, com facilidades para aceder a muitas e diferentes fontes de dados;
- Desempenho que varia entre adequado e lento;
- Funcionalidades *Dynamic Data Languages* (DDL);
- Suporte para cursores complexos;
- Dificuldade de codificação.

2.2.2.1. Modelo de Objectos

O modelo de objectos do DAO é uma colecção de objectos que modelizam a estrutura de um sistema de bases de dados relacional. Com as propriedades e métodos fornecidos pelos objectos do DAO, podem-se executar todas as operações necessárias para a manutenção do sistema, isto é criar bases de dados, definição de tabelas, campos, índices, estabelecer relações entre tabelas, navegar e fazer perguntas à base de dados [Micr97B].

O motor Microsoft *Jet Database* traduz operações dos objectos de acesso em operações físicas nos ficheiros da base de dados.

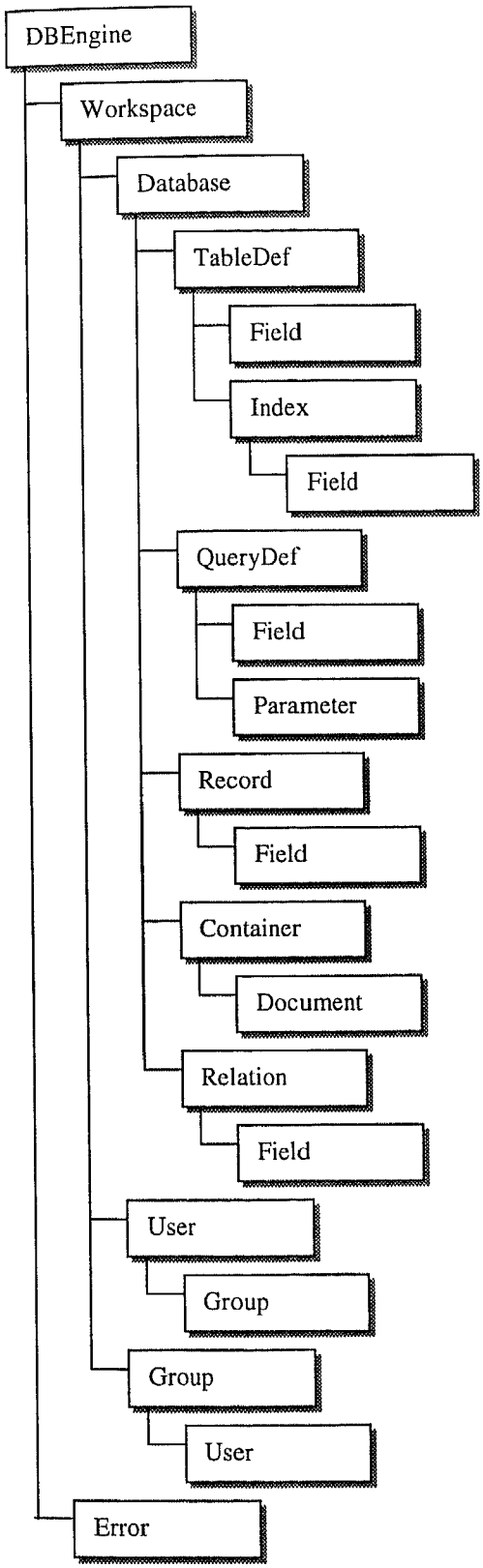

Figura II-2 - Modelo de Objectos do DAO

Existem 15 objectos no modelo de objectos do DAO:

- *DBEngine* - este objecto é a base do modelo de objectos do DAO e referencia os restantes objectos do modelo, bem como as propriedades do motor;
- *Workspace* - identifica e controla a sessão de um determinado utilizador. Este objecto contém informação sobre as bases de dados abertas e fornece mecanismos de transacções simultâneas;
- *Database* - representa uma base de dados com pelo menos uma conexão aberta. Esta conexão pode ser com uma base de dados do Microsoft *Jet* ou com uma fonte de dados externa ODBC;
- *TableDef* - contém os objectos dos campos e índices para descrever as tabelas da base de dados;
- *Recordset* - representa um conjunto de resultados através de um cursor a partir de uma determinada pergunta. O DAO tem cinco tipos destes objectos: *table, dynaset, snapshot, forward-only* e dinâmicos;
- *Container* - representa um conjunto de objectos na base de dados para o qual se definem permissões de um determinado grupo de trabalho;
- *Relation* - representa as relações entre os campos das tabelas e/ou perguntas. É possível utilizar este objecto quer para criar, apagar e modificar um tipo de relação e determinar quais as tabelas que fornecem campos que se relacionam, quer para forçar a integridade relacional ou para fazer operações de eliminação e actualizações em cascata;
- *Field* - representa um campo que pode constar em tabelas, perguntas, índices, relações ou *recordsets*. Este objecto contém informação do tipo do campo, tamanho, etc. e pode ser usado para ler e gravar dados num registo;
- *Index* - representa os índices das tabelas da base de dados;
- *Parameter* - representa os parâmetros associados ao objecto *QueryDef*. Estes parâmetros podem ser de entrada ou de saída;
- *Document* - contém informação sobre objectos individuais da base de dados (tais como tabelas, perguntas ou relações);
- *User* - representa a validação do utilizador e as suas permissões;
- *Group* - grupo de utilizadores que têm as mesmas permissões;

- *Error* - contém informação sobre os erros que vão ocorrendo durante uma operação DAO. Quando acontece mais do que um erro durante uma operação é criado mais do que um objecto deste tipo.

Cada objecto do tipo *Workspace* tem uma colecção de objectos do tipo *Database*. Cada *Database* representa todos os objectos que podem estar presentes numa base de dados individual. Destes objectos, os objectos *Recordset* são certamente os mais usados, fornecendo os meios para executar expressões SQL e manipular o conjunto de resultados. Os objectos *TableDef* fornecem um acesso simples a tabelas e aos seus campos e índices. O modelo de objectos DAO é relativamente complicado porque fornece bastantes funcionalidades através do Microsoft *Jet Engine* para diferentes tipos de bases de dados.

2.2.2.2. Acesso a dados

Existem 4 categorias de dados acessíveis através do DAO e do Microsoft *Jet Engine*:

- Bases de dados nativas Microsoft *Jet*: estes ficheiros de base de dados usam o mesmo formato que os ficheiros do Microsoft *Access*. Estas bases de dados são criadas e manipuladas directamente pelo Microsoft *Jet Engine* e fornecem a máxima flexibilidade e rapidez;
- Bases de dados externas: estas são as bases de dados ISAM em vários formatos, incluindo *Btrive, dBase III, dBase IV*, Microsoft *FoxPro* 2.0, 2.5 e *Paradox* 3.x e 4.0;
- Ficheiros de texto, folhas de cálculo Microsoft *Excel* ou *Lotus* 1-2-3;
- Bases de dados ODBC: estas incluem bases de dados relacionais que utilizam a norma ODBC, tal como Microsoft SQL *Server*.

O motor da base de dados Microsoft *Jet* traduz as operações sobre objectos DAO em operações físicas nos ficheiros das bases de dados, manipulando todos os mecanismos das diferentes bases de dados suportadas.

Uma aplicação típica baseada em DAO usa as seguintes operações para aceder a uma fonte de informação:

- Criar uma área de trabalho (*workspace*) - cria uma sessão de uma utilizador, incluindo a identificação do utilizador, a palavra-chave e o tipo de base de dados (tal como Microsoft *Jet* ou ODBC);
- Abrir uma conexão - especifica uma expressão para a conexão para uma determinada área de trabalho com informação da fonte de dados e do nome da tabela;
- Abrir uma tabela - executa uma pergunta (com ou sem parâmetros) e preenche a tabela do resultado;
- Usar o resultado - o resultado da pergunta está agora disponível para a aplicação e, dependendo do tipo de cursor, é possível alterar a informação de cada linha;
- Fechar os resultados - fecha o bloco de registos resultantes da pergunta;
- Fechar a base de dados - significa fechar as conexões de uma determinada base de dados;
- Fechar a área de trabalho.

Com o DAO é possível trabalhar directamente com tabelas e índices ISAM, o que representa uma verdadeira vantagem para quem usa o modelo de objectos DAO.

Com o DAO é possível executar operações DDL que afectam directamente a estrutura da base de dados, sendo possível, por exemplo, criar, modificar e apagar estruturas de tabelas.

Sendo uma tecnologia antiga, o DAO está limitado a fontes de informação que podem ser manipuladas exclusivamente com o Microsoft *Jet Engine*. Se uma aplicação requer o acesso a outro tipo de fontes de informação, o DAO não consegue fornecer esse acesso. Adicionalmente, o DAO não consegue construir perguntas usando cursores do lado do servidor. A utilização do DAO apresenta uma degradação do desempenho devido à utilização nativa do Microsoft *Jet database engine*.

2.2.2.3. Quando é que se usa

O DAO é a única tecnologia de acesso que suporta operações com 16 bits e, por isso, se a aplicação tiver que ser executada num ambiente de 16 bits então o DAO é a única escolha como tecnologia de acesso.

Se a aplicação tem que aceder a dados nativos do Microsoft *Jet* e a recursos ODBC, o DAO fornece um modelo de programação consistente (a alternativa é considerar a utilização do OLE DB e aceder através do modelo universal disponibilizado pelo ADO (secção 2.3)).

Se a aplicação tem que aceder a fontes de informação remotas, o DAO juntamente com o Microsoft *Jet engine* é a pior escolha porque, além de ser lento, consome bastantes recursos quando comparado com novas tecnologias, tais como o ADO e o RDO.

Quando se tem um grande conhecimento de trabalho em DAO, se tem bastante código em DAO e se necessita de fazer uma extensão da aplicação que usa esta tecnologia, então o DAO deve ser mantido. Se, pelo contrário, a aplicação necessita de aceder a outras fontes de informação, o DAO não vai conseguir realizar esse acesso. Vai então ser necessário fazer um novo projecto, codificar novamente e tirar partido de todas as capacidades do ADO.

2.2.3. *Remote Data Objects* (RDO)

O RDO foi especificamente desenhado para aceder remotamente a fontes de informação relacionais do tipo ODBC, facilitando portanto a utilização do ODBC sem a complexidade da API do ODBC. O RDO é o método principal de acesso a fontes de informação tais como SQL *Server*, *Oracle* e bases de dados relacionais que disponibilizam *drivers* para o ODBC [Micr98].

As características gerais do RDO são:

- Simplicidade (quando comparado com a API do ODBC) ;
- Controlo programático de cursores;
- Cursores complexos, incluído funcionalidades de *batch;*
- Capacidade de retornar várias tabelas a partir de uma única pergunta;
- Respostas síncronas ou assíncronas;
- Reutilização e mudança de características dos objectos;
- Excelente controlo de erros.

Comparado com a tecnologia DAO, o RDO é mais pequeno, mais rápido e uma alternativa mais sofisticada. O RDO é especialmente capaz de construir e executar perguntas e de receber qualquer tipo de resultados, incluindo aqueles que resultam de procedimentos no servidor, parâmetros ou *status.*

2.2.3.1. Modelo de Objectos

O RDO é uma pequena camada baseada num modelo de objectos que fica "por cima" da API do ODBC, dependendo do *driver* ODBC e do motor da base de dados para muitas funcionalidades. O acesso a bases de dados usando RDO é praticamente exclusivo de bases de dados relacionais ODBC.

O modelo de objectos do RDO inclui objectos com propriedades e métodos especificamente desenhados para trabalhar com procedimentos remotos e com os seus argumentos, com parâmetros de entrada e saída e como retorno de valores de controlo.

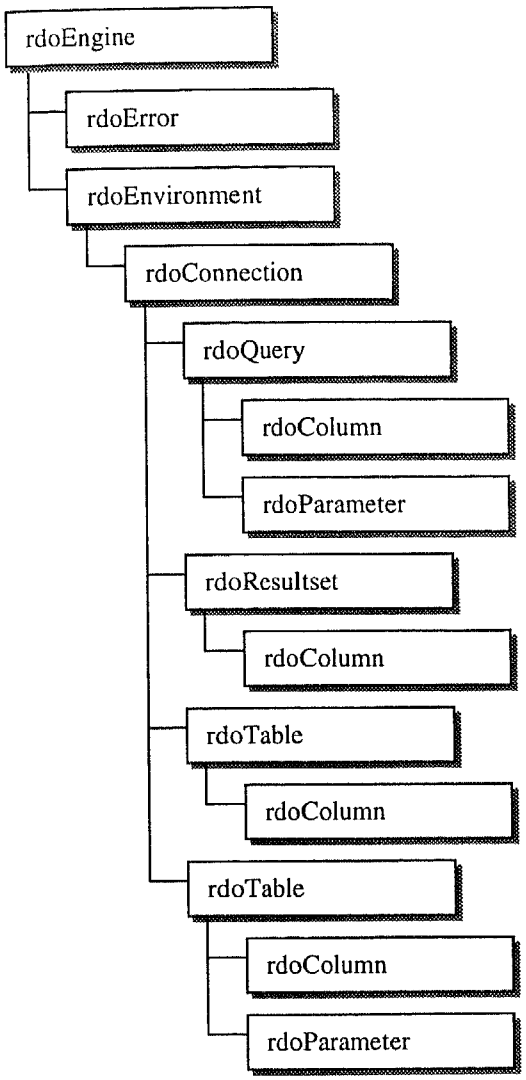

Figura II-3 - Modelo de Objectos do RDO

Existem 10 objectos no modelo RDO:

- *rdoEngine* – é o objecto base do RDO, sendo criado automaticamente quando a aplicação acede a primeira vez aos dados;
- *rdoError* - guarda todos os erros do ODBC e mensagens geradas pelo RDO. Este objecto também é criado automaticamente;
- *rdoEnvironment* - define o raio de acção de um conjunto lógico de conexões e transacções de um terminado utilizador. Este objecto contém as conexões abertas e alocadas (ainda não abertas) e fornece mecanismos de transações simultâneas. Contém também as definições de segurança para operações com a base de dados. Este objecto também é criado automaticamente;

- *rdoConnection* - representa uma conexão aberta para uma fonte de dados remota ou uma conexão alocada mas ainda não aberta;
- *rdoQuery* - representa uma expressão SQL com ou sem parâmetros;
- *rdoColumn* - representa uma coluna de informação, incluindo o tipo de dados e as propriedades comuns;
- *rdoParameter* - representa os parâmetros associados ao objecto *rdoQuery*. Os parâmetros das perguntas podem ser de entrada ou de saída;
- *rdoRecordset* - um conjunto de linhas retornadas por uma determinada pergunta;
- *rdoTable* - representa as definições de uma tabela ou de uma vista armazenadas na base de dados;
- *rdoPreparedStatement* - representa uma expressão SQL com ou sem parâmetros. Este objecto é obsoleto nas últimas versões do RDO, já que o objecto *rdoQuery* faz as mesmas funções de uma maneira bastante mais simples.

2.2.3.2. Acesso a dados

O RDO é a maneira mais popular e mais eficiente para aceder, via ODBC, a bases de dados relacionais. Com o RDO, é possível aceder a um conjunto de dados sem grandes preocupações com cursores, ou criar ambientes complexos de actualizações com um conjuntos de cursores do lado do cliente que se adaptam a diversas situações. É possível também limitar o número de linhas retornadas e das mensagens geradas pela fonte de dados sem que isto provoque a paragem da extracção de informação.

Uma importante capacidade do RDO e que não está disponível no DAO (na altura da construção destes dois métodos não existia ADO) é a de executar perguntas e procedimentos remotos que retornem múltiplas tabelas de resultado. Esta capacidade elimina bastante redundância de tráfego e *overhead.*

A maior parte dos métodos do RDO podem ser executados sincronamente ou assincronamente e, através de eventos, a aplicação é informada de quando é que acabou ou vai começar uma determinada acção na base de dados. Através das

conexões assíncronas, as aplicações ficam completamente livres para realizar outras operações, não ficando paradas ou presas aquando das extracções das perguntas.

O RDO tem a capacidade de dissociar e voltar a associar um objecto *rdoQuery* de uma determinada pergunta, funcionalidade esta pode ter algum interesse quando se pretende fazer a mesma pergunta a várias fontes de dados.

Tal como o ADO, o RDO também inclui as funcionalidades de *batch updates*. Com esta funcionalidade, é possível ter um conjunto de dados, modificá-los e só no fim de todas as modificações actualizar os dados na fonte. Este funcionalidade aumenta significativamente a rentabilidade do servidor, bem como provoca menos tráfego na rede e reduz os *overheads* do ODBC.

Uma aplicação típica baseada em RDO usa as seguintes operações para aceder a uma fonte de informação:

- Cria o ambiente - aloca memória para dados e cria informação para as conexão definidas;
- Abre a conexão - especifica a expressão de abertura de uma conexão, que é constituía pelo nome da fonte de informação, nome de utilizador, palavra chave, base de dados por omissão, nome da máquina onde esta a fonte de informação e nome do identificador de dados;
- Procura resultados - executa uma pergunta e recebe informação com o resultado;
- Usa o resultado - o resultado da pergunta fica disponível para uso. Dependendo do tipo de cursor, é possível navegar nos dados e alterá-los, independentemente destes residirem no lado do cliente ou do servidor;
- Fecha a conexão - desconecta a fonte de informação;
- Liberta o ambiente - liberta a memória alocada na criação do ambiente.

Com o RDO é possível criar código independente da base de dados, que se adapta às várias bases de dados suportadas via ODBC.

2.2.3.3. Quando é que se usa

Se a aplicação estiver implementada/projectada em RDO, não há vantagem em alterar a forma de acesso. Mas se a aplicação tem necessidade de comunicar com outras fontes de informação não suportadas via ODBC, então a migração da aplicação para ADO será aconselhável. Se a aplicação estiver a ser construída de raiz, então a escolha do método de acesso a dados deve incidir no ADO.

2.2.4. ODBCDirect

O ODBCDirect é um modo alternativo ao DAO para aceder directamente a fontes de informação via ODBC, ou seja sem passar pelo Microsoft *Jet Engine*, e tirando partido de todas as funcionalidades e vantagem do acesso remoto.

O ODBCDirect realmente é o RDO mas com os objectos do DAO. Quando se usa o ODBCDirect, o DAO não carrega o motor próprio, ou seja, o Microsoft *Jet*, mas carrega as bibliotecas do RDO 2.0. Quando comparado com a norma DAO, existem algumas vantagem na utilização do ODBCDirect, entre as quais:

- Melhor desempenho;
- Acesso às funcionalidades específicas da fonte de dados (como por exemplo a um procedimento remoto ou a um conjunto de parâmetros);
- Perguntas assíncronas;
- Actualizações de dados de vários registos de uma única vez (*batch updates*).

O ODBCDirect é um melhoramento relativamente à norma DAO que usa o Microsoft *Jet Engine*. Contudo, comparado com o novo ADO ou com o RDO, o ODBCDirect é mais lento e com menos funcionalidades do que os outros métodos de acesso.

2.2.4.1. Modelo de Objectos

O modelo de objectos do ODBCDirect é essencialmente um subconjunto do modelo de objectos do DAO. No entanto, não contém nenhum objecto específico do

Microsoft *Jet*, nomeadamente os objectos *user*, *group*, *container* e *document*, e também não contém os objectos de definições, nomeadamente *tabledef* e *relation*, mas contém um novo objecto, *Connection*. Com este novo objecto as aplicações podem abrir múltiplas conexões assíncronas a várias bases de dados e realizar outras operações enquanto as conexões estão ocupadas. A figura mostra o modelo de objectos do ODBCDirect [Micr97A].

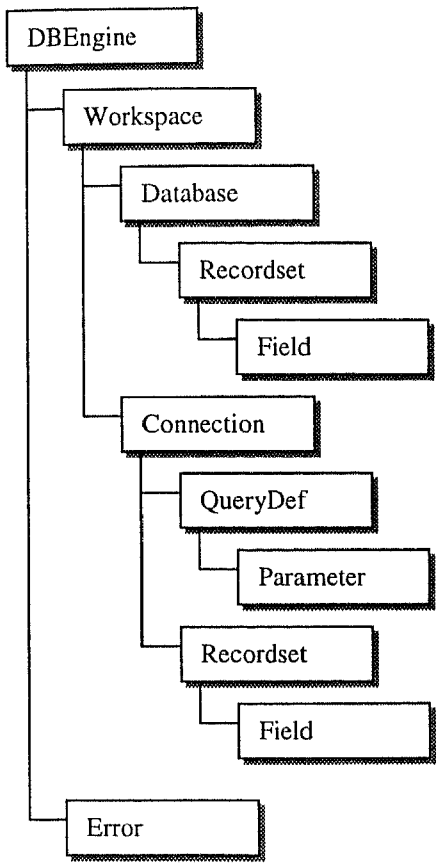

Figura II-4 - Modelo de Objectos do ODBCDirect

Existem nove objectos no modelo de objectos do ODBCDirect:

- *DBEngine* - este objecto é a base do modelo de objectos do ODBCDirect e é o objecto de entrada para os restantes objectos do modelo bem como para as propriedades do motor;
- *Workspace* - identifica e controla a sessão de um determinado utilizador. Este objecto contém informação sobre as bases de dados abertas e fornece mecanismos de transacções simultâneas;

- *Database* - representa uma base de dados ODBC com pelo menos uma conexão aberta;
- *Connection* - representa a conexão ODBC para uma determinada base de dados incluindo o nome do utilizador, a palavra-chave e a base de dados;
- *QueryDef* - armazena as perguntas em SQL, com zero ou mais parâmetros contidos na base de dados ODBC;
- *Recordset* - representa a resposta a uma determinada pergunta através de um cursor. O ODBCDirect tem cinco tipos deste objecto: *table, dynaset, snapshot, forward-only* e dinâmicos;
- *Field* - representa um campo que pode ser usado em tabelas, perguntas, índices, relações ou *recordsets*. Este objecto contém informação e pode ser usado para ler e gravar dados num registo;
- *Parameter* - representa os parâmetros associados ao objecto *QueryDef*. Estes parâmetros podem ser de entrada ou de saída;
- *Error* - contém informação sobre os erros que vão ocorrendo durante uma operação ODBCDirect. Quando acontece mais do que um erro durante uma operação são criados mais do que um objecto deste tipo.

2.2.4.2. Acesso a dados

Quando uma aplicação usa ODBCDirect na realidade está a usar RDO. Mas então porque não usar directamente RDO? Uma transformação do código DAO para suportar ODBC não é simplesmente uma troca de objectos para RDO mas é bastante simples a alteração do código para ODBCDirect. Este tipo de situação acontece quando uma aplicação usa DAO e necessita de aceder a fontes de informação ODBC.

Uma aplicação típica baseada em ODBCDirect usa as seguintes operações para aceder a uma fonte de informação:

- Criar uma área de trabalho (*workspace*) - cria uma sessão de um utilizador, incluindo a identificação do utilizador e palavra-chave;

- Abrir uma conexão - especifica uma expressão para a conexão para um determinada área de trabalho, com informação da fonte de dados e do nome da tabela;
- Abrir uma tabela - executa uma pergunta (com ou sem parâmetros) e preenche a tabela do resultado;
- Usar o resultado - o resultado da pergunta está agora disponível para a aplicação. Dependendo do tipo de cursor é possível alterar a informação de cada linha;
- Fechar os resultados - fecha o bloco de registos resultantes da pergunta;
- Fechar a base de dados - significa fechar as conexões a um determinada base de dados;
- Fechar a área de trabalho.

Mesmo que o ODBCDirect use RDO, existem pequenas funcionalidades que são tratadas de maneira diferente. Por exemplo, com o ODBCDirect deve ser criada uma área de trabalho (*Workspace Object*) por cada cursor criado para uma diferente conexão. As transacções são coordenadas ao nível da área de trabalho e não a nível das conexões ou das bases de dados. Finalmente, apesar do ODBCDirect suportar operações assíncronas, é necessário pedir informação sobre o estado da operação (não existem *callbacks*).

Comparado com o DAO, um dos melhoramentos no ODBCDirect é actualizações de dados de vários registos de uma única vez (*batch updates*) e de uma maneira optimista. O termo optimista significa assumir a existência de um único processo a aceder a estes dados. Com esta funcionalidade, a aplicação pode guardar os dados localmente e fazer a actualização destes no servidor, operação que tende a ser bastante eficiente porque reduz o tráfico da rede e os *overheads.*

O ODBCDirect é uma boa tecnologia mas está limitada às bibliotecas existentes para o ODBC. Se a aplicação necessita de outras fontes de informação, então o ODBCDirect não é uma boa solução.

2.2.4.3. Quando é que se usa

O ODBCDirect é uma escolha aceitável quando a aplicação necessita de executar perguntas, procedimentos remotos numa determinada base de dados relacional ODBC ou necessita de algumas capacidades específicas do ODBC, tal como actualizações de dados de vários registos de uma única vez ou perguntas assíncronas. Todas as capacidades do ODBCDirect estão no ADO.

Se se possui um grande conhecimento de trabalho em ODBCDirect, se se tem bastante código em ODBCDirect e se se necessita de fazer uma extensão de uma aplicação que usa esta tecnologia então o ODBCDirect deve ser mantido. Se, pelo contrário, a aplicação necessita de acesso a fontes de informação não OBDC, o ODBCDirect não vai conseguir aceder a essas fontes: vai ser necessário fazer um novo projecto, codificar novamente e tirar partido de todas as capacidades do ADO.

2.2.5. *ActiveX Data Objects* (ADO)

O ADO foi concebido para ser um interface ao nível aplicacional de fácil utilização para qualquer fonte de informação OLE DB, incluindo bases de dados relacionais ou não, *e-mail*, texto ou gráficos bem como qualquer *driver* ODBC existente. Através da tecnologia de acesso ADO, é possível aceder a toda a informação existente dentro de uma determinado domínio.

O ADO é de fácil utilização, independente da linguagem, não consome muitos recursos, tenta minimizar o tráfego na rede e tem poucas camadas entre a aplicação cliente e a fonte de informação. Tudo isto para proporcionar um acesso leve e de elevado desempenho.

As características gerais são:

- Facilidade de utilização;
- Elevado desempenho;
- Controlo programático de cursores;

- Cursores complexos, incluindo *batch, server* e *client-side;*
- Capacidade de retornar várias tabelas de resultados para um mesma pergunta;
- Possibilidade de executar perguntas síncronas, assíncronas e orientadas por objectos;
- Objectos reutilizáveis e com propriedades mutáveis;
- Boa gestão de recursos de memória *cache;*
- Flexibilidade, pois funciona com todas as bases de dados existentes e todas as fontes de informação disponíveis via OLE DB;
- Bom controlo de erros.

A universalidade aplicacional e a simplicidade semântica do ADO implicam um treino mínimo da equipa de programação, facilidade no desenvolvimento de aplicações e custos de manutenção relativamente baixos.

2.2.5.1. Modelo de Objectos

O modelo de objectos ADO define um conjunto (*collection*) de objectos programáveis que suportam COM e OLE Automation para tirar partido de uma tecnologia chamada OLE DB. O modelo de objectos ADO, quando comparado com outros modelos de objectos de acesso a bases de dados, tais como RDO ou DAO, possui menos objectos e de uso mais simples.

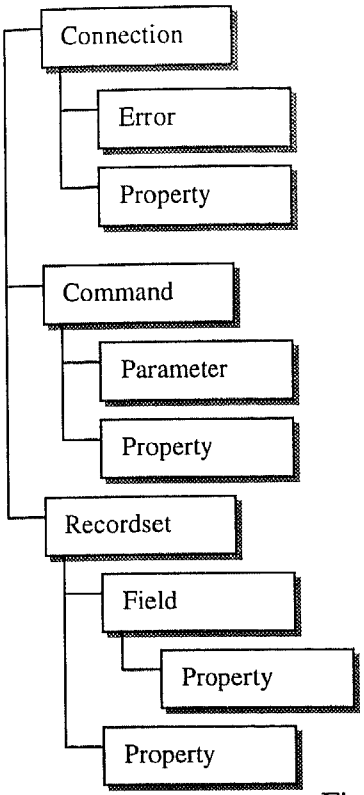

Figura II-5 - Modelo de Objectos ADO

Existem sete objectos no modelo ADO:

- *Connection* - este objecto contém informação sobre a conexão, incluindo o tipo de cursor, a expressão de conexão, o *time-out* da pergunta, o *time-out* da conexão e a base de dados por omissão;
- *Error* - contém informação extra sobre os erros disponibilizados pelo fornecedor de informação. Porque pode haver mais que um erro de cada vez, pode existir um conjunto de objectos deste tipo (*Error collection*);
- *Property* - este objecto é criado conforme a fonte de informação que lhe está associada, podendo conter dois tipos de propriedades: intrínsecas ou dinâmicas. As propriedades intrínsecas são aquelas que já estão implementadas e disponíveis quando este objecto é criado. As dinâmicas são propriedades disponibilizadas pela fonte de informação sob a forma de conjunto no objecto ADO. Por exemplo, uma propriedade pode indicar se um objecto tabela (objecto *Recordset)* suporta transacções ou actualizações. Esta grande capacidade do ADO permite a presença de interfaces novos e especiais [Micr99F];

- *Command* - mantém a informação de comandos tais como expressões de perguntas, definição de parâmetros, etc. É possível executar uma expressão de comando a partir de um objecto *Connection* (uma pergunta) que teve como resultado um conjunto de linhas sob a forma de um objecto tipo tabela (objecto *Recordset*) sem definir este objecto. O objecto *Command* é bastante útil quando se quer definir uma pergunta com parâmetros ou executar um procedimento remoto (*stored procedure*) ou este retorna parâmetros. Este objecto suporta um conjunto de propriedades para descrever o tipo e o objectivo da pergunta, ajudando por isso o ADO a optimizar esta operação;
- *Parameter* - cada parâmetro associa-se a um objecto *command* que usa um conjunto de objectos *parameter* para guardar a sua informação. Este objecto é criado automaticamente quando são enviadas as expressões de *query* para a base de dados. Contudo pode-se programar esse conjunto de parâmetros para melhorar o desempenho;
- Objecto *Recordset* - é um conjunto de linhas provenientes de uma pergunta, incluindo o cursor para essas linhas. É possível abrir um objecto deste tipo (basta executar uma *query*) sem abrir explicitamente um objecto para a conexão. Contudo, se for criado um objecto conexão *(Connection object)*, é possível abrir múltiplos objectos tabela *(object Recordset)* na mesma conexão.
- *Field* - contém informação sobre uma coluna de uma determinada tabela *(Recordset)*. O objecto tabela *(Recordset)* usa um conjunto deste objectos para guardar informação de cada coluna da tabela. Este objecto contém a informação do tipo de dados, precisão, escala numérica, etc.;

2.2.5.2. Acesso a dados

O interface ADO é o único que é necessário conhecer para todas as aplicações do tipo cliente/servidor, Web ou não. Uma das características do ADO é disponibilizar e usar as propriedades específicas de cada fonte de informação não interessando qual a fonte de dados usada. o ADO é pois totalmente flexível e adaptável aos requisitos das aplicações para acederem aos diferentes dados.

O ADO (tal como o RDO) inclui uma biblioteca cliente de cursores que suporta múltiplas actualizações optimistas, isto é actualizações em que é possível através do resultado de uma pergunta modificar os dados conforme necessário e só depois fazer a actualização. Com isto, reduz-se os *overheads* no servidor e na rede, resultando consequentemente um aumento do desempenho.

Uma importante característica do ADO é capacidade de manipulação do dados em *cache*, através do *Remote Data Service (RDS),* que permite ter opcionalmente dados em memória na máquina cliente. Com o RDS, é possível transmitir registos de uma tabela para a frente ou para trás entre o servidor e o cliente. Por exemplo, uma aplicação poderá usar bastantes dados como resultado de uma pergunta que ficam armazenados do lado do cliente, reduzindo assim o número de pedidos por parte do cliente e melhorando o desempenho da aplicação do lado do cliente. Para além disso é possível abrir e preencher um objecto tabela assincronamente, podendo portanto no momento do preenchimento desse objecto estar o cliente desconectado do servidor. Existe um aumento do desempenho visto que deixa o cliente livre para executar outras tarefas, enquanto os registo retornam ao servidor.

Uma aplicação típica baseada em ADO usa as seguintes operações para aceder a uma fonte de informação:

- Criar um objecto *Connection* – especifica-se a expressão de conexão com informação do nome da fonte de dados, informação do utilizador e palavra-chave, *time-out* da conexão, base de dados por omissão e localização do cursor. Este objecto representa apenas uma sessão com a fonte de informação, sendo possível controlar as transacções entre o objecto de conexão e a fonte de informação através dos métodos *BeginTrans*, *CommitTrans* e *RollBack*.
- Abrir a conexão – abre-se a conexão com a fonte de informação;
- Executar um comando SQL - depois da conexão estar estabelecida, pode-se executar uma pergunta. Esta pergunta poderá ser executada assincronamente e o resultado também pode ser assíncrono;
- Usar o resultado da pergunta - dependendo do tipo de cursor, é possível manipular a informação quer do lado do cliente quer do lado do servidor;
- Terminar a conexão.

Apesar do modelo de objectos do ADO ter muitas propriedades e métodos, usá-lo é de facto bastante simples. O ADO pode representar o futuro na tecnologia de acesso a informação.

2.2.5.3. Quando é que se usa

O ADO é actualmente a principal tecnologia de acesso a bases de dados da Microsoft. Esta tecnologia em conjunto com o OLE DB é a recomendada para aplicações que contenham uma qualquer fonte de dados. Na construção das novas aplicações este método é o método recomendado.

Para considerar a migração das aplicações para o ADO, deve-se considerar as características e benefícios que o ADO oferece são ou não suficientes para justificar tal conversão. O antigo código escrito usando RDO ou DAO não é automaticamente convertido para ADO; no entanto qualquer das soluções previamente desenvolvidas usando outra tecnologia de acesso pode ser implementada usando ADO.

2.2.6. Escolher a melhor tecnologia

Existem muitas questões a considerar antes de escolher uma determinada tecnologia de acesso a dados. Eis algumas:

- É um novo projecto ou uma modificação de uma aplicação existente que usa um método "antigo"? Para modificações, deve-se continuar com o método já usado. Para um futuro imediato, é mais uma decisão de custo. Contudo se a aplicação crescer, pode aceder a novas fontes de informação não suportadas pelo método que está a ser usado.
- Onde é que está a informação? Está na Web, num servidor remoto, ou está localmente na máquina do utilizador? Se a informação está na máquina do utilizador, a necessidade de criar um servidor diferente para guardar e manipular a informação está posta de lado. Se a informação é remota, como é o acesso? O

que é que acontece se a aplicação não se consegue conectar? Devem-se usar tecnologias que suportem mecanismos assíncronos, tais como ADO ou RDO?

- O que é que os programadores ou as equipas de desenvolvimento estão habituados a usar? Têm alguma experiência em ADO, RDO, DAO ou ODBC? Será que vale a pena dar formação a uma equipa de desenvolvimento para aprender ADO?
- Será que a aplicação requer acessos a bases de dadas relacionais e não relacionais? Existe o interface OLE DB para todas ?
- Será que "todas" as aplicações já usam a API do ODBC? Se se continuar a usar ODBC, será que aplicação não necessita no futuro de aceder a outros tipos de fontes de informação não suportadas pelo ODBC?

É possível usar diferentes tecnologias de acesso à informação para implementar o acesso a essa mesma informação e diferentes estratégias de comunicação. O quadro seguinte compara os métodos e as respectivas aplicações.

A melhor escolha é	Se a aplicação necessita...	Descrição
ADO	Informação em *mainframes*	Com o Microsoft SNA Server é possível acessos a informação armazenados em VSAM, CICS, IMS e AS/400
	Reengenharia	Para as aplicações existentes, será sempre de considerar a troca para ADO. Como alternativa, pode-se usar o método já usado.
	Novos desenvolvimentos	Para todos os novos desenvolvimentos, o método de acesso deve ser sempre o ADO
	Uniformizar informação de fontes diferentes	O ADO é o único método que pode assegurar o maior número de tipos de acessos a fontes diferentes
	Rápido desenvolvimento	O ADO ajuda a minimizar o custo de desenvolvimento, devido à sua uniformidade, é consistente e fácil de usar. É possível treinar novos programadores uma vez e beneficiar continuadamente disso.
	Elevado desempenho	O ADO é bastante rápido

	Internet Information Server (IIS) e *Active Server Pages* (ASP).	Se a aplicação usa IIS com ASP para gerar HTML independente do browser, tem que se usar ADO.
	Acesso a informação não normalizada.	É perfeitamente possível escrever um *driver* especifico para uma determinada fonte de informação e usar ADO nos clientes para acesso
RDO	Acessos rápidos a uma base de dados que suporte ODBC	O RDO é bastante rápido
ODBCDirect	Acesso a base de dados que suporte ODBC	O ODBCDirect fornece uma melhoria de desempenho relativamente ao antigo DAO.
DAO	Melhoramentos relativamente ao método existente de acesso DAO	O DAO fornece um modelo consistente de programação para situações em que tem que ser fornecido através do Microsoft Jet. Se o código desta tecnologia for bastante e se se passar por uma pequena revisão no projecto da aplicação, codificação e desempenho não há grandes razões para trocar de tecnologia de acesso.
	Código em 16 bits	O DAO é a única possibilidade
ODBC	Rápidos acesso a bases de dados que suportem ODBC	Se escrever muito código com uma manutenção complexa não for problema, então a API do ODBC é uma boa opção

Tabela II-1 – Resumo das tecnologias

2.3. Universal Data Access

2.3.1. Introdução

As equipas de desenvolvimento estão a construir novos sistemas de informação baseados na Web e a tecnologia de acesso deve satisfazer novos e complexos cenários. As gerações anteriores de aplicações acediam a dados em *mainframes* mas, agora os programadores desenvolvem aplicações baseadas na Web, que necessitam de

acesso a informação distribuída ou espalhada por vários sítios com hardware, sistemas operativos e sistemas de armazenamento completamente distintos.

Normalmente a informação encontra-se disponível em diferentes sistemas de armazenamento ou tipos de bases de dados tais como:

- Ficheiros AS/400
- Bases de Dados Relacionais (ex.: *Oracle*, SQL *Server*, *Informix*)
- Folhas de Cálculo (ex.: Microsoft *Excel*)
- Base de Dados Pessoais (ex.: Dbase)
- Ficheiros do *Word*
- Ficheiros de Texto
- *E-Mail*
- *Virtual Sequencial Access Method* (VSAM)
- *Information Management System* (IMS)
- *Custormer Information Control System* (CICS).

Seria bom ter um único método de acesso para todos estes tipos de dados. Isto significaria que seria possível aceder a informação com um único método de acesso, quer se tratasse de uma base de dados relacional, uma base de dados num *mainframe*, ou simplesmente uma folha de cálculo ou um ficheiro de texto.

A construção de uma aplicação que use diferentes tipos de dados é mais complicada do que parece à partida já que é necessário criar/implementar diferentes métodos de acesso, o que implica usar diferentes *Aplication Programing Interfaces* (APIs), interfaces *Component Object Model* (COM), automatismos diferentes e diferentes procedimentos para que as transacções da informação sejam asseguradas.

Uma solução era colocar todos os diferentes tipos de dados num único tipo de armazenamento, como por exemplo, uma base de dados relacional. Só que esta aproximação pressupõe um único método de acesso e isso pode ser inaceitável pelas seguintes razões:

1. Seria necessário mover grandes quantidades de informação para uma única base de dados e reescrever as ferramentas para aceder a uma única localização da fonte de informação.
2. Seria necessário reaprender o uso dessas novas ferramentas.
3. Não seria possível implementá-la porque provavelmente não se tem controlo sobre todos os dados.
4. Uma base de dados relacional ou uma única tecnologia de armazenamento de informação não é a melhor maneira de armazenar os diferentes tipos de informação, porque a estrutura de informação apropriada está directamente relacionada com essa informação e com a forma como é manipulada.

2.3.2. Como é que o OLE DB e ADO fornecem *Universal Data Access*

Como já foi referido, a Microsoft desenvolveu várias tecnologias que se relacionam entre si, nomeadamente o ADO, OLE DB, e *Open Database Connectivity* (ODBC) e designou-as *Universal Data Access* [Laza98].

Se um único tipo de armazenamento para todos os tipos de dados não é a melhor solução, qual será a alternativa? A solução para uma aplicação aceder a vários tipos de dados continuamente é o uso do OLE DB como fornecedor de dados e ADO como tecnologia de acesso a esses dados. A utilização da tecnologia OLE DB e do ADO é a melhor abordagem para a construção de aplicações, desde uma pequena aplicação que use uma estação de trabalho única, até aplicações Web de grande escala.

O OLE DB é um vasto conjunto de interfaces COM, que consegue fornecer um acesso uniformizado a dados armazenados em diferentes fontes de informações. Estes interfaces, por sua vez, são melhores para suportar as funcionalidades de acesso proprietárias de cada tipo de fonte de dados porque os interfaces OLE DB conseguem transmitir as funcionalidades nativas dos diferentes tipos de armazenamento de informação.

OLE DB é apropriado para fontes de informação do tipo relacional, não relacional, incluindo *mainframes* VSAN, ficheiros AS/400, informação de regiões CICS, bases de dados hierárquicas IMS, e muitos outros tipos de armazenamentos de dados como se ilustra na figura II-6.

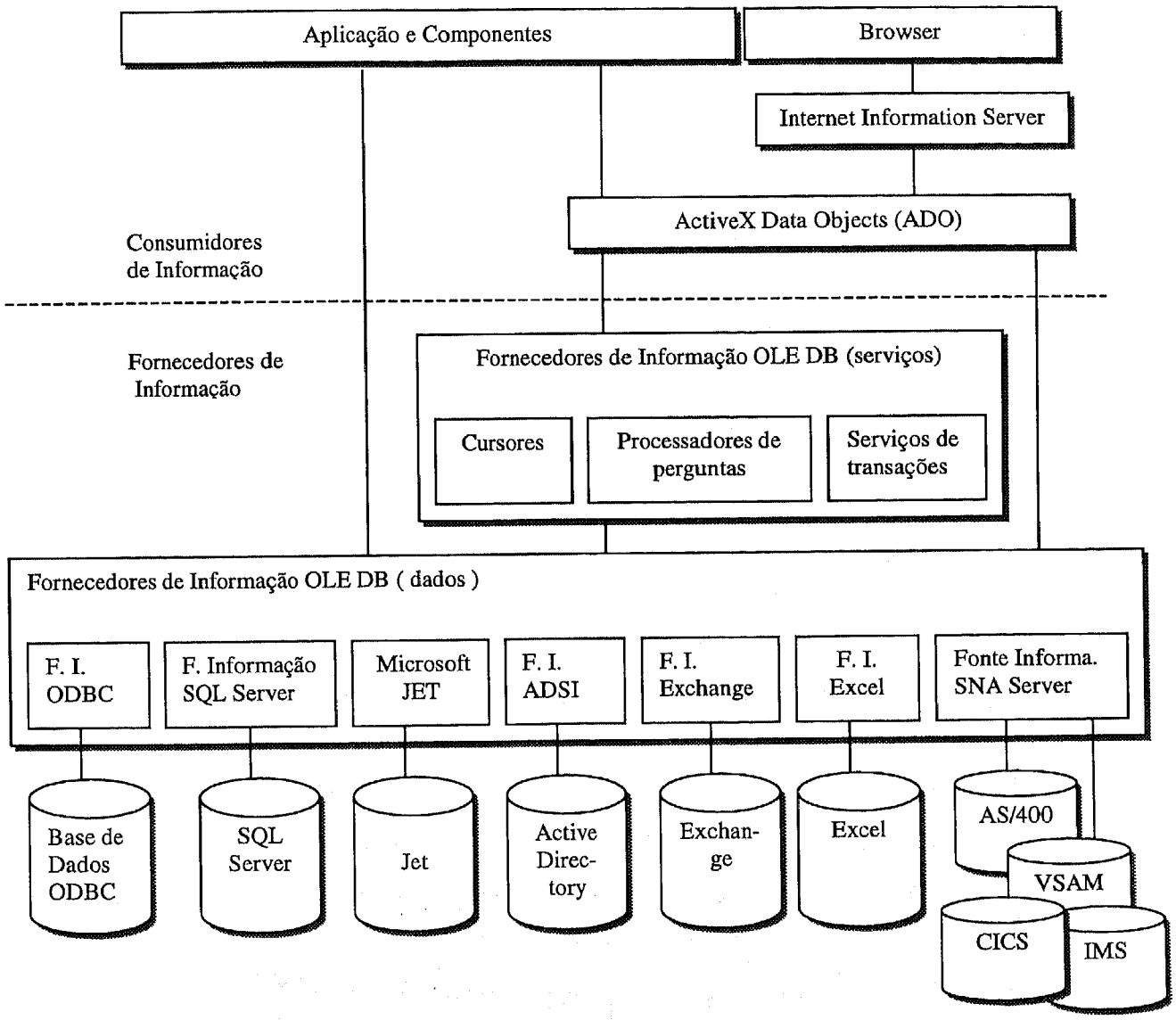

Figura II-6 - *Universal Data Access*

Os componentes do OLE DB consistem em fornecedores de informação que disponibilizam dados para os consumidores de informação e para componentes

(serviços) que processam e transportam informação (tais como *query processers*, *cursor engines* e *business services*).

Porque o OLE DB e os interfaces ADO são baseados em objectos COM, que por si só possuem um grande conjunto de serviços integrados (incluindo transacções, segurança e filas de mensagens) para suportar o mais largo espectro de cenários de aplicações com acesso a informação, proporcionam um conjunto elevado de funcionalidades para a construção de aplicações com diferentes tipos de dados. Os métodos, propriedades e eventos são uniformes e podem ser usados por todas as ferramentas de desenvolvimento da Microsoft, bem como por algumas dos concorrentes, tais como da Borland (nomeadamente o Borland C++ Builter 4.0) e outros.

Uma indicação de quão universal é esta aproximação é a lista de ferramentas de desenvolvimento da Microsoft que usam tecnologias de acessos de dados baseadas de COM, OLE DB e ADO [Will99]:

- Microsoft Visual Basic
- Microsoft Visual C++
- Microsoft Visual J++
- Microsoft Visual FoxPro
- VBScript (incluindo *Active Server Pages extensions*)
- JScript (incluindo *Active Server Pages extensions*)
- Visual Basic for Applications (incluindo todas as aplicações Microsoft Office).

É importante mencionar que OLE DB com ADO dispõem de vantagens em relação a custo e desempenho em relação ao ODBC de duas maneiras:

1. Os *drivers* ODBC têm que implementar um motor relacional SQL para disponibilizar informação não relacional;
2. Serviços como cursores e processamento de perguntas (*querys*) têm que ser implementados a nível do *driver* ODBC, representando um custo de desenvolvimento ODBC, bem como um consumo de recursos sob a forma de motores de cursores múltiplos e processadores de perguntas (*querys*). Com o OLE DB, componentes de serviços reutilizáveis tratam de tarefas de processamento para uma variedade de fornecedores de informação.

Para integrar ODBC com os outros tipos de dados, a Microsoft fornece o tradutor de informação (*data provider*) OLE DB/ODBC, assegurando a continuação do suporte para uma grande variedade de *drivers* ODBC de bases de dados relacionais que existem actualmente, e sendo possível aceder à mesma informação via OLE DB como era possível aceder via ODBC e com o mesmo desempenho.

Uma aplicação usará provavelmente ADO para comunicar com diferentes tipos de dados usando o *driver* para o OLE DB, mas é possível utilizar directamente o interface OLE DB. Regra geral, não é necessário criar ou desenvolver os *drivers* da fonte de dados para o OLE DB mas, se uma determinada aplicação necessitar de usar uma fonte de dados que não seja suportada pelo OLE DB, é conveniente ou aconselhável desenvolver o *driver* para o OLE DB em vez de desenvolver um acesso directo da aplicação à fonte de dados.

Porque os interfaces OLE DB usam ponteiros e estruturas por razões de desempenho e como a informação é partilhada entre componentes, estes interfaces não são susceptíveis de serem chamados directamente das ferramentas Visual Basic e Visual J++. Normalmente deverá ser usado ADO para acesso e manipulação de dados através do OLE DB.

Genericamente, a ideia de utilizar *Universal Data Access* pressupõe uma construção simples e independente das fontes de informação usando o modelo OLE DB com ADO e, consequentemente, um baixo custo de implementação do modelo de acesso à informação.

2.4. ADO versus JDBC

O objectivo desta secção é fazer a comparação entre o interface Java – *Java Database Connection* (JDBC) e o Microsoft® *Windows Foundation Classes* (WFC) *ActiveX*® *Data Objects* (ADO) que é disponibilizado com o Microsoft Visual J++™.

Especificamente, o modelo de objectos, o modelo de programação e as funcionalidades do JDBC vão ser comparados com os do ADO/WFC. O resultado desta comparação mostrará que a API do JDBC é limitada à transferência de dados a partir de bases de dados SQL para os *buffers* de clientes Java e, por isso, o JDBC não fornece um bom modelo de interacção com a informação. Não permite o acesso a fontes de dados não SQL e não fornece uma arquitectura para a construção de novos componentes de informação. Contudo, o factor multiplataforma confere-lhe um estatuto invejável.

2.4.1. O modelo de objectos JDBC e o correspondente ADO

Do ponte de vista do programador, o JDBC define quatro objectos [Hamil96A]. A Tabela II-2 mostra como é que estes quatro objectos JDBC se relacionam com os objectos equivalentes ADO.

A documentação do JDBC subdivide os objectos em interfaces, tipos e classes. Contudo para o bem da comparação com o OLE DB e o ADO, considera-se que os interfaces, tipos e classes do JDBC são equivalentes aos objectos do ADO.

Objecto JDBC	**Equivalente ADO**
DriverManager	Connection
Connection	Connection
Statement	Command
PreparedStatement	Prepared Command
CallableStatement	Command
ResultSet	Recordset

Tabela II-2 – Modelo de objectos do JDBC e os equivalentes no ADO

A funcionalidade destes objectos JDBC pode ser descrita do seguinte modo:

- o objecto *DriverManager* carrega os *drivers* e fornece o suporte para a criação de novas ligações às bases de dados (similar ao objecto *Connection* do ADO);
- o objecto *Connection* define a conexão para uma determinada base de dados (similar ao Objecto *Connection* do ADO);

- o objecto *Statement* atribui e executa uma expressão SQL (este objecto é equivalente ao objecto *Command* do ADO);
  - o sub-tipo *PreparedStatement* é usado quando se quer repetir várias vezes a mesma expressão SQL (é equivalente ao objecto *Command* e a propriedade *Prepared* do ADO);
  - o sub-tipo *CallableStatement* permite a chamada de procedimentos remotos (não existe nada equivalente no ADO; os procedimentos remotos são passados como expressões através do objecto *Command*, tal como as expressões SQL);
- o objecto *ResultSet* representa o resultado de um pedido, retornando apenas uma linha (o objecto *Recordset*, no ADO, tem a mesma função, mas retorna uma ou mais linhas).

2.4.2. Modelo de Programação: JDBC versus ADO

Os modelos de programação do JDBC e do ADO são comparados na Tabela II-3 e descritos de seguida.

Capacidades	JDBC	ADO
Nível de Programação	Baixo	Alto
Tipos de Dados	SQL	Todos
Linguagem	Independente	Independente
Escrita	Estática	Tipo dinâmica e estática
Proprietário dos dados	Cliente	Partilhado

Tabela II-3 – Modelo de programação: JDBC versus ADO

- Nível de Programação – o JDBC pode ser visto como uma API de baixo nível que suporta funcionalidades de SQL básicas. Os autores do JDBC esperam que uma API de alto nível seja definida e possivelmente implementada por cima do JDBC [Hamil96B];
- Tipos de Dados – o JDBC foi desenvolvido para ser uma API para dados SQL exclusivamente. Para ser compatível com o JDBC, e para passar nos teste de compatibilidade do JDBC chamados *JDBC COMPLIANT*, o *driver* tem que

suportar pelo menos o ANSI SQL-92 [Hamil96B]. O SQL é usado para definir o resultado ou para modificar dados através dos comandos *UPDATE, INSERT* ou *DELETE;*

- Linguagem - o ponto mais positivo do JDBC é que este é a única API nativa para aceder a bases de dados SQL com o Java. Contudo, o lançamento do ADO para WFC altera esta situação (em ambientes *Windows*);
- Escrita – o JDBC é muito rígido. Existem métodos de *setxxx* e *getxxx* separados para cada tipo de parâmetro para atribuição de um dado valor e o mesmo se passa quando se pretende obter dados num determinado resultado (*resulset*). Por exemplo, quando se pretende ir buscar a uma coluna um valor inteiro usa-se o método *getint()*; se for um valor longo usa-se o *getlong*(). Existem duas variantes para cada método *getxxx:* uma vai buscar de acordo com o índice e a outra de acordo com o nome da coluna. Por fim, existem 34 métodos diferentes para ir buscar informação a uma determinada coluna de um resultado. Em comparação, o ADO e o OLE DB suportam variáveis do tipo universal que se transformam no tipo de dados correspondente quando lhes é atribuído um determinado conteúdo;
- Proprietário dos dados – o modelo do JDBC, tal como o modelo ODBC, coloca os dados sempre do lado do cliente, isto é, os clientes extraem os dados para a memória local. A versão 2.0 do JDBC permite misturar vários resultados, posicionamento relativo e absoluto e vários mecanismos para actualizar os dados na base de dados. Ao contrário do JDBC, o ADO suporta vários tipos de cursores e diferentes tipos de bloqueio de informação. Algumas destas características dependem da implementação do *driver* OLE DB. Outra característica importante do ADO é permitir que os dados possam ser extraídos assincronamente e que a conexão seja desligada quando já não é necessária. Com um resultado na própria máquina é possível fazer um conjunto de operações como se a conexão estivesse aberta e por fim volta-se a repor as actualizações na fonte de dados.

2.4.3. Funcionalidades: JDBC versus ADO

A Tabela II-4 compara as principais funcionalidades do JDBC e do ADO.

Funcionalidade	JDBC	ADO
	Data manipulation language **(DML)**	
Abertura de *Resultset* ou *Recordset*	Via SQL	Expressões SQL SELECT; passar o nome da tabela ou do objecto através de um procedimento remoto
Criação de tabelas	Baseado em SQL	*Extensões Data Definition Language* (DDL)
Administração da base de dados	N/A	Extensões DLL
	Extracção de dados	
Capacidades de Navegação	Sim	Sim
Actualizar / Inserir / Apagar	Só SQL; a versão 2.0 permite actualizações através da utilização de métodos, suporta e procedimentos remotos	Através de procedimentos remotos ou SQL, através de comandos específicos do fornecedor do *driver* ou através de métodos do objecto *Recordset* tais como *addNew*, *Updade* ou *Delete*
Obtenção de linhas	Uma única linha	Múltiplas linhas
Obtenção de colunas	Uma única coluna	Múltiplas colunas
Sincronização com a base de dados	Não	Sim
Notificações	Não	Sim
Navegação indexada	Não	Sim. O objecto *Recordset* do ADO tem uma propriedade onde se pode explicitar qual o índice que se quer usar; possui também o método *seek* para fazer posicionamentos dentro do objecto
Tratamento de nulos	Método extra	VARIANT_NULL
Forma dos resultados	Tabelar	Além de tabelar, permite *recordsets* hierárquicos.
	Transacções	
Níveis de isolamento	Todos	Todos
Suporte *auto-commit*	Sim	Sim
Transacções embutidas	Não	Sim
Transacções distribuídas	Sim	Sim
	Extensibilidade	
Propriedades	Baseadas em métodos	Extensões
Navegação dos objectos	Não	Sim
Modelo de erros	Lançados	Objecto *Error*

Tabela II-4 – Funcionalidades: JDBC versus ADO

A descrição seguinte resume os resultados da comparação das funcionalidades entre as duas tecnologias:

- *Data manipulation language* – no JDBC, tal como no ODBC, os resultados são obtidos através de expressões SQL ou de procedimentos remotos, não sendo possível usar um método, como por exemplo *open*, como no ADO. Para a criação ou alteração de tabelas o SQL DDL é a única possibilidade, não sendo possivel a administração das bases de dados no JDBC ou no ODBC, isto porque não existem regras no SQL para criação de bases de dados;
- Extracção de dados – o JDBC 2.0 permite navegar nos resultados através de posições absolutas ou relativas e oferece mecanismos alternativos ao SQL para fazer actualizações na base de dados. Para fazer actualizações através do ADO basta invocar os métodos do objecto *Recordset*. Através do JDBC é possível fazer actualizações baseadas em cursores do tipo *dynamic* ou *keyset*. O JDBC possibilita ter os cursores *static*, *dynamic* ou *static* (além destes cursores o ADO ainda suporta o cursor *forward-only*). No JDBC a informação dos *resultset* é extraída uma linha de cada vez. Esta limitação é uma grande desvantagem em relação ao ADO já que este consegue extrair uma linha de cada vez ou múltiplas linhas de uma só vez. Um conceito que o JDBC não possui é o de voltar a extrair uma linha que tinha sido extraída previamente. Sem fazer um bloqueio a um registo quando se está a ler, o utilizador não consegue determinar se esse mesmo registo já foi alterado. O ADO implementa um sofisticado mecanismo de sincronismo: os utilizadores podem especificar as operações de sincronização através de procedimentos remotos que podem ser invocados para operações de sincronização de *recordsets*. O ADO também suporta sincronismo automático nas operações de actualização;
- Transacções – o JDBC possui uma capacidade transaccional bastante simples nas expressões SQL, possuindo suporte para vários níveis de isolamento e a capacidade de *auto-commit*. Na presente versão, JDBC 2.0, já existe o suporte de transacções distribuídas. O ADO suporta ainda transacções umas dentro das outras e, além desta funcionalidade, este mecanismo de acesso disponibiliza ainda métodos e propriedades específicas de cada fonte de informação. As propriedades comuns são disponibilizadas nos objectos *Connection*, *Command*, *Recordset* e

Field e, para suportar propriedades específicas de cada fonte de informação, o ADO fornece uma colecção onde a fonte de informação pode disponibilizar todas as suas propriedades específicas;

- Extensibilidade – o tratamentos de erros no JDBC é similar ao do ADO/WFC, porque ambos os tratam como extensões. No entanto o ADO fornece os erros numa colecção de objectos *Error* que pode ser consultada para ter mais informação sobre o erro ou quando ocorre mais do que um erro de uma única vez. Esta situação é particularmente interessante porque podem existir erros que sejam provocados por erros anteriores ou não.

2.4.4. Conclusão

O JDBC continua a ser um interface muito limitado que tem como objectivo principal a transferência de dados de uma base de dados SQL para um cliente Java. A sua limitação advém da funcionalidade que está disponível e do nível de detalhe necessário para implementar uma aplicação baseada em JDBC.

O ADO/WFC, por outro lado, oferece uma plataforma mais madura para construção de aplicações complexas orientadas para acesso à informação. Com a arquitectura de componentes do OLE DB, o ADO/WFC pode ser usado de uma forma dinâmica, para aceder a dados de diferentes fontes de informação, incluindo bases de dados relacionais e fontes de dados não SQL, tais como dados guardados sob a forma de e-mail. O ADO/WFC oferece também um modelo de programação mais versátil, respeitando a navegação e manipulação de dados.

III. ACTIVEX DATA OBJECTS (ADO)

3.1. Introdução

O ADO é uma *application programming interface* (API) que fornece um acesso a dados de uma forma consistente, facilidade de utilização, com elevado desempenho, pouco gasto de memória de cabeçalhos, e que responde a uma grande variedade de necessidades das equipas de desenvolvimento, nomeadamente a criação de aplicações clientes para acessos a bases de dados, de objectos de processamento ou de objectos lógicos que são usados pelas aplicações, por ferramentas, por linguagens de programação ou por *browsers* WWW. O ADO foi concebido para poder ser o único interface necessário para acesso a dados, desde soluções simples até soluções bastante complexas, tais como soluções de vários níveis cliente/servidor ou soluções baseadas na Web para manipulação de informação.

O ADO fornece um interface simples para o OLE DB, sendo este o responsável pelo acesso à informação. O ADO foi implementado para minimizar a utilização de recursos de rede de comunicação em cenários chave e minimizar também o número de camadas entre as aplicações cliente e a fonte de informação, sempre com a preocupação de ser um interface "leve" e com elevado desempenho em qualquer situação. Este interface é simples de usar porque usa uma metáfora conhecida, o automatismo COM, disponível nas principais ferramentas *Rapid Application Development* (RAD), ferramentas para bases de dados e para as linguagens de programação disponíveis no mercado.

As secções seguintes apresentam os diferentes aspectos do ADO. A primeira parte refere como manipular dados, a segunda como manipular a estrutura da fonte de informação e a terceira como consultar dados via OLAP [Brus99].

3.2. Modelo Genérico

3.2.1. O Modelo de Objectos

O ADO possui um conjunto de objectos que serve de interface com a fonte de informação. Embora mais simples do que outros métodos, como já foi referido, possui as características necessárias para um bom funcionamento. Assim, para se ter uma noção mais precisa do modelo bem como do seu funcionamento passam a ser descritos os objectos constituintes do modelo, as suas colecções (*collection*), alguns métodos, algumas propriedades e os eventos mais importantes.

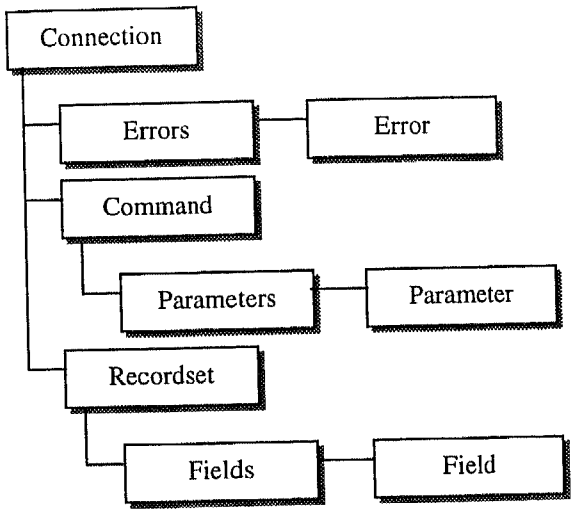

Figura III-1 - Modelo de Objectos do ADO

Cada objecto *Connection*, *Command*, *Recordset*, e *Field* contém uma colecção de propriedades.

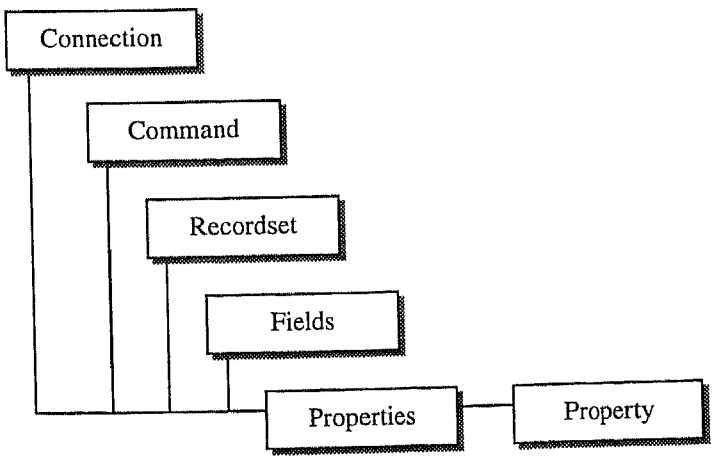

Figura III-2 - Colecções dos Objectos do ADO

3.2.2. Objecto Command

O objecto *Command* define um determinado comando para ser executado na fonte de informação.

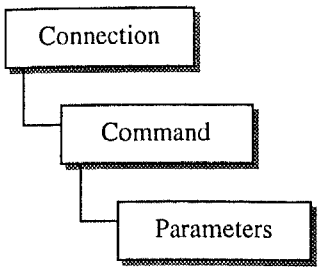

Figura III-3 - Objecto *Command*

Usa-se este objecto quando se pretende executar uma pergunta (*query*) e retornar os registos para o utilizador (retornam no objecto *Recordset*), para executar uma operação pesada sobre a fonte de informação ou para manipular a estrutura da fonte de dados. Dependendo das funcionalidades da fonte de informação, algumas colecções, métodos ou propriedades podem gerar erros quando invocados.

Com as colecções, métodos e propriedades deste objecto, é possível:

- Definir o texto de um comando (por exemplo um comando SQL), com a propriedade *CommandText;*
- Definir perguntas parametrizadas ou argumentos dos procedimentos remotos na base de dados, com o objecto *Parameter* ou com a colecção *Parameteres*;
- Executar um determinado comando e retornar um objecto do tipo *Recordset* através do método *Execute;*
- Especificar, por questões de desempenho, o tipo de comando, com a propriedade *CommandType*;
- Indagar se houve ou não uma execução prévia deste comando, com a propriedade *Prerared*;
- Indicar o número máximos de segundos de espera para um determinado comando ser executado, através da propriedade *CommandTimeOut*;

- Associar uma determinada conexão ao objecto *Command,* através da propriedade *ActiveConnection*;
- Dar um nome para identificar qual o objecto *Command,* através da propriedade *Name*, para possibilitar a associação desse comando a um método do objecto *Connection*;
- Ligar este objecto a outro do tipo *Recordset* para se obter informação através da propriedade *Source*.

É possível executar uma determinada pergunta sem usar o objecto *Command*, bastando especificar a expressão para o método *Execute* do objecto *Connection* ou usar o método *Open* do objecto *Recordset*. Contudo, o objecto *Command* é necessário quando se pretende preservar a expressão (através da propriedade *CommandText* do objecto *Command*) e reutilizá-lo ou usar parâmetros nas perguntas.

Para criar este objecto sem ter definido anteriormente o objecto *Connection*, basta atribuir uma expressão válida (expressão para fazer a conexão) à propriedade *ActiveConnection*. Neste caso, o objecto *Connection* é criado só que não é atribuído a nenhuma variável. Se for necessário utilizar múltiplos objectos *Command*, é vantajoso criar um objecto do tipo *Connection* e abrir uma única conexão com a fonte de dados.

3.2.3. Objecto Connection

Este objecto representa uma conexão aberta para a fonte de informação

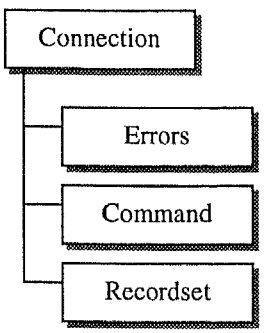

Figura III-4 - Objecto *Connection*

O objecto *Connection* representa uma única ligação com a fonte de informação. No caso de um sistema de base de dados do tipo cliente/servidor, essa ligação pode ser

considerada equivalente a uma conexão entre o cliente e o servidor. Dependendo das funcionalidades suportadas pela fonte de informação, algumas colecções, métodos ou propriedades podem não estar disponíveis no objecto *Connection*.

Com as colecções, métodos e propriedades deste objecto é possível:

- Configurar a conexão antes de a abrir através das propriedades *ConnectionString*, *ConnectionTimeout* e *Mode*;
- Especificar o tipo de cursor através da propriedade *CursorLocation* e eventualmente suportar *batch updates;*
- Especificar a base de dados para uma determinada conexão através da propriedade *DefaultDatabases*;
- Especificar o nível de isolamento das transacções abertas pela conexão através da propriedade *IsolationLevel*;
- Especificar a fonte de informação OLE DB através da propriedade *Provider*;
- Estabelecer e libertar uma conexão com uma determinada fonte de informação através dos métodos *Open* e *Close*;
- Executar um comando na conexão através do método *Execute* e configurá-la através da propriedade *CommandTimeout*;
- Controlar as transacções abertas, incluindo as que podem estar dentro de outras desde que a fonte de informação as suporte, através dos métodos *BeginTrans*, *CommitTrans* e *RollbackTrans* e da propriedade *Attributes*;
- Examinar os erros que foram retornados pela fonte de informação através da colecção *Errors*;
- Ler a versão da implementação do ADO com a propriedade *Version*;
- Obter a informação da estrutura e funcionalidades suportadas pela fonte de informação através do método *OpenSchema*.

Para executar um determinado comando é necessário dar-lhe um nome e criar um elo de ligação entre o objecto *Command* e o objecto *Connection*, através da propriedade *ActiveConnection,* invocar a conexão com o nome do comando como se se tratasse de um método, seguido dos parâmetros e da variável *Recordset* para onde serão retornadas as linhas se for o caso.

Para a execução de um procedimento remoto, basta usar o nome do procedimento como se de um método se tratasse. Os tipos dos parâmetros são tratados pelo ADO como se fosse "a melhor escolha".

3.2.4. Objecto DataControl

Este objecto faz a ligação entre um resultado de uma pergunta (objecto *Recordset*) e um ou mais objectos gráficos (tais como *TextBox*, *ComboBox*, *Grid*) tendo como objectivo fazer aparecer a informação do *ADOR.Recordset* numa página Web automaticamente.

Para um cenário simples, bastam as propriedades *SQL*, *Connect* e *Server*, do objecto *RDS.DataControl*, e este liga-se automaticamente ao objecto *RDSServer.DataFactory*. Todas as propriedades do *RDS.DataControl* são opcionais porque os objectos situados no nível lógico podem substituir as suas funcionalidades.

3.2.5. Objecto DataFactory (RDSServer)

O objecto *RDSServer.DataFactory* foi desenvolvido para estar do lado do servidor com a função de receber os pedidos dos clientes. Numa implementação tipo Internet o objecto reside no servidor de páginas. O objecto *RDSServer.DataFactory* fornece acessos de leitura e escrita às fontes de informação, mas não contém quaisquer mecanismos de validação, ou procedimentos lógicos.

3.2.6. Objecto DataSpace (RDS)

Os serviços remotos usam o objecto *RDS.DataSpace* para fazer um *proxy* dos objectos do servidor para que os componentes do cliente possam comunicar com os objectos do servidor. Este *proxy* facilita o empacotamento, desempacotamento e transporte dos dados.

3.2.7. Objecto Error

O objecto *Error* contém detalhes sobre erros de uma determinada operação na fonte de informação.

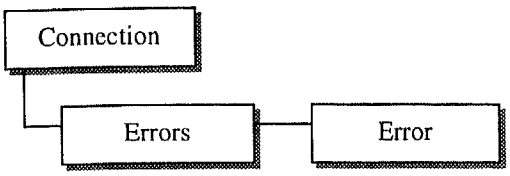

Figura III-5 - Objecto *Error*

Qualquer operação envolvendo objectos ADO pode gerar mais do que um erro, provenientes da fonte de informação. Cada vez que um erro ocorre, é criado um objecto do tipo *Error* que fica disponível na colecção *Errors* acessível no objecto *Connection*. Quando outra operação ADO gera um novo erro, é limpa a colecção *Errors* e são criados novos objectos do tipo *Error*, novamente disponíveis na colecção *Errors*.

É possível ler os objectos *Error* para obter informação detalhada sobre o erro, informação essa que é descrita de seguida:

- Propriedade *Description*, que contém o texto com a explicação do erro;
- Propriedade *Number*, o número que indica a constante do erro;
- Propriedade *Source*, que identifica o objecto que gerou o erro. Isto é particularmente útil para identificar a proveniência do erro quando são gerados vários erros;
- Propriedades *SQLState* e *NativeError* que disponibilizam informação dos erros dada pelas fontes de informação.

Algumas propriedades e métodos retornam avisos que aparecem como objectos do tipo *Error*, na colecção *Errors*, mas que não provocam a paragem da execução do programa.

3.2.8. Objecto Field

O objecto *Field* representa uma coluna da informação de um determinado tipo de dados.

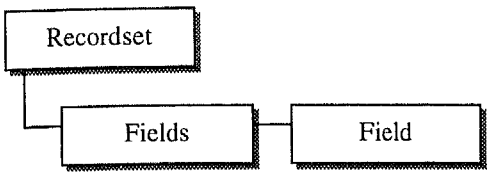

Figura III-6 - Objecto *Field*

O objecto *Recordset* contém uma colecção de objectos chamada *Fields*, objectos esses que são do tipo *Field*. Cada objecto deste tipo corresponde a uma coluna no *Recordset* a propriedade *Value* do objecto *Field* é que contém a informação retornada pela fonte de informação. Dependentemente das funcionalidades da fonte de informação, algumas colecções, métodos ou propriedades do objecto *Field* podem não estar disponíveis.

Com as colecções, métodos ou propriedades do objecto *Field* pode-se:

- Retornar o nome do campo na fonte de informação através da propriedade *Name*;
- Ver e alterar o valor de um determinado campo através da propriedade *Value*;
- Saber as características básicas do campo através das propriedades *Type*, *Precision* e *NumericScale*;
- Determinar o tamanho do campo através da propriedade *DefineSize*;
- Determinar o tamanho real do campo através da propriedade *ActualSize*;
- Determinar quais as funcionalidades suportadas por um determinado campo, através da propriedade *Attributes* e da colecção *Properties;*.
- Manipular grandes quantidades de informação, como por exemplo binários longos ou longas cadeias de caracteres, através dos métodos *AppendChunk* e *GetChunk*;
- Determinar se a fonte de dados suporta *Batch Updates*, sendo possível resolver discrepâncias entre os campos durante actualizações em bloco com as propriedades *OrigianlValue* e *UnderlyingValue*.

3.2.9. Objecto Parameter

Este objecto representa os parâmetros ou argumentos e está associado ao objecto *Command* na parametrização de perguntas ou de procedimentos remotos

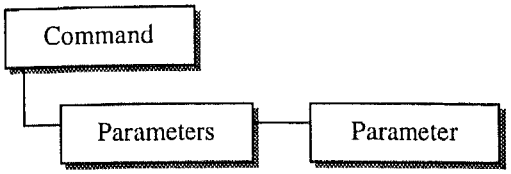

Figura III-7 - Objecto *Parameter*

Muitas fontes de dados suportam comandos parametrizados. Normalmente os comandos são acções definidas uma única vez, mas as variáveis (ou parâmetros) são usadas para alterar os detalhes desses comando. Por exemplo, uma expressão SQL que seja *SELECT* pode usar como parâmetro a definição de critérios de extracção na cláusula *WHERE* ou ainda pode definir o tipo de ordenação com a cláusula *ORDER BY*.

O objecto *Parameter* representa os parâmetros associados às perguntas parametrizáveis, argumentos de entrada e saída, e retorno de valores em procedimentos remotos. Dependentemente das funcionalidades da fonte de informação, algumas colecções, métodos ou propriedades do objecto *Command* podem não estar disponíveis.

Com as colecções, métodos ou propriedades do objecto *Command* pode-se:

- Atribuir e retornar o nome dos parâmetros através da propriedade *Name*;
- Atribuir e retornar o valor dos parâmetros através da propriedade *Value*;
- Atribuir e retornar as características dos parâmetros através das propriedades *Attributes*, *Direction*, *Precision*, *NumericScale*, *Size* e *Type*;
- Passar um binário longo ou uma cadeia de caracteres através do método *AppendChunk*.

Sabendo-se os nomes e propriedades dos parâmetros associados aos procedimentos remotos ou às perguntas parametrizadas que se pretende invocar, é possível usar o

método *CreateParameter* para criar objectos do tipo *Parameter* e preencher as propriedades com a informação apropriada/pretendida, ou usar o método *Append* para acrescentar novos parâmetros à colecção *Parameters*. Esta funcionalidade evita a necessidade de invocar o método *Refresh* na colecção *Parameters* para saber informação sobre os parâmetros da fonte de informação, o que representa uma operação que envolve alguns recursos.

3.2.10. Objecto Property

Este objecto representa dinamicamente as características de um objecto ADO definidas pela fonte de informação.

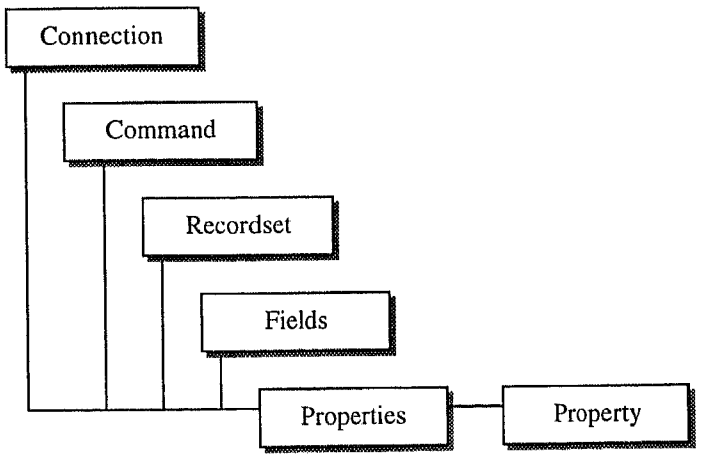

Figura III-8 - Objecto *Property*

Os objectos ADO têm dois tipos de propriedades: intrínsecas e dinâmicas. Estas propriedades não podem ser apagadas. As propriedades intrínsecas são implementadas no ADO e estão disponíveis logo após a criação de qualquer objecto, através da sintaxe *objecto.propriedade*. Estas propriedades não aparecem nos objectos como colecções, mas é possível alterar os valores das propriedades e impossível alterar as suas características. As propriedades dinâmicas, são herdadas da fonte de informação e aparecem como sendo um colecção em cada objecto ADO correspondente. Por exemplo, a fonte de informação pode ter uma propriedade que indique se os seus *Recordset*s suportam transacções e/ou alterações. Estas propriedades encontram-se

sob a forma de colecção em objectos *Property* ligados aos objectos *Recordset*. As propriedades dinâmicas só podem ser acedidas através de colecções.

Qualquer propriedade dinâmica tem 4 propriedades intrínsecas:

- Propriedade *Name* que identifica o seu nome;
- Propriedade *Type* que identifica o seu tipo;
- Propriedade *Value*, que contém o valor;
- Propriedade *Attributes*, um valor longo que identifica as características da propriedade perante a sua fonte de informação;

3.2.11. Objecto RecordSet

Este objecto representa um conjunto de registos provenientes de uma tabela ou de um comando. Em qualquer altura o objecto *Recordset* refere-se sempre a um único registo dentro de um conjunto, sendo este o registo actual.

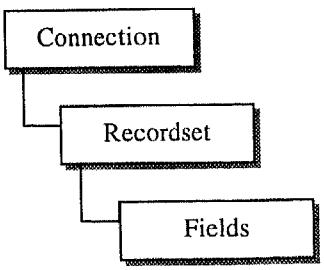

Figura III-9 - Objecto *Recordset*

Este objecto serve para manipular a informação da fonte de dados. Todos os objectos *Recordset* são constituídos por registos (linhas) e campos (colunas). Dependentemente das funcionalidades da fonte de informação, algumas colecções, métodos ou propriedades do objecto *Recordset* podem não estar disponíveis.

Existem quatro tipos de cursores no ADO:

- Cursor *Dynamic* – permite ver novos registos, alterações e registos apagados de outros utilizadores e permite também fazer todos os tipos de movimentos dentro

de um *Recordset*. É possível a utilização de *bookmarks* se a fonte de informação também os permitir;

- Cursor *Keyset* – comporta-se como o mecanismo anterior, exceptuando a possibilidade de ver registos que os outros utilizadores acrescentam ou apagam. As alterações dos outros utilizadores são visíveis;
- Cursor *Static* - fornece uma cópia estática do conjunto de registos que normalmente é usada para efectuar pesquisas e gerar relatórios. As alterações efectuadas por outros utilizadores não são visíveis devido ao facto dos dados serem transportados para a máquina local. Este tipo de cursor só pode ser definido do lado do cliente;
- Cursor *Forward-only* – comporta-se como o cursor *Dynamic* mas só permite movimentos num sentido ao longo do *Recordset*, do princípio para o fim, ao contrário dos outros cursores. Isto pode ser bastante útil quando só se pretende fazer uma passagem pelo *Recordset*.

Quando é aberto um *Recordset*, o registo corrente é o primeiro (se existir mais que um) e as propriedades *BOF (begin of file)* e *EOF (end of file)* possuem o valor falso. Se o *Recordset* não contiver nenhum registo então estas propriedades contêm o valor verdadeiro.

É possível usar os métodos *MoveFirst*, *MoveLast*, *MoveNext* e *MovePrevious* bem como o método *Move* e as propriedades *AbsolutePosition*, *AbsolutePage* e *Filter* para fazer posicionamentos dentro do *Recordset*, se a fonte de dados suportar todas estas funcionalidades. Quando é definido o cursor *Forward-only* o único método que pode ser usado é o *MoveNext*.

3.3. ActiveX Data Objects Extensions

ActiveX Data Objects Extensions for Data Definition Language and Security (ADOX) é uma extensão dos objectos ADO e do seu modelo. O ADOX inclui objectos para criação, modificação do esquema ou estruturação da fonte de informação e também para efeitos de segurança nomeadamente para a manutenção de utilizadores e grupos,

e pode atribuir ou retirar permissões a objectos da fonte de informação [MacM99]. Porque é uma aproximação baseada em objectos, é possível manipular as estruturas das fontes de informação apesar das sintaxes nativas serem diferentes [MacM99].

3.3.1. Modelo de Objectos do ADOX

O diagrama seguinte mostra os objectos do modelo e ainda como eles se relacionam no ADOX.

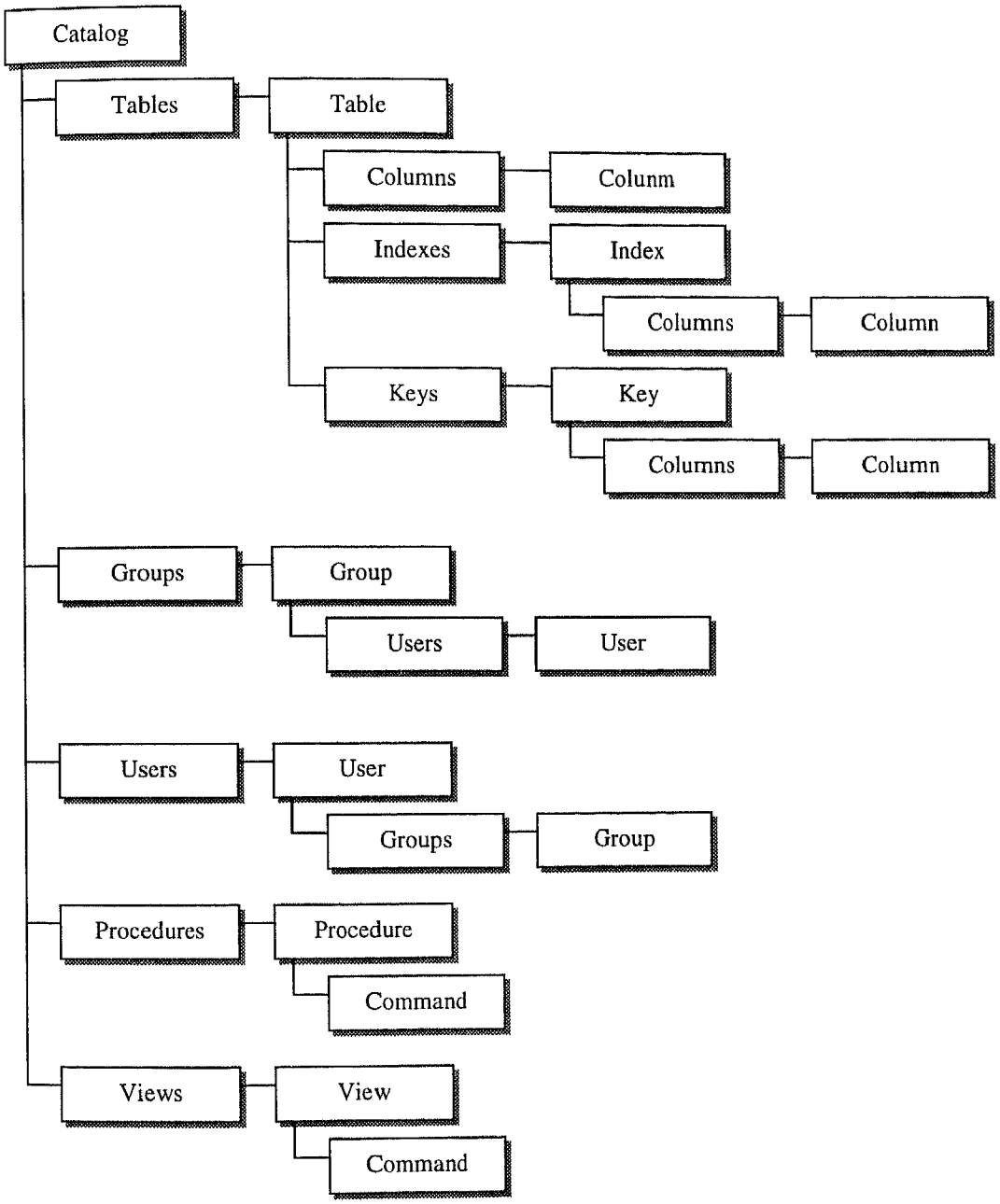

Figura III-10 - Modelo de Objectos ADOX [Micr99A]

Cada objecto do tipo *Table*, *Index* e *Column* possui também uma colecção de propriedades normalizadas do ADO.

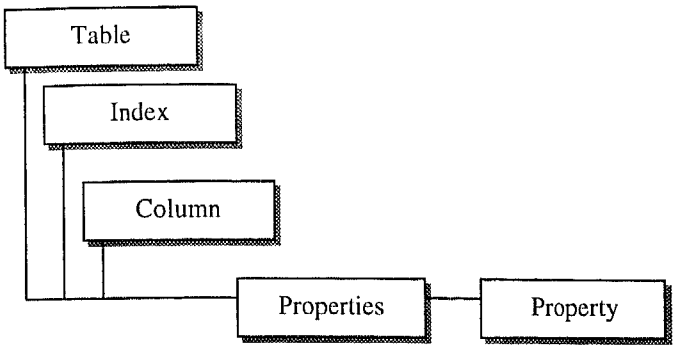

Figura III-11 - Colecção de objectos do ADOX [Micr99A]

3.3.2. Objecto Catalog

Este objecto contém as colecções (tabelas, grupos, utilizadores, procedimentos e vistas) que descrevem a estrutura da fonte de informação.

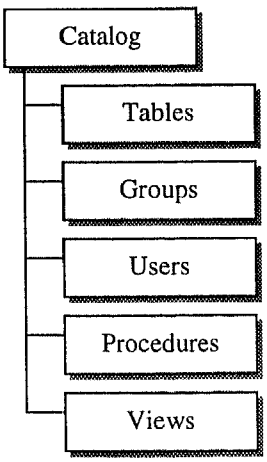

Figura III-12 - Objecto *Catalog*

É possível modificar o objecto *Catalog*, bastando para isso adicionar, eliminar ou alterar os objectos. Algumas fontes de informação não suportam todos os objectos do *Catalog* ou só permitem visualizar a informação da estrutura, mas não alterá-la.
Com as propriedades e métodos do objecto *Catalog* é possível:

- Abrir o objecto *Catalog* através da atribuição do objecto *Connection* do ADO ou de uma expressão de conexão à propriedade *ActiveConnection*;
- Criar um novo *Catalog* através do método *Create*;

- Determinar os utilizadores que criaram os objectos dentro da fonte de informação com os métodos *GetObjectOwner* e *SetObjectOwner*.

3.3.3. Objecto Column

Este objecto representa uma coluna de uma tabela, um índice ou uma chave.

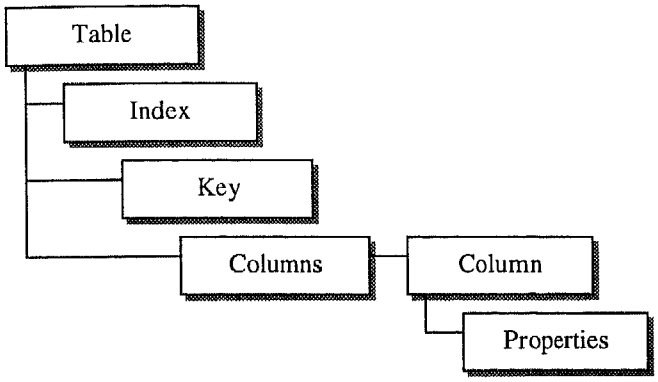

Figura III-13 - Objecto *Column*

Com as propriedades e colecções do objecto *Column* é possível:

- Identificar as colunas através da propriedade *Name*;
- Especificar o tipo de dados da coluna através da propriedade *Type*;
- Determinar se a coluna é de comprimento fixo ou se pode ser nula, através da propriedade *Attributes*;
- Especificar o tamanho máximo da coluna com a propriedade *DefinedSize*;
- Para os valores numéricos, especificar a escala através da propriedade *NumericScale*;
- Para os valores numéricos, especificar a precisão máxima através da propriedade *Precision*;
- Saber qual o objecto *Catalog* que possui uma determinada coluna através da propriedade *ParentCatalog*;
- Para as colunas chave, especificar o nome da coluna relacionada e da tabela relacionada através da propriedade *RelatedColumn*;

- Para colunas de índices, especificar com a propriedade *SortOrder* se a ordenação é crescente ou decrescente;
- Aceder às propriedades especificas através da colecção *Properties*.

3.3.4. Objecto Group

Representa um grupo de utilizadores que têm permissões de acesso a uma fonte de dados.

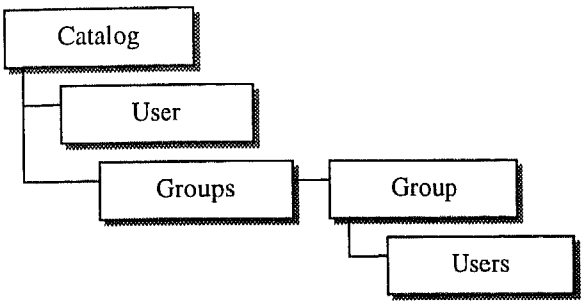

Figura III-14 – Objecto *Group*

A colecção *Groups* de um determinado *Catalog* representa todas as contas dos grupos desse *Catalog*. A colecção *Groups* para um determinado *User* representa apenas os grupos a que o utilizador pertence.

Com as propriedades, colecções e métodos do objecto *Groups* é possível:

- Identificar o grupo através da propriedade *Name*;
- Determinar ou alterar as permissões do grupo, nomeadamente de leitura, escrita ou de eliminar, através dos métodos *GetPermissions* e *SetPermissions*;
- Saber quais os utilizadores que pertencem a esse grupo, através da colecção *Users*.

3.3.5. Objecto Index

Representa os índices de uma determinada tabela.

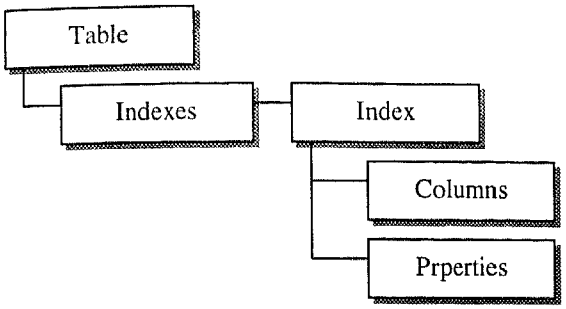

Figura III-15 - Objecto *Index*

Com as propriedades e colecções do objecto *Index*, é possível:

- Identificar o índice através da propriedade *Name*;
- Aceder às colunas da base de dados que contém índices através da colecção *Columns*;
- Identificar se os índices são únicos através da propriedade *Unique*;
- Identificar se os índices são de chave primária através da propriedade *PrimaryKey*;
- Identificar se os registos que contêm valores nulos nos campos que são índices podem ser nulos através da propriedade *IndexNulls*;
- Saber a que objecto *Catalog* pertence um determinado índice através da propriedade *ParentCatalog*;
- Aceder a propriedades específicas das diferentes fontes de informação, através da colecção *Properties*.

3.3.6. Objecto Key

Representa um campo que pode ser uma chave primária, uma chave externa ou uma chave única de uma determinada tabela de uma base de dados.

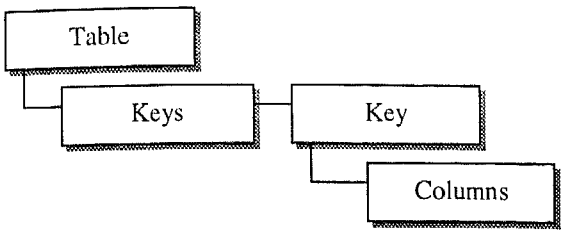

Figura III-16 – Objecto *Key*

Com as propriedades e colecções deste objecto é possível:

- Identificar a chave através da propriedade *Name*;
- Identificar o tipo de chave através da propriedade *Type*;
- Aceder às colunas da base de dados que são chaves, através da colecção *Columns*;
- Identificar o nome da tabela relacionada através da propriedade *RelatedTable*;
- Identificar uma acção quando se executam operações de inserção ou eliminação de uma determinada chave através da propriedades *DeleteRule* e *UpdateRule*.

3.3.7. Objecto Procedure

Representa os procedimentos remotos e quando usado com o objecto do ADO *Command*, o objecto *Procedure* pode servir para editar, apagar e modificar procedimentos remotos. O objecto *Procedure* permite também a criação de procedimentos remotos desconhecendo a sintaxe de cada fonte de informação.

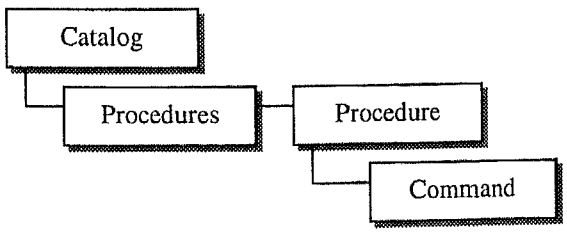

Figura III-17 - Objecto Procedure

Com as propriedades deste objecto é possível:

- Identificar o nome do procedimento remoto através da propriedade *Name*.
- Especificar o objecto *Command* do ADO que pode ser usado para criar ou executar um procedimento remoto através da propriedade *Command*.
- Saber informação sobre a criação e modificação de procedimentos remotos através das propriedades *DateCreated* e *DateModified*.

3.3.8. Objecto Table

Representa as tabelas da base de dados incluindo as colunas, chaves e índices.

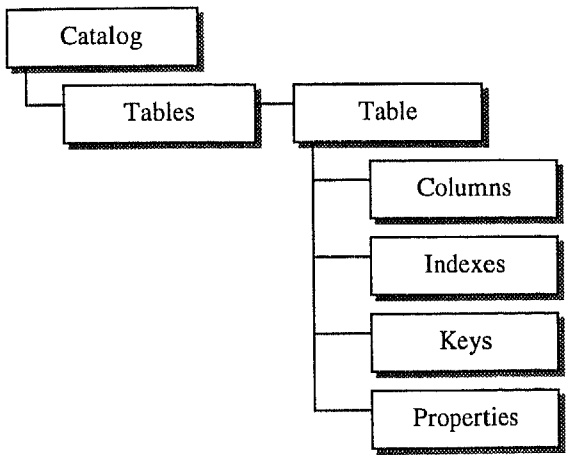

Figura III-18 – Objecto *Table*

Com as propriedades e colecções de um objecto *Table* é possível:

- Identificar a tabela através da propriedade *Name*;
- Determinar o tipo de tabela através da propriedade *Type*;
- Aceder às colunas de uma tabela através da colecção *Columns*;
- Aceder aos índices de uma tabela através da colecção *Indexes*;
- Aceder às chaves de uma tabela através da colecção *Keys*;
- Especificar a que objecto do tipo *Catalog* pertence a tabela através da propriedade *ParentCatalog*;

- Saber informação da data de criação e alteração da estrutura da tabela através das propriedades *DateCreate* e *DataModified.*
- Aceder a especificações das tabelas para as diferentes fonte de informação através das colecções de propriedades.

3.3.9. Objecto User

Representa a identificação e privilégios de um determinado utilizador de uma base de dados.

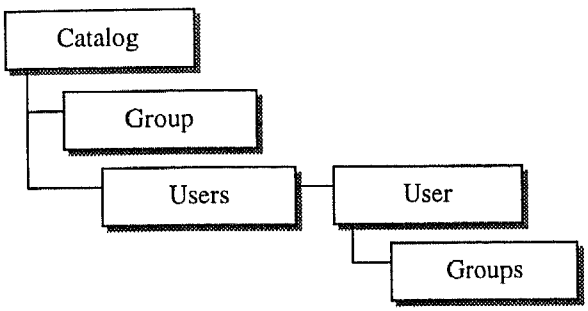

Figura III-19 – Objecto *User*

A colecção de utilizadores de uma determinado *Catalog* representa todos os utilizadores desse mesmo *Catalog*. A colecção de utilizadores para um determinado *Group* representa unicamente os utilizadores desse *Group*.

Com as propriedades, colecções e métodos do objecto *User*, é possível:

- Identificar o utilizador através da propriedade *Name*;
- Trocar a palavra-chave através do método *ChangePassword*;
- Determinar/alterar as permissões de leitura, escrita ou eliminação de um utilizador através dos métodos *GetPermissions* e *SetPermissions*;
- Aceder aos grupos a que pertence o utilizador através da colecção *Groups.*

3.3.10. Objecto View

Representa um conjunto de registos filtrados ou uma tabela virtual e, quando usado em conjunto com o objecto *Command* do ADO, este objecto permite criar, modificar e apagar vistas.

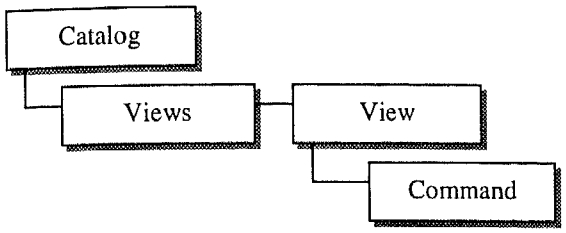

Figura III-20 - Objecto *View*

Uma vista é uma tabela virtual, criada a partir de tabelas ou vistas da base de dados. Este objecto permite criar novas vistas desconhecendo-se a sintaxe da fonte de informação que está a ser usada.

Com as propriedades do objecto *View*, é possível:

- Identificar a vista através da propriedade *Name*;
- Especificar o objecto *Command* do ADO que vai ser usado para criar, alterar ou apagar vistas através da propriedade *Command*;
- Saber informação sobre a data de criação e alteração da vista através das propriedades *DateCreate* e *DateModified*.

3.4. ADO Multidimensional

O ADO MD fornece acesso a fontes de informação multidimensional, através das linguagens Microsoft, Visual Basic, Microsoft Visual C++, and Microsoft Visual J++ O ADO MD é uma extensão do ADO para incluir objectos específicos para informação multidimensional tais como os objectos *CubeDef* ou *Cellset*. Com o ADO MD é possível navegar numa estrutura multidimensional de informação e retirar dados [Brus99].

Tal como o ADO, o ADO MD usa o OLE DB para obter acesso às fontes de informação. Para trabalhar com o ADO MD, a fonte de informação tem que ser uma

multidimensional data provider (MDP) definida no OLE DB como uma especificação OLAP. Os MDPs apresentam dados numa perspectiva multidimensional, ao contrário dos *tabular data providers* (TDPs) que apresentam os dados com vistas do tipo tabela.

3.4.1. O Modelo de ADO MD

O diagrama seguinte mostra os objectos do modelo e ainda como eles se relacionam no ADO MB. Estes objectos são: *Axis, Catalog, Cell, Cellset, CubeDef, Dimension, Hierarchy, Level, Member* e *Posiction.*

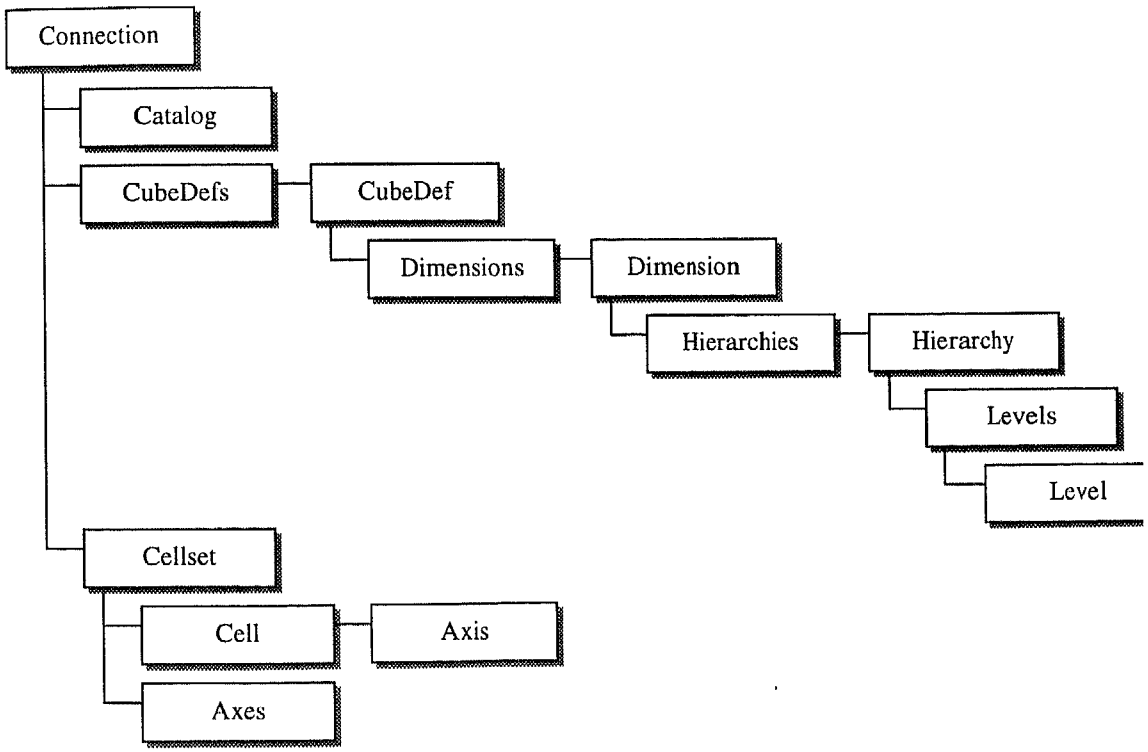

Figura III-21 - Modelo de Objectos do ADO MD [Micr99B]

Os objectos *Axis* e *Cell* contêm cada um uma colecção de *Positions.*

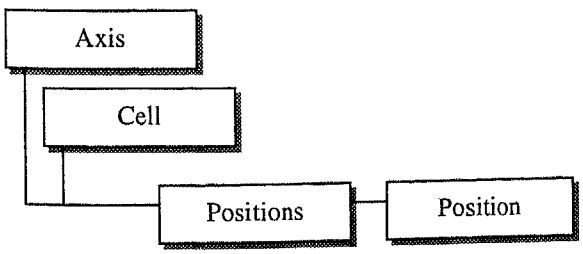

Figura III-22 – Colecções dos Objectos *Axis* e *Cell*

Os objectos *Level* e *Position* têm uma colecção de membros.

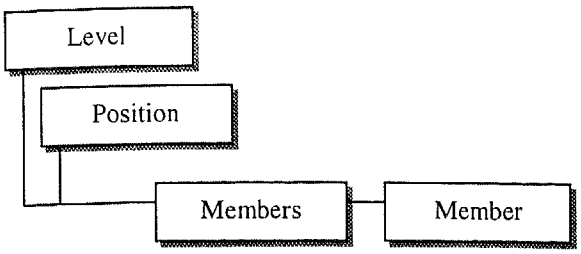

Figura III-23 – Colecções dos Objectos *Level* e *Position*

Os objectos *Axis*, *Cell*, *Callset*, *CubeDef*, *Dimension*, *Hierarchy*, *Level* e *Member* ainda têm a colecção *Properties* do ADO.

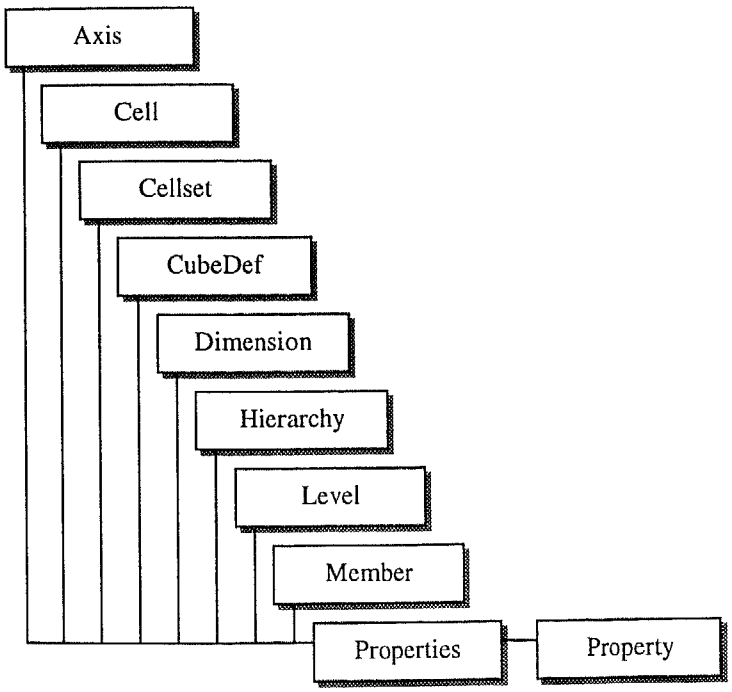

Figura III-24 – Colecções do Modelo de Objectos ADO MD

3.4.2. Objecto Axis

Representa uma posição ou filtro de um eixo de um objecto do tipo *cellset* e contém os membros de uma ou mais dimensões.

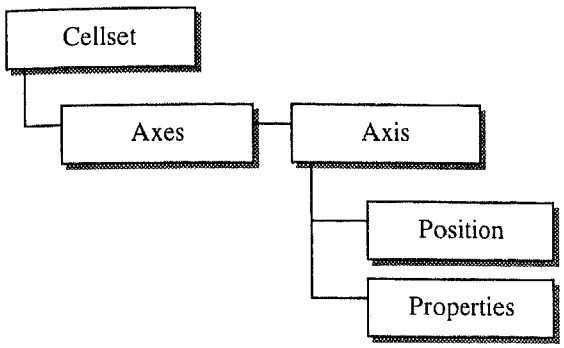

Figura III-25 – Objecto Axis

Um objecto *Axis* pode estar contido numa colecção de *Axes* ou ser retornado pela propriedade *FilterAxis* do objecto *Cellset.*

Com a colecção e propriedades do objecto *Axis*, é possível:

- Identificar os objectos *Axis* por um determinado nome, com a propriedade *Name;*
- Fazer iterações por posição do objecto *Axis* usando a colecção *Positions*;
- Obter o número de dimensões por eixo com a propriedade *DimensionCount;*
- Obter propriedades específicas das fontes de informação através da colecção *Properties* do ADO.

3.4.3. Objecto Catalog

Contém informação da estrutura multidimensional (tal como cubos, dimensões subjacentes, hierarquias, níveis e membros) de uma fonte de informação multidimensional (MDP - *multidimensional data provider*)

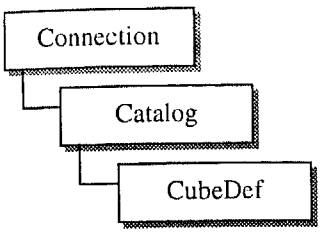

Figura III-26 – Objecto *Catalog*

Com as propriedades e colecções do objecto *Catalog*, é possível:

- Abrir um objecto do tipo *Catalog*, atribuindo à propriedade *ActiveConnection* um objecto *Connection* do ADO, ou simplesmente atribuindo uma expressão válida de uma conexão à referida propriedade;
- Identificar o *Catalog* através da propriedade *Name*;
- Navegar pelos cubos de um *Catalog* através da colecção *CubeDefs*.

3.4.4. Objecto Cell

Apresenta a informação das diferentes intersecções dos eixos contida num objecto *Cellset*.

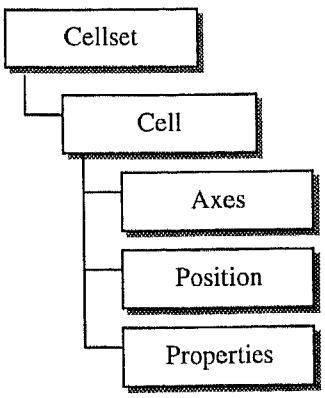

Figura III-27 – Objecto *Cell*

Um objecto *Cell* é retornado pelo método *Item* do objecto *Cellset* e, com as colecções e propriedades do objecto *Cell* é possível:

- Identificar a informação que está no objecto através da propriedade *Value*;
- Identificar uma expressão com o valor formatado (valor que está na propriedade *Value*), com a formatação definida na propriedade *FormattedValue*;
- Identificar a posição ordinal de um objecto *Cell* dentro do objecto *Cellset*;
- Identificar a posição de um objecto *Cell* dentro do objecto *CubeDef* (definido secção 3.4.6) através da colecção *Positions*;
- Identificar outra informação sobre o objecto *Cell* através da colecção *Properties*.

3.4.5. Objecto Cellset

Representa o resultado de uma pergunta multidimensional e é uma colecção de objectos *Cell* provenientes de cubos ou de outros objectos do tipo *Cellset*.

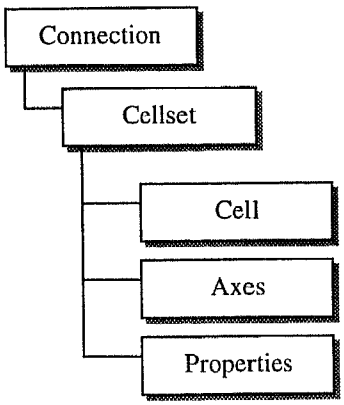

Figura III-28 – Objecto *Cellset*

A informação num objecto *Cellset* é extraída directamente como se se tratasse de matriz, sendo possível navegar directamente para um objecto *Member* para obter informação sobre esse objecto.

Não existe a noção do objecto *Cell* corrente num objecto *Cellset* e o método *Item* devolve um objecto *Cell* específico de um *Cellset*. Os argumentos do método *Item* determinam que objectos *Cell* são extraídos, sendo possível extrair um objecto pela sua posição ordinal, pela posição de um eixo, etc.

Com as colecções, métodos e propriedades de um objecto *Cellset*, é possível:

- Associar uma conexão aberta a um objecto *Cellset*, através da propriedade *ActiveConnection;*
- Executar e extrair os resultados de uma pergunta multidimensional através do método *Open*;
- Visualizar a informação de um objecto *Cell* através do método *Item*;
- Retirar informação sobre as dimensões do filtro, usadas para extrair a informação de um *Cellset,* através da propriedade *FilterAxis*;
- Identificar ou especificar uma pergunta para definir um *CellSet* através da propriedade *Source*;
- Identificar o estado de um objecto *Cellset* (aberto, fechado, em execução, ou em estado de conexão) através da propriedade *State*;
- Fechar um *Cellset* com o método *Close*;
- Identificar outra informação sobre o objecto *Cellset* através da colecção *Properties.*

3.4.6. Objecto CubeDef

Representa a estrutura multidimensional de um cubo contendo o conjunto das relações das dimensões.

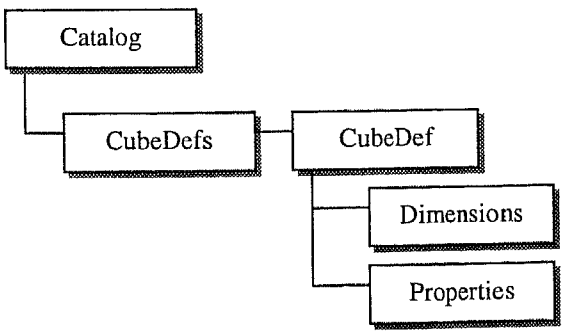

Figura III-29 – Objecto *CubeDef*

Com as colecções e propriedades do objecto *CubeDef* é possível:

- Identificar um *CubeDef,* através da propriedade *Name*;

- Identificar a expressão com a descrição do cubo, através da propriedade *Description*;
- Identificar as dimensões do cubo, através da colecção *Dimensions*;
- Identificar outra informação sobre o objecto *CubeDef* através da colecção *Properties.*

3.4.7. Objecto Dimension

Representa uma dimensão de cubo multidimensional e pode conter um ou mais membros hierárquicos.

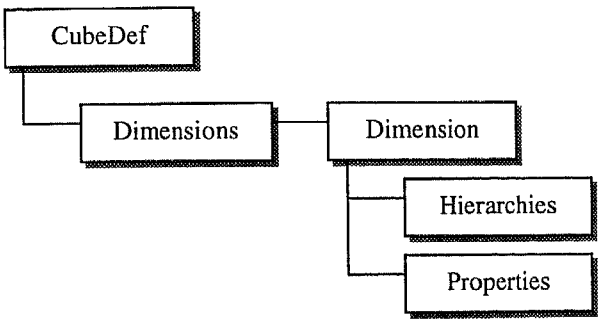

Figura III-30 – Objecto *Dimension*

Com as colecções e propriedades do objecto *Dimension* é possível:

- Identificar a dimensão através das propriedades *Name* e *UniqueName*;
- Identificar uma expressão que descreve a dimensão, através da propriedade *Description*;
- Identificar a colecção *Hierarchies*, que contém os objectos *Hierarchy*;
- Usar a colecção do ADO *Properties*, para obter informação adicional sobre o objecto *Dimension.*

3.4.8. Objecto Hierarchy

Representa uma maneira de agregar os membros de uma dimensão, podendo uma dimensão fazer parte de uma ou mais hierarquias.

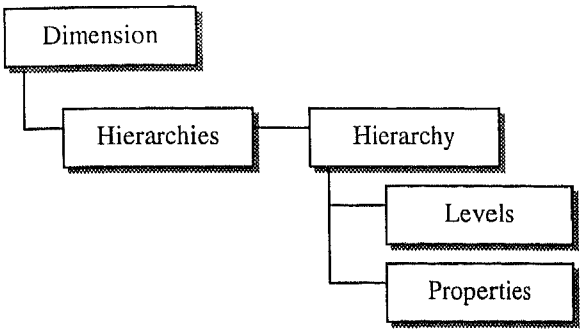

Figura III-31 – Objecto *Hierarchy*

Com as colecções e propriedades do objecto é possível:

- Identificar a hierarquia, através das propriedades *Name* e *UniqueName*;
- Identificar uma expressão que descreve a dimensão, através da propriedade *Description*;
- Identificar a colecção *Levels*, que contém os objectos *Level*;
- Usar a colecção do ADO *Properties*, para obter informação adicional sobre o objecto *Hierarchy*.

3.4.9. Objecto Level

Contém um conjunto de membros, tendo cada um o mesmo grau dentro de uma hierarquia.

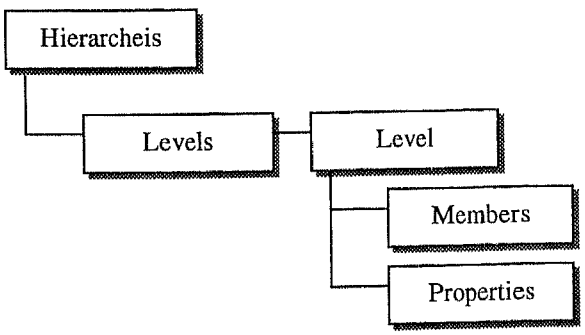

Figura III-32 - Objecto *Level*

Com as colecções e propriedades do objecto *Level* é possível:

- Identificar o *Level,* através das propriedades *Name* e *UniqueName*;
- Identificar uma expressão do tipo título, através da propriedade *Caption*;
- Identificar uma expressão que descreve o nível, através da propriedade *Description*;
- Identificar a colecção *Members*, que contém os objectos *Member*;
- Devolver o número de níveis desde o nó principal, através da propriedade *Depth*;
- Usar a colecção do ADO *Properties*, para obter informação adicional sobre o objecto *Level.*

3.4.10. Objecto Member

Representa um membro de um nível num cubo, um filho de um membro num nível ou uma posição ao longo de um eixo de um *Cellset.*

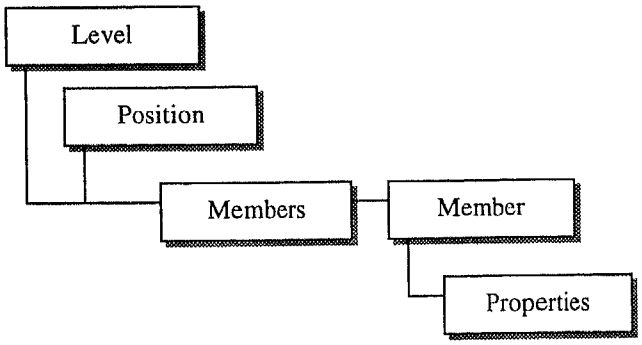

Figura III-33 – Objecto *Member*

As propriedades do objecto dependem do contexto em que é usado. O objecto *Member* de um objecto *Level* de um *Cubedef* tem a propriedade *Children* que retorna os membros do nível mais abaixo na hierarquia a partir desse membro. Um *Member* do objecto *Position* contém sempre a propriedade *Children* vazia. A propriedade *Type* do objecto *Member* também só é aplicável aos objectos *Level.*

Um objecto *Member* associado ao objecto *Position* tem duas propriedades, *DrilledDown* e *ParentSameAsPrev*, que são bastante úteis quando se visualiza um *Cellset*.

Com as colecções e propriedades de objecto *Member* associado ao objecto *Level* é possível:

- Identificar o *Member* através das propriedades *Name* e *UniqueName*;
- Identificar uma expressão do tipo título através da propriedade *Caption*;
- Identificar uma expressão que descreve o membro através da propriedade *Description*;
- Determinar a natureza do membro através da propriedade *Type*;
- Obter informação em relação ao nível do membro através das propriedades *LevelDepth* e *LevelName*;
- Obter as colecções de *Members* relacionadas numa hierarquia através das propriedades *Parent* e *Children*;
- Contar o número de filhos de um *Member* através da propriedade *ChildCount*.

Com as colecções e propriedades de objecto *Member* associado ao objecto *Position* ao longo de um eixo é possível:

- Identificar o *Member,* através das propriedades *Name* e *UniqueName;*
- Identificar uma expressão tipo título, através da propriedade *Caption*;
- Identificar uma expressão que descreve o membro, através da propriedade *Description*;
- Usar a propriedade *DrilledDown* para determinar se existe pelo menos mais um filho no objecto *Axis* depois desse membro;
- Usar a propriedade *ParentSameAsPrev* para determinar se o objecto *Parent* é o mesmo.

3.4.11. Objecto Position

Representa o conjunto de um ou mais objectos do tipo *Member* de diferentes dimensões que se encontram num ponto ao longo de um dos eixos.

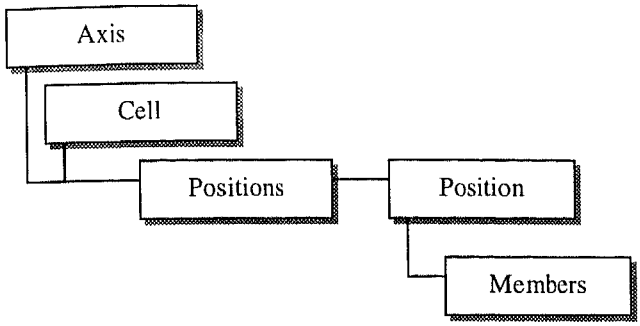

Figura III-34 - Objecto *Position*

Com as propriedades e colecções do objecto *Position* é possível:

- Usar a propriedade *Ordinal* para saber a posição ordinal do objecto *Position* ao longo do eixo;
- Usar a colecção *Member* para conhecer os membros que estão numa posição ao longo do eixo.

3.5. Algumas Características do ADO

O ADO possui muitas características para além das já focadas, entre as quais, é importante salientar: serem relevantes para a compreensão de funcionalidades; serem inovadoras nesta área e poderem reduzir o trabalho para quem as pode usar.

3.5.1. Criação rápida de Objectos do tipo Recordset

O ADO fornece um método rápido para a criação de *Recordsets*, bastando acrescentar um objecto do tipo *Field* à colecção de *Fields* do objecto *Recordset*. Esta funcionalidade proporciona a inserção de informação em qualquer fonte de dados, não sendo necessariamente uma base de dados. É possível também usar o objecto *Recordset* como se se tratasse de um *array* ou de outra forma qualquer de armazenamento de dados.

O objecto *Recordset* pode ser usado para armazenamento de dados tal como foi descrito, ou para armazenar dados que não estão directamente relacionado com uma fonte de dados. Esta característica fornece aos programadores uma nova forma de armazenamento de dados.

3.5.2. *Recordsets* persistentes

Com esta funcionalidade é possível gravar o conteúdo de um *Recordset* num ficheiro. Mais tarde, é possível recriar novamente o objecto original, através dos dados que foram armazenados, em qualquer local, tal como: na máquina local, num servidor de rede ou num sítio Internet.

Através do método *GetString* do objecto *Recordset* os dados são transformados em linhas e colunas e delimitados através de um caracter escolhido pelo programador.

Os dados ao serem convertidos podem ser armazenados sob duas formas: num formato proprietário, *Advanced Data TableGram* (ADTG), ou num formato *Extensible Markup Language* (XML).

Todas as alterações pendentes podem ser gravadas no ficheiro persistente e é possível executar uma pergunta que devolve um *Recordset*, editar os dados, gravar as alterações e mais tarde actualizar a fonte de informação com as alterações realizadas.

Gravar um Recordset:

```
Dim rs as New ADODB.Recordset
rs.Save "c:\nome.adtg", adPersistADTG
```

Abrir um ficheiro gerado pelo Recordset:

```
dim rs as New ADODB.Recordset
rs.Open "c:\nome.adtg", "Provider=MSPersist",,,adCmdFile
```

Abrir um ficheiro através de uma conexão:

```
dim conn as New ADODB.Connection
```

```
dim rs as New ADODB.Recordset
conn.Open "Provider=MSPersist"
set rs = conn.execute("c:\yourFile.adtg")
```

3.5.3. Suporte de índices e das funcionalidades de *Find*, *Short* e *Filter*

Índices nos campos geralmente aumentam significativamente o desempenho do método *Find* e das propriedades *Sort* e *Filter*. É possível criar índices internos para um determinado objecto do tipo *Field*, através da propriedade dinâmica *Optimize*. Esta propriedade só está disponível na colecção *Properties* do objecto *Fields* quando a propriedade *CursorLocation* é igual ao valor *adUseClient*. O índice é interno ao ADO e não pode ser usado com qualquer outro propósito.

A propriedade *Sort* ordena o *Recordset* pelos campos escolhidos, a propriedade *Filter* determina quais as linhas que podem ser acedidas e o método *Find* localiza rapidamente um determinado valor de um campo.

3.5.4. ADO para *Windows Foundation Classes*

É possível usar o ADO para aceder a dados através do Microsoft Visual J++. O ADO para *Windows Foundation Classes* suporta todos os métodos, propriedades e eventos existentes no ADO original. Contudo operações que requerem um VARIANT como parâmetro e, com excelentes resultados de desempenho em linguagens como o *Visual Basic*, não possuem o mesmo rendimento em linguagens tipo Visual J++. Por esta razão o ADO/WMC fornece funções no objecto *Field* para manipular dados nativos do *Java* em vez do tipo VARIANT.

3.5.5. ADO e o modelo de eventos e operações assíncronas

O modelo de eventos do ADO suporta operações síncronas e assíncronas que emitem eventos ou notificações antes de certas operações começarem ou depois de estarem concluídas.

Os eventos que são chamados antes das operações começarem permitem examinar ou modificar parâmetros da operação, cancelar ou permitir que essa operação continue.

Os eventos chamados após as operações terem terminado são bastante importantes agora que o ADO suporta operações assíncronas. Por exemplo uma aplicação que comece uma operação assíncrona de abertura de um *Recordset* é notificada que está concluída através de um evento.

Existem duas famílias de eventos: os *ConnectionEvent,* que informam sobre operações baseadas no objecto *Connection*, e os *RecordsetEvent,* que possuem informação de operações sobre o objecto *Recordset.*

Os eventos do tipo *ConnectionEvent* ocorrem quando se começa, acaba ou quando se cancela uma transacção. Existem também eventos quando começa ou acaba uma conexão.

Os eventos de *RecordsetEvents* ocorrem durante a transição entre os registos de um objecto do tipo *Recordset,* quando se altera um valor de um determinado campo e, genericamente, quando há uma alteração a qualquer nível do objecto *Recordset.*

3.5.6. Extensões do C++ para ADO

Um problema que os programadores em C++ têm quando usam ADO é a conversão do tipo VARIANT retornado pelo ADO para um tipo em C++ e o armazenamento dessa informação numa classe ou numa estrutura. Além de ser incómodo, o facto de se fazer uma conversão de um tipo VARIANT diminui o desempenho do acesso.

O ADO disponibiliza um interface que suporta a extracção de dados directamente para tipos de dados nativos do C++, sem que tenha de passar pelo tipo VARIANT, e disponibiliza também um pré-processador de macros que simplificam o uso de interfaces. O resultado é uma ferramenta flexível e fácil de usar e que possui um elevado desempenho.

3.5.7. Data Shaping

Data Shaping permite a definição das colunas de um *Recordset*, as relações entre as entidades representadas pelas colunas e a maneira do *Recordset* ser preenchido com dados [Sieb99].

Colunas de um *Recordset* podem conter dados, referências para outro *Recordset*, valores que derivaram de um cálculo numa única linha de um *Recordset*, valores que derivaram de uma operação sobre uma coluna de um *Recordset* inteiro ou podem estar simplesmente vazias.

Quando se extrai o valor de uma coluna que contém uma referência para outro *Recordset*, o ADO automaticamente retorna o *Recordset* que representa a referência.

Um *Recordset* que contém outro *Recordset* é chamado um *Recordset* hierárquico. *Recordsets* hierárquicos exibem uma relação do tipo pai/filho, em que o pai é o *Recordset* que contém, e o filho é o *Recordset* contido. Uma referência a um *Recordset* que é uma referência para um subconjunto do filho chama-se um capítulo. Um único pai pode conter mais de um *Recordset* filho.

A sintaxe de comando *Shape* permite fazer um objecto *Recordset* hierárquico de dois modos. O primeiro junta um *Recordset* filho a um *Recordset* pai. O pai e o filho têm que ter pelo menos em comum uma coluna: o valor da coluna do pai é igual ao valor de todas as colunas no *Recordset* filho.

O segundo gera um *Recordset* pai a partir de um *Recordset* filho. Deve haver uma coluna de capítulo no pai que faça referência ao *Recordset* filho. As outras colunas de pai são criadas através de operações de agregação em cima de uma ou mais colunas do *Recordset* filho. É possível ter vários níveis de *Recordsets*.

IV. APLICAÇÕES

4.1. Introdução

Este capítulo é constituído por dois exemplos que procuram demonstrar as potencialidades do ADO como método de acesso a fontes de informação.

Devido às diferentes aplicações que possamos dar ao ADO, com estes dois exemplos pretende-se ilustrar as funcionalidades principais do método; contudo existem outras funcionalidades que não são evidentes ou não estão contempladas.

O primeiro exemplo é bastante concreto: trata-se de um sistema de *Trouble Tickets* que explora algumas das potencialidades do ADO. O sistema é constituído por duas partes: na primeira o ADO é utilizado para fazer o acesso de uma aplicação em Windows entre uma fonte de informação. A segunda parte é constituída por um servidor Web e por um conjunto de páginas que usam o ADO para aceder à fonte. Existem dois tipos de utilizadores do sistema: os que trabalham com a aplicação em Windows são os que introduzem novos problemas e os resolvem. Os outros utilizadores são os que reportam os problemas e podem utilizar a parte da Web para consultar o seu estado.

O segundo exemplo é um conjunto de bibliotecas COM que usam o ADO para suportar um *front-end* que pode ser construído a *posteriori*. Este exemplo pretende demostrar um cenário possível para a utilização deste método de acesso.

No final deste capítulo são mencionados mais alguns exemplos da relação que existe entre o ADO e a multimédia e qual a contribuição deste como uma mais valia para as aplicações que necessitam de vários acessos a diferentes tipos de dados.

4.2. Trouble Tickets

4.2.1. Introdução

Neste capítulo o principal objectivo é apresentar o desenvolvimento de um sistema de *Trouble Tickets* para o CICA, Centro de Informática Prof. Correia de Araújo da Faculdade de Engenharia da Universidade de Porto.

O sistema deve permitir catalogar e gerir problemas relacionados com o parque informático e as suas relações com um vasto número de utilizadores: com a ajuda de um sistema deste tipo, uma instituição pode gerir melhor os seus problemas, o que possibilita o controlo sobre o tempo de resolução e a eficácia, entre outros parâmetros.

Não se pretende que o exemplo seja muito complexo a nível de funcionalidade do próprio sistema, mas sim que demonstre algumas das principais capacidades do ADO, nomeadamente a utilização dos seus objectos e como o ADO pode contribuir para uma aplicação multimédia. Na construção de *software*, podem existir várias formas para realizar certas operações. Por vezes o método escolhido para essas operações não foi o mais eficiente mas o que se pretendeu dar a conhecer, de uma forma simples, foi a utilização dos diferentes objectos.

Este sistema é constituído por duas partes distintas. A primeira parte refere-se ao *software* de gestão que é utilizado para manipular a informação, que foi implementado em *Visual Basic* por duas razões: o código ser de fácil compreensão, mesmo para quem não conheça a linguagem, e a rapidez de implementação. A segunda parte é constituída por uma aplicação Web, compatível não só com o *browser Internet Explorer* mas também com o *browser Netscape*, para visualizar os dados, ou seja, o estado de cada problema reportado. Por razões de compatibilidade com os *browsers*, algumas características do ADO não podem ser demonstradas, nomeadamente funções que comuniquem com o servidor *http* para, por exemplo, manter um *Recordset* aberto do lado do servidor, ou fazer o *bind* entre uma página Web e o servidor.

Se o ADO pôde ser usado com linguagens de programação que não sejam da Microsoft, o mesmo não se pode afirmar no caso de um servidor *http* que use ADO. Nestes casos o servidor tem mesmo de ser o *Internet Information Service* (IIS) ou o *Personal Web Server* da Microsoft, embora este não tenha nem o mesmo desempenho nem todas as funcionalidades do IIS.

Esta secção começa por integrar o exemplo no *Universal Data Access*, referindo como este é composto, como funciona o OLE DB e qual é a relação entre os objectos OLE DB e ADO. Descreve-se seguidamente o exemplo, isto é como foi implementado numa linguagem de programação e na Web. Descrevem-se também as principais funcionalidades de *software* e mostra-se o resultado visível dessa implementação.

4.2.2. Universal Data Access

O OLE DB é um interface de programação a nível do sistema para diversas fontes de informação, sendo uma especificação de vários interfaces *Component Object Model* (COM) que encapsulam vários sistemas de manipulação e serviços das fontes de informação. Estes interfaces permitem criar componentes de *software* baseados na plantaforma do *Universal Data Access.*

Considerando que o OLE DB é um interface a nível de sistema, o ADO é um interface ao nível da programação. O ADO apresenta um modelo de programação orientado para fontes de informação que permite programar aplicações servindo de interface ao OLE DB, através de várias linguagens.

A figura seguinte ilustra a arquitectura *Universal Data Access*, onde se pode ver os três tipos de componentes de base de dados: os fornecedores de informação, os serviços e os consumidores de informação.

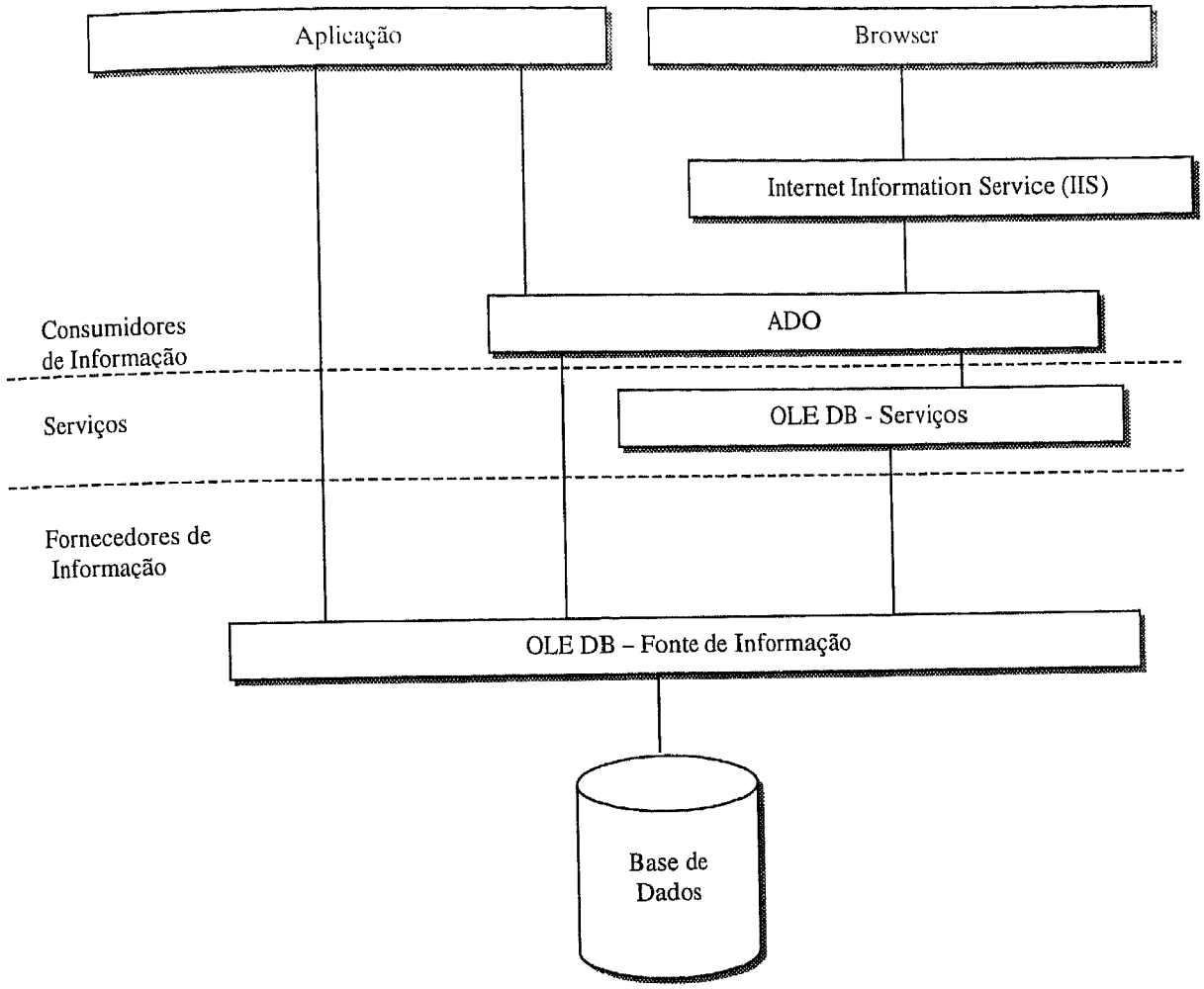

Figura IV-1 – *Universal Data Access*

Os fornecedores de informação são componentes que representam as diferentes fontes de informação, tais como bases de dados SQL, documentos *Excel* e outros já referidos. Os fornecedores de informação disponibilizam os dados utilizando *Rowsets*.

Os serviços são componentes que estendem a funcionalidade dos fornecedores de informação, oferecendo funções que não são suportadas nativamente pela fonte de informação. Por exemplo o *cursor engine* é um serviço que pode receber informação da fonte que só pode ser navegada num único sentido e "produz" informação que pode ser navegada em qualquer sentido.

Os consumidores são componentes que recebem a informação OLE DB. Um exemplo de um consumidor é o ADO que disponibiliza informação para as aplicações.

O OLE DB define uma hierarquia de quatro objectos ilustrados na seguinte figura

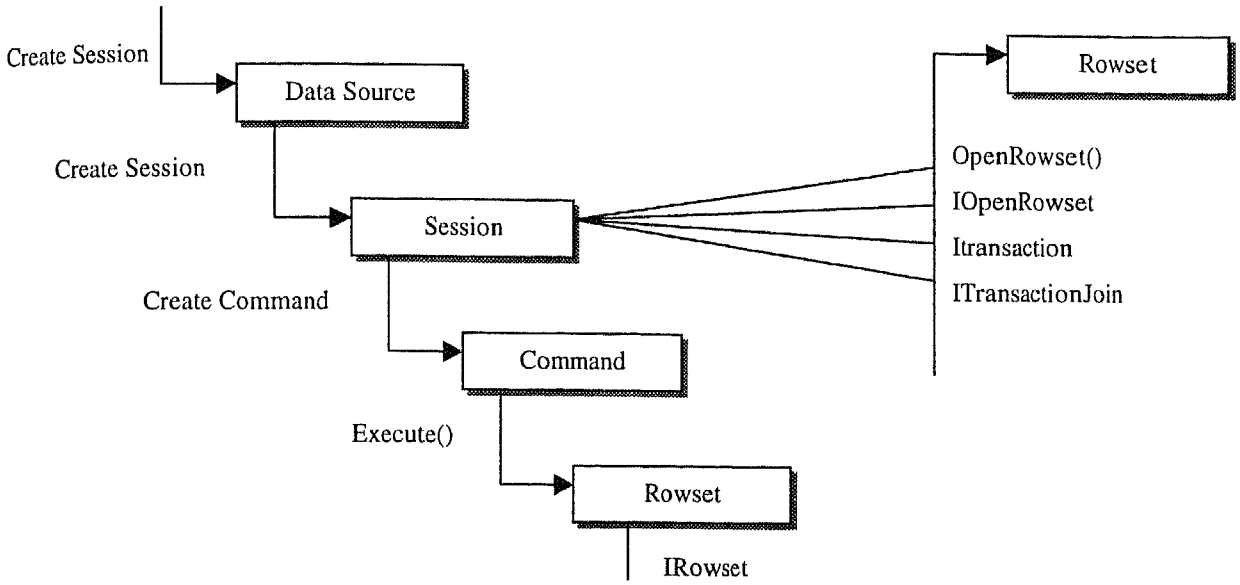

Figura IV-2 - OLE DB

O objecto *Data Source* encapsula funções que identificam uma fonte de dados, verifica se um determinado utilizador tem permissões para se conectar à fonte de informação e faz a conexão com a fonte de informação.

O objecto *Session* define o âmbito das transacções em operações efectuados pela conexão.

O objecto *Command* encapsula as funções que permitem a um consumidor invocar a execução da definição de informação ou comandos de manipulação de informação, tais como perguntas.

Os objectos *Rowset* são a representação comum de dados. Eles são criados tipicamente numa sessão ou como resultado da execução de um comando, mas podem ser gerados como o resultado de outros métodos que devolvem dados.

Antes de cada objecto OLE DB ser criado, a camada OLE DB que comunica com o ADO tem a oportunidade de pedir as capacidades e os comportamentos da fonte de informação para saber se necessita de passar pelos serviços OLE DB ou se pode fazer o pedido directamente à fonte de dados.

4.2.3. ActiveX Data Objects e a relação com os objectos OLE DB

ActiveX Data Objects (ADO) é um conjunto de interfaces de alto nível que comunicam com o OLE DB. Se o OLE DB é um poderoso interface para manipulação de informação, a grande maioria dos programadores não tem a necessidade de ter o controlo de baixo nível tal como o OLE DB o fornece.

A figura IV-3 mostra as relações entre os objectos do ADO

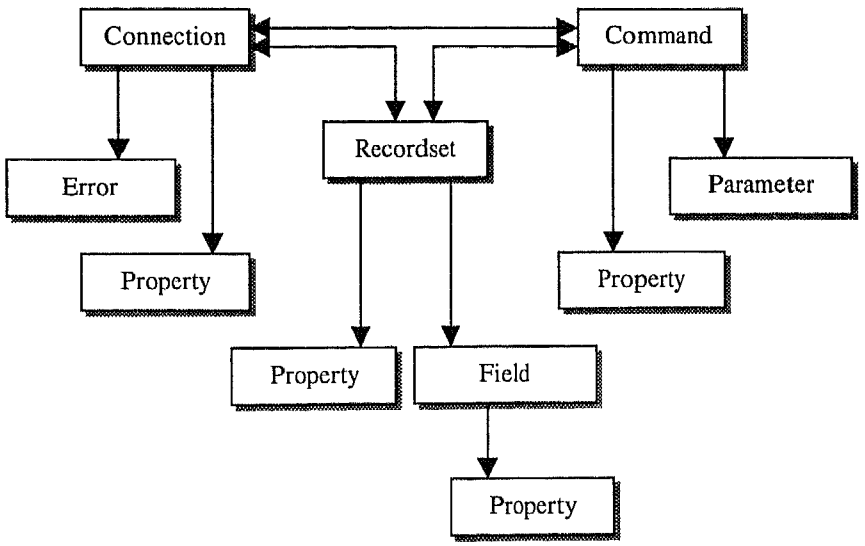

Figura IV-3 - Relações entre os objectos ADO

Os objectos *Connection*, *Command* e *Recordset* são os objectos do topo da estrutura que podem ser criados e destruídos separadamente. Embora se possa criar o objecto *Parameter* antes do objecto *Command*, àquele tem que ser associado ao objecto *Command* antes de ser usado. Os objectos *Error*, *Field* e *Property* só podem existir associados aos objectos representados na figura e nunca podem ser criados separadamente.

O objecto ADO *Connection* encapsula os objectos OLE DB *Data Source* e *Session.* Este objecto representa uma ligação à fonte de informação e contém as propriedades inerentes à conexão, controla as transacções locais, fornece um repositório único de erros e possibilita a execução de perguntas sobre a estrutura da fonte.

O objecto *Command* encapsula o objecto OLE DB *Command.* Este objecto tem como função executar expressões para manipulação de dados. No caso da fonte de dados ser uma base de dados relacional, o comando executa uma expressão SQL. O objecto permite especificar parâmetros para modificar o comportamento da expressão a ser executada.

O objecto *Recordset* encapsula o objecto OLE DB *Rowset.* Este objecto é o interface para os dados; contudo esses dados podem ser provenientes de vários formas como por exemplo de perguntas ou métodos para a abertura de registos. Este objecto controla o mecanismo de bloqueio de registos, o número de linhas extraídas, para além de outras funcionalidades. O objecto possui também uma colecção de objectos do tipo *Field* que contém informação sobre as colunas do *Recordset,* tais como nome e tamanho, e contém também o valor actual do campo. Usa-se este objecto para navegar na informação e para alterar essa mesma informação.

Cada objecto *Connection, Command* e *Recordset* possui uma colecção de objectos *Property.* Estas colecções permitem identificar as capacidades da fonte de informação, porque nem todas as fontes possuem as mesmas características nem as mesmas funcionalidades, sendo importante para o ADO saber dinamicamente quais são as especificações da fonte de dados. Esta funcionalidade é bastante importante porque evita ao ADO interferências com propriedades que só estão disponíveis em certas circunstâncias.

4.2.4. Parte Cliente Windows

Este exemplo permite exemplificar como é que uma aplicação usa o ADO como método de acesso aos dados. A primeira parte do exemplo diz respeito à aplicação em Windows.

Para melhor compreender a aplicação, a figura IV-4 mostra a definição da base de dados, quais são as tabelas mais importantes e quais as relações entre as mesmas.

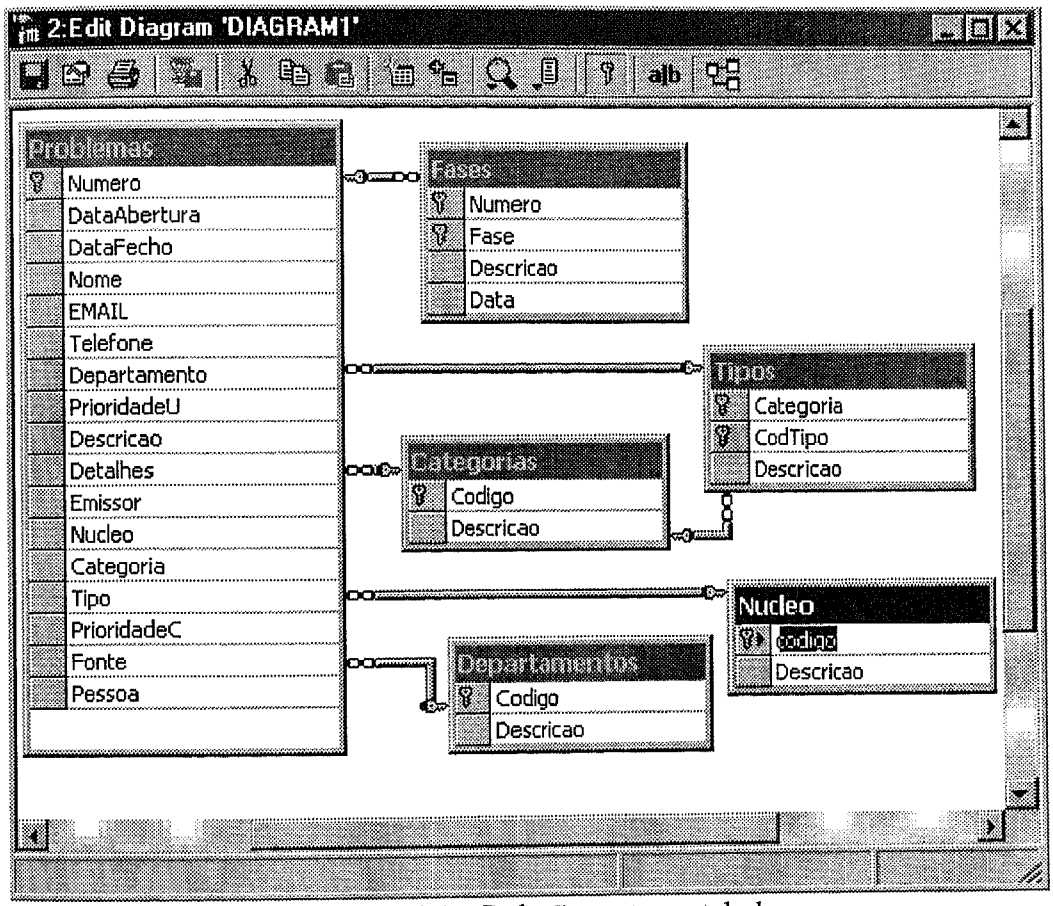

Figura IV-4 - Relações entre as tabelas

Tal como é perceptível na figura IV-4, existem várias tabelas: a tabela "problemas", onde são armazenados os dados reportados pelos utilizadores, a tabela "fases", onde é armazenada a informação sobre as várias etapas para a resolução dos problemas, a tabela "departamentos", que contém as diferentes áreas da Faculdade de Engenharia, a tabela "núcleos", que identifica quais os grupos de pessoas que o Centro de Informática possui para a resolução de problemas, a tabela "categorias", que identifica e armazena grupos de problemas, e a tabela "tipos", onde são armazenados os

diferentes problemas para uma determinada categoria. As relações entre as tabelas também são explicitadas na figura IV-4, onde se destaca a relação entre a tabela de problemas e fases porque um problema pode ter várias fases. As outras relações existem por questões de integridade de dados, (no caso da relação entre a tabela de problemas e a tabela de departamentos por exemplo), ou para ajudar a introdução dos dados por parte do utilizador.

Para aceder a uma determinada fonte de informação é necessário criar uma expressão para a conexão. O código seguinte permite criar uma conexão para uma fonte de informação, que é constituída pelo nome do *driver* OLE DB que vai ser utilizado, pelo utilizador e pela palavra chave (esta informação não é necessária neste momento), pelo catálogo e pelo nome da máquina onde está residente a fonte de informação:

```
Public Const Cn = "Provider=SQLOLEDB,Password=taplic,Persist Security
Info=True;User ID=taplic,Initial Catalog=TT,Data Source=BLACKBIRD"
Public CNN As New ADODB.Connection
```

Ao iniciar a aplicação o primeiro passo é criar a conexão com a base de dados.

```
CNN.ConnectionString = Cn ; atribui a expressão de conexão
CNN.CursorLocation = adUseClient ; identifica o dono dos dados
CNN.Open ; Conexão aberta.
```

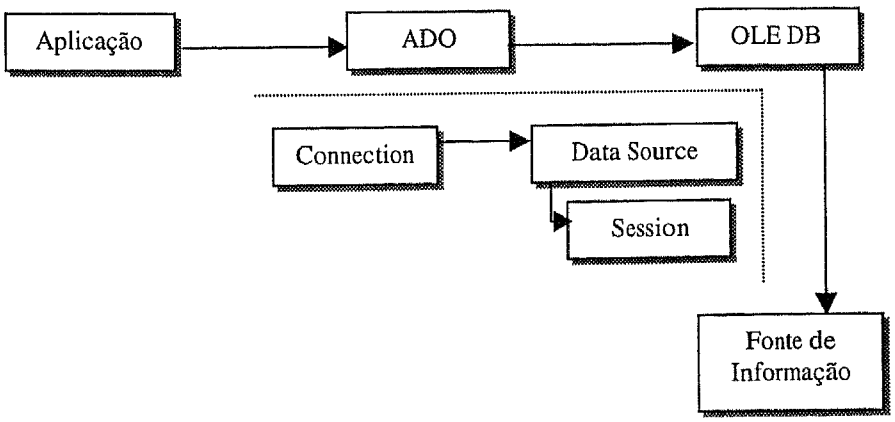

Figura IV-5 – Processo de conexão

A figura IV-5 representa o processo de conexão de uma aplicação com a fonte de informação. Como é possível observar na figura, o objecto *Connection* do ADO vai originar a criação dos objectos *Data Source* e *Session* que por sua vez ficam encarregues de fazer a comunicação com o *driver* da fonte de informação.

Tal como é perceptível pelo código anterior, logo após ser estabelecida a conexão com a fonte de informação, é apresentada ao utilizador a janela de validação.

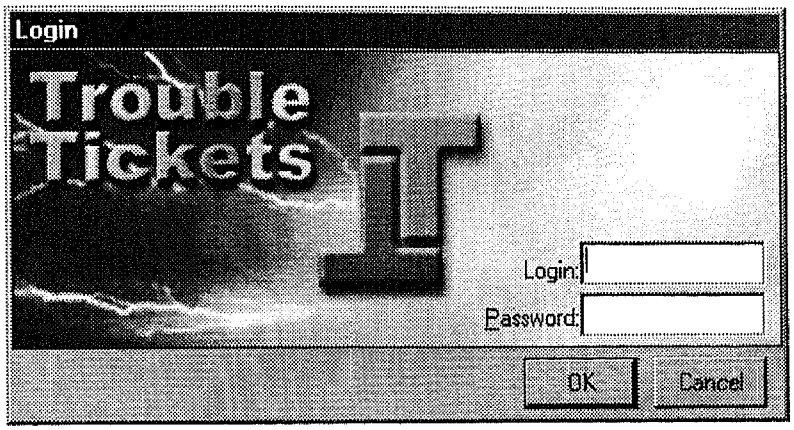

Figura IV-6 - Janela de validação

Seguidamente é criado um *Recordset* local à janela para a manipulação dos dados, `Private Rs As New Recordset`, e ao carregar a janela é executado o código:

```
Rs.CursorLocation = adUseClient ; é a aplicação cliente que contém os
dados do Recordset
Rs.Open "select * from login", CNN, adOpenStatic, adLockReadOnly
;abertura do Recordset.
```

O método *Open* do objecto *Recordset* faz com que os dados da tabela *login* sejam extraídos para a aplicação. Para isso são necessários os seguintes parâmetros: primeiro a expressão SQL para a fonte de dados saber qual é a informação pedida, segundo qual é o objecto *Connection,* ou seja, qual é a ligação com a fonte de dados, terceiro o tipo de cursor que vai ser utilizado e quarto qual o tipo de *lock* que vai ser feito na fonte de dados.

Supondo novamente que o utilizador introduziu um *login* e *password* correctos é apresentada a seguinte janela (figura IV-7).

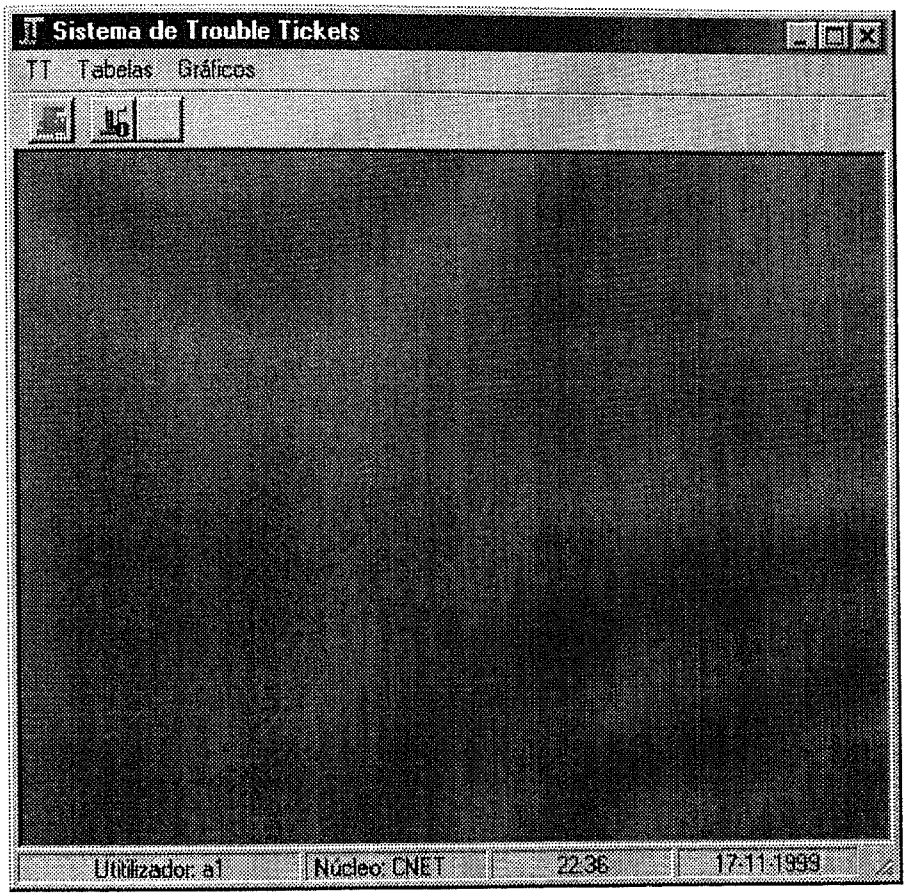

Figura IV-7 – Janela de Ambiente

Nesta janela (figura IV-7) são oferecidas ao utilizador várias opções: pode verificar se existem novos problemas e atribui-los a um núcleo (no caso do CICA existem diferentes núcleos para os diferentes tipos de problemas), pode inserir um novo problema, pode introduzir novos dados para um problema ou pode verificar qual o núcleo mais solicitado.

Se a primeira opção for a escolhida, o utilizador depara-se com a seguinte janela (figura IV-8).

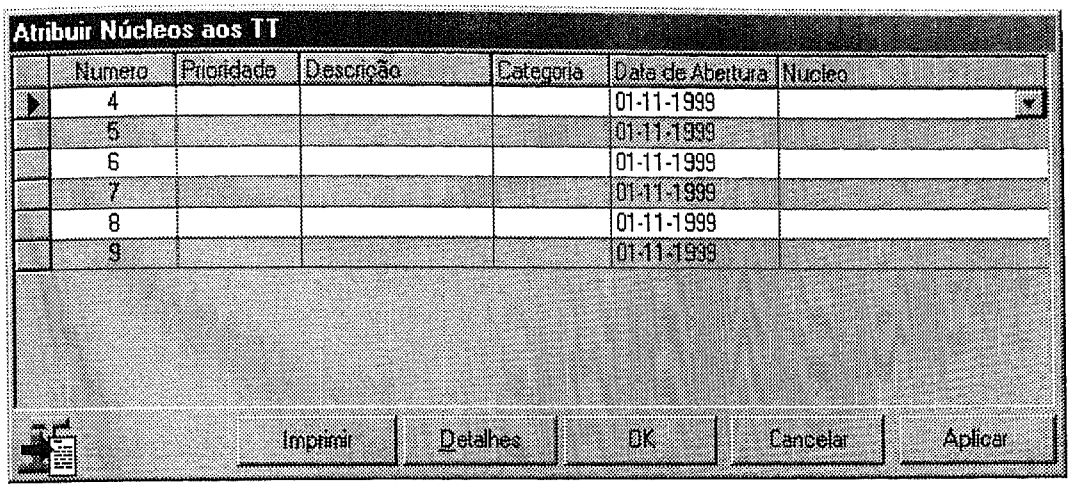

Figura IV-8 Atribuição dos problemas aos núcleos

Esta janela possui os seguintes *Recordsets* locais:

```
Private Rs As New ADODB.Recordset; Recordset para a tabela de problemas
Private RSCombo As New ADODB.Recordset; Recordset com os núcleos disponíveis.
```

Ao carregar esta janela vai ser executado o seguinte código:

```
; Abertura dos Recordsets
Rs.Open "Select * from Problemas where Nucleo IS NULL", CNN, adOpenStatic, adLockOptimistic
RSCombo.Open "Select * from Nucleo", CNN, adOpenStatic, adLockReadOnly
...
CNN.BeginTrans
...
```

Nesta rotina são abertos dois objectos do tipo *Recordset*, um com a função de apresentar na janela da tabela todos os problemas que ainda não foram atribuídos aos núcleos e o segundo com a função de preencher uma *ComboBox* para o utilizador saber quais os núcleos existentes. Ao ser executada esta rotina é inicializada uma

transacção, através do expressão *CNN.Begintrans*. Esta transacção permite estabelecer um nível de isolamento com a fonte de dados. Esta funcionalidade permite ao utilizador alterar os dados do *Recordset* sem que estes sejam actualizados imediatamente.

Entre várias opções que o utilizador pode escolher, as mais importantes são as de validar as atribuições que fez ou cancelar as escolhas. No caso de ter cancelado as alterações, a função cancelar executa o seguinte código:

```
...
CNN.RollbackTrans
...
```

Além de sair da janela, é executada a expressão *CNN.RollbackTrans*, que possibilita anular todas as alterações que foram feitas desde a execução da instrução *CNN.BeginTrans*. No caso de o utilizador pressionar a tecla OK e proceder à validação dos dados é executado o código:

```
...
CNN.CommitTrans
...
```

Esta rotina verifica se houve ou não alterações e, no caso afirmativo, actualiza os dados na fonte de informação se não existirem erros. Quando a instrução *CNN.CommitTrans* for executada é a altura em que a fonte de dados actualiza as alterações na base de dados.

Uma outra opção que o utilizador tem é a de introduzir um novo problema. Neste caso é apresentada ao utilizador a janela da figura IV-9.

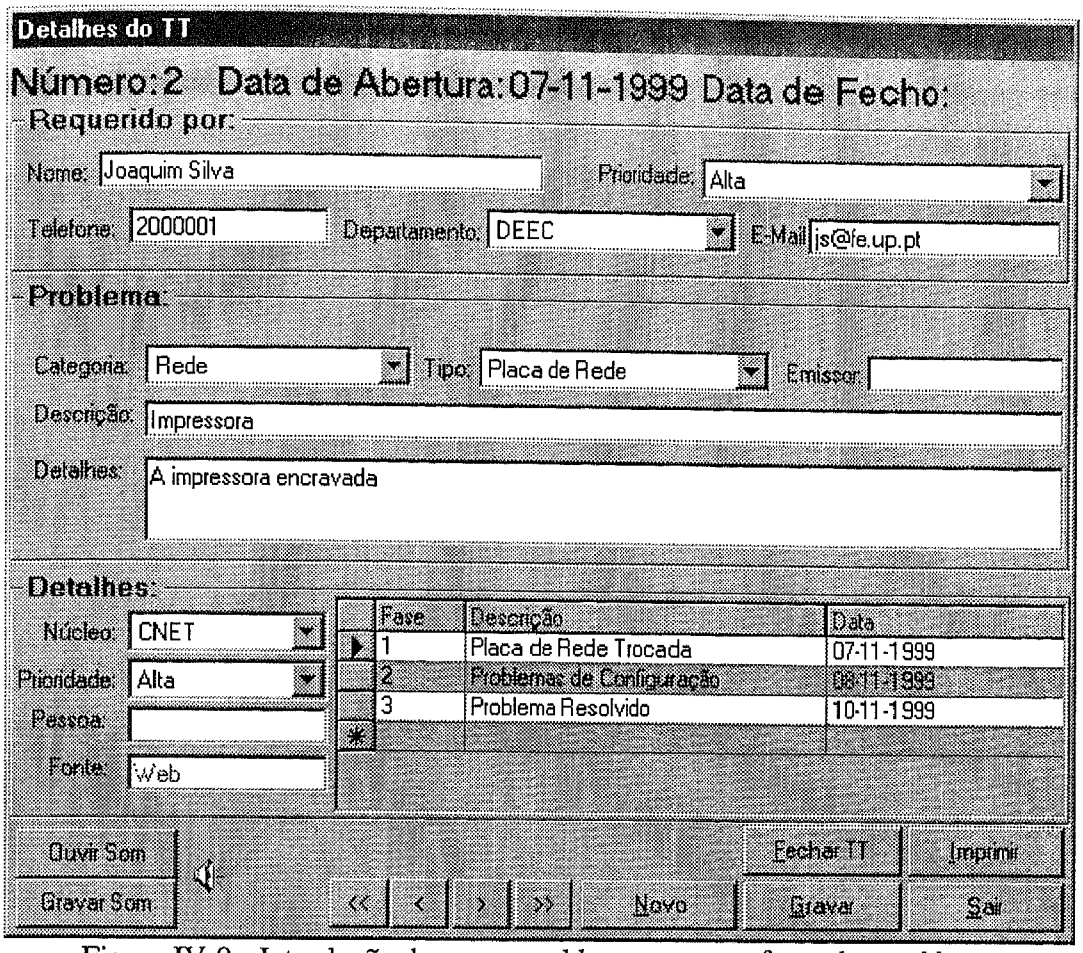

Figura IV-9 - Introdução de novos problemas e novas fases dos problemas

Nesta janela (figura IV-9) é possível introduzir novos problemas ou simplesmente introduzir uma nova iteração num determinado problema. A janela é constituída por três partes: qual foi o utilizador que apresentou o problema e os seus dados para um posterior contacto; os dados do problema, onde é especificada a categoria do problema, o tipo, que depende directamente do problema, a descrição sucinta do problema e uma zona de detalhe do problema; a última parte indica quem é que está a tratar do problema, qual o núcleo do técnico responsável, a fonte geradora do problema, qual a prioridade que o técnico atribui ao problema e, por fim, as fases que já foram executas para a resolução do problema.

Esta janela possui as seguintes variáveis locais:

Private WithEvents CAB As ADODB.Recordset; Recordset para a tabela de problemas; a opção WithEvents cria as subrotinas de eventos sobre o Recordset CAB

```
Private LIN As New ADODB.Recordset; Recordset para a tabela das fases
...
```

Ao carregar esta janela vai ser executado o seguinte código:

```
' Abertura dos Recordset
Nucleo.Open "select * from Nucleo", CNN, adOpenStatic,
adLockReadOnly; Tabela Nucleo
Cat.Open "Select * from Categorias", CNN, adOpenStatic,
adLockReadOnly; Tabela Categorias
... outras tabelas de apoio
CAB.Open "Select * from Problemas where Nucleo = '" &
Trim$(User.Nucleo) & "'", CNN, adOpenDynamic, adLockOptimistic; a
constante adOpenDynamic implica que outras alterações feitas por
outros utilizadores são logo visíveis
; Bind - Fazer a ligação entre os recordsets e os objectos que
mostram a informação
Set NLabel.DataSource = CAB: NLabel.Refresh
... fazer o bind dos outros objectos à base de dados
ActOutros; esta rotina permite extrair só os registos da tabela fases
de um determinado problema e permite também carregar só os tipos de
problemas de uma determinada categoria.
```

A rotina ActOutros carrega para um objecto *combobox* (mais concretamente para o objecto *TDBCombo6*) os tipos de problemas definidos na categoria e carrega também para a tabela fases só os registos do problema em questão.

```
Sub ActOutros()
If Tipo.State = adStateOpen Then; a propriedade state indica o estado
de um Recordset que pode ser aberto, fechado, etc.
    Tipo.Close
End If
Tipo.Open "Select * from Tipos where Categoria ='" & TDBCombo2.Text &
"'", CNN, adOpenStatic, adLockReadOnly;
Set TDBCombo6.RowSource = Tipo; faz a atribuição do Recordset ao
objecto
TDBCombo6.ReBind; faz com que o objecto releia o seu Recordset
; Tratar das Relações, neste caso da tabela fases em relação à tabela
problemas.
If LIN.State = adStateOpen Then LIN.Close
```

If NLabel.Caption = "" Then; a propriedade caption do objecto NLabel indica qual o número do problema em questão

*LIN.Open "Select * from fases where Numero = -1", CNN, adOpenStatic, adLockOptimistic;* no caso de não existir número (em situações de novo registo por exemplo) o Recordset é aberto com um número negativo para se ter a certeza que não existem linhas

Else

*LIN.Open "Select * from fases where Numero = " & NLabel.Caption, CNN, adOpenStatic, adLockOptimistic*

End If

Set TDBGrid1.DataSource = LIN; atribuição do objecto Lin que como foi comentado anteriormente contém a tabela fases.

TDBGrid1.ReBind; atribuição da tabela do Recordset para o objecto TDGGrid1

End Sub.

Supondo que o utilizador pretende criar um registo novo então é executada a seguinte função:

Sub Novo( )

Dim Ok As Boolean

If Not (CAB.EOF = CAB.BOF) Then; verifica se não está vazio ou seja se o início do ficheiro é igual ao fim do ficheiro, informação que é dada através da propriedade *EOF* e *BOF*

If CAB.EditMode <> adEditNone Then; a propriedade EditMode permite saber se o registo corrente do Recordset está em modo edição, criação ou se não teve qualquer alteração

...

CAB.AddNew; o método addnew permite por o Recordset em modo de criação de um novo registo

...

End Sub.

No fim de inserir, o utilizador grava a informação.

Private Function Grava(Optional Pergunta As Boolean = True) As Boolean

Dim Cn As New ADODB.Command; cria um objecto do tipo Command

Dim Pr As New ADODB.Parameter; cria um objecto do tipo Parameter

Dim Numero As Long

If CAB.EditMode = adEditNone Then; verifica através da propriedade EditMode do Recordset se houve alterações; neste caso se o valor é igual à constante adEditNone

Grava = True

Exit Function

End If

If CAB.EditMode = adEditDelete Then; verifica o caso de outro utilizador ter apagado o registo

Grava = False

CAB.CancelUpdate; se o registo foi apagado é necessario cancelar todas as alterações do Recordset

MsgBox "Atenção o Registo foi apagado por outro utilizador", vbExclamation, "Atenção"

Exit Function

End If

...

(mais validações)

...

If CAB.EditMode = adEditInProgress Then; no caso de fazer uma actualização a um registo já existente

On Error Resume Next; no caso de haver um erro prossegue

Err.Clear; limpa erros anteriores (função do Visual Basic)

CAB.Update; grava a informação do registo através do método Update

If Err > 0 Then

MsgBox Error(Err); no caso de haver um erro mostra-o

...

End If

If CAB.EditMode = adEditAdd Then; no caso de ser um registo novo

Cn.CommandText = "PRXTT"; através do objecto command executa um procedimento para saber qual é o próximo número disponível, com a propriedade CommandText refere-se o nome do procedimento

Cn.CommandType = adCmdStoredProc; tipo de comando, neste caso um procedimento remoto

Cn.ActiveConnection = CNN; indica qual é o conexão activa para a fonte de informação

Pr.Direction = adParamOutput; especifica a direcção dos resultados no caso do objecto Parameter

Pr.Type = adInteger; tipo de dados do objecto Parameter.

Pr.Name = "@Num"; nome do parâmetro

```
    Cn.Parameters.Append Pr;  acrescenta o parâmetro ao objecto
Command
    Cn.Execute; executa o comando
    NLabel = Cn(0);  primeiro parâmetro do objecto Command e neste
caso único
    DAbertura = Date
    CAB.Update; grava o registo
...
End Function.
```

Além de introduzir novos problemas, editá-los e gravá-los, é possível também navegar nos problemas através dos botões de navegação. Quando o utilizador pressiona um botão para se posicionar no primeiro registo é executado o seguinte código:

```
Private Sub Inicio()
...
CAB.MoveFirst; o método MoveFirst do objecto Recordset permite
posicionar o mesmo no primeiro registo
  End Sub.
```

No caso do utilizador pressionar a tecla de navegação de um único registo para trás, o código executado é o seguinte:

```
Private Sub Anterior()
...
    CAB.MovePrevious;  o método MovePrevios permite mover o registo
corrente uma posição para trás
    If CAB.BOF Then CAB.MoveFirst; neste caso se o registo se
posicionar antes do primeiro registo o método MoveFirst obriga o
Recordset a voltar à primeira posição
    ...
End Sub.
```


No caso do utilizador pretender movimentar um registo para a frente ou para o fim do *Recordset*, os métodos a usar são respectivamente os *MoveNext* e *MoveLast*.

No caso do utilizador introduzir uma nova fase, a tabela a ser actualizada é a tabela das fases. Mas o utilizador não tem que se preocupar com o número da fase ou com a

relação que esta tem com a tabela de problemas. Para perceber como é que se resolve este tipo de problemas o código seguinte ilustra este tipo de situação:

```
Private Sub TDBGrid1_BeforeUpdate(Cancel As Integer)
Dim RSc As Recordset; Cria um Recordset
Set RSc = LIN.Clone; o método clone do objecto Recordset cria um
Recordset exactamente igual ao primeiro.
If LIN.EditMode = adEditAdd Then; verifica se é um registo novo
    If LIN.RecordCount = 1 Then; no caso de este ser o primeiro
registo
        Total = 1
    Else; se não for um registo novo é necessário saber quantos
registos são; a propriedade RecordCount do objecto Recordset indica o
número de registos; para que a propriedade não devolva -1, ou seja,
não sabe quantos registos existem, é necessário indicar e posicionar
pelo menos uma vez o registo corrente no fim do  Recordset
        RSc.MoveLast
        Total = RSc.RecordCount
    End If
     TDBGrid1.Columns(0).Value = NLabel.Caption; o objecto NLabel
contém o número do problema em questão
    TDBGrid1.Columns(1).Value = Total; indica qual o número da fase
    ...
End If.
```

Esta rotina é executada exactamente antes da gravação do registo. O objecto onde se está a visualizar as fases do problema contém um conjunto de rotinas para melhor controlar os actos de gravação. Não existe no procedimento qualquer expressão para gravar, porque o objecto está ligado ao *Recordset*, e, ao trocar o registo corrente, este é gravado automaticamente a menos que haja indicação do contrário.

Também é possível introduzir comentários sob a forma de voz num determinado problema. Na figura IV-9 existem dois botões que possibilitam ao utilizador introduzir som e mais tarde poder ouvi-lo. O *stream* de áudio pode ser inserido na base de dados através do método *AppendChunk* e pode ser lido através do método *GetChunk* do ADO. Neste caso a aplicação usa o *Multimedia Control Interface* (MCI)

para captar ou reproduzir o som. Através da utilização dos métodos do ADO referidos e do MCI também é possível o tratamento de *streans* de vídeo.

Mas para o utilizador não ter que navegar nos problemas, pode visualizá-los numa lista e mesmo impor algumas restrições. A figura IV-10 mostra como é que isto se faz.

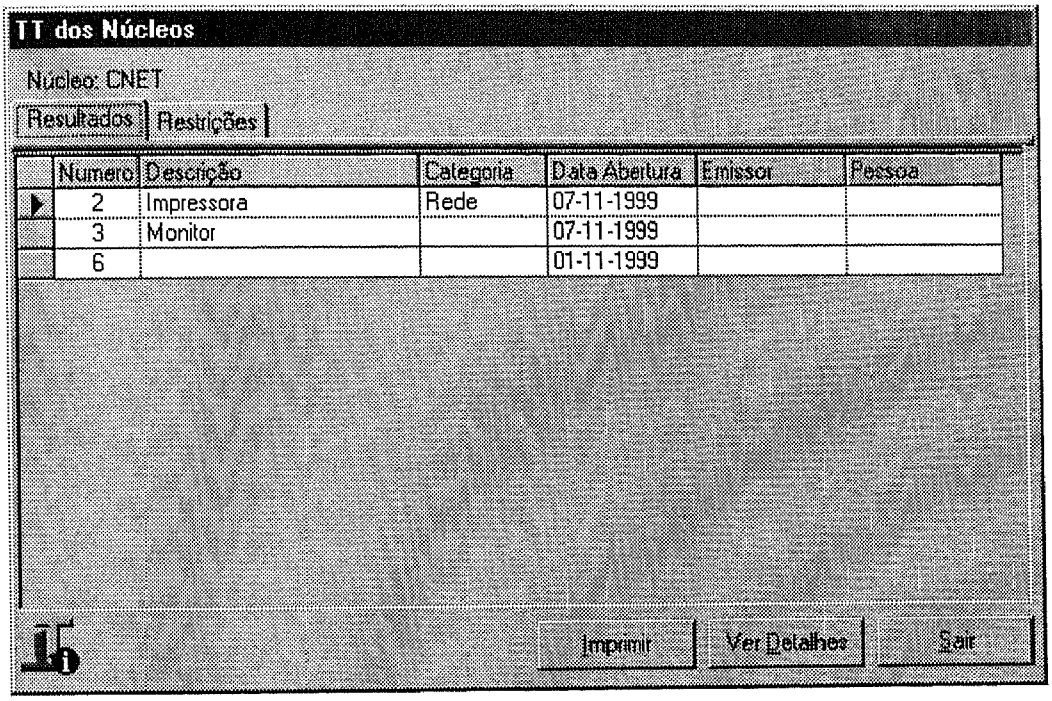

Figura IV-10 - Visualização de Problemas

Esta janela só possui como variável local um único *Recordset*, `Dim Rs As New ADODB.Recordset`, e quando se carrega a janela é executado o seguinte código:

```
Private Sub Form_Load()
Rs.Open "Select * from Problemas where Nucleo = '" & User.Nucleo &
"'", CNN, adOpenDynamic, adLockReadOnly; nesta situação o Recordset
pode ser aberto só para leitura; visto que é para consulta; a
constante adOpenDynamic implica que se houver novas introduções de
problemas estas ficam logo visíveis.
...
End Sub.
```

Também é possível filtrar problemas para facilitar a pesquisa como mostra a figura IV-11.

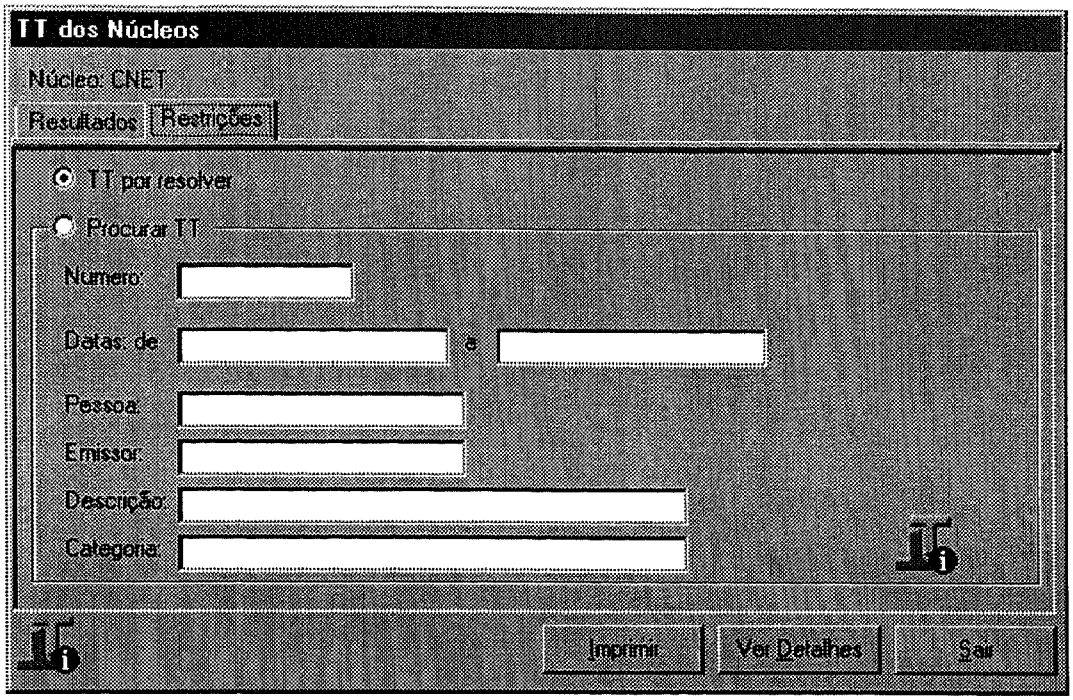

Figura IV-11 - Filtrar problemas

Quando o utilizador premir o *tab* de resultados, o filtro vai ser aplicado e o utilizador pode ver os resultados na grelha. Para implementar esta funcionalidade, basta fechar o *Recordset* actual e reabri-lo com uma restrição feita em SQL.

No caso do utilizador pretender visualizar qual a distribuição dos problemas pelos núcleos, pode seleccionar a opção gráficos do menu principal e obtém um gráfico que mostra exactamente essa distribuição.

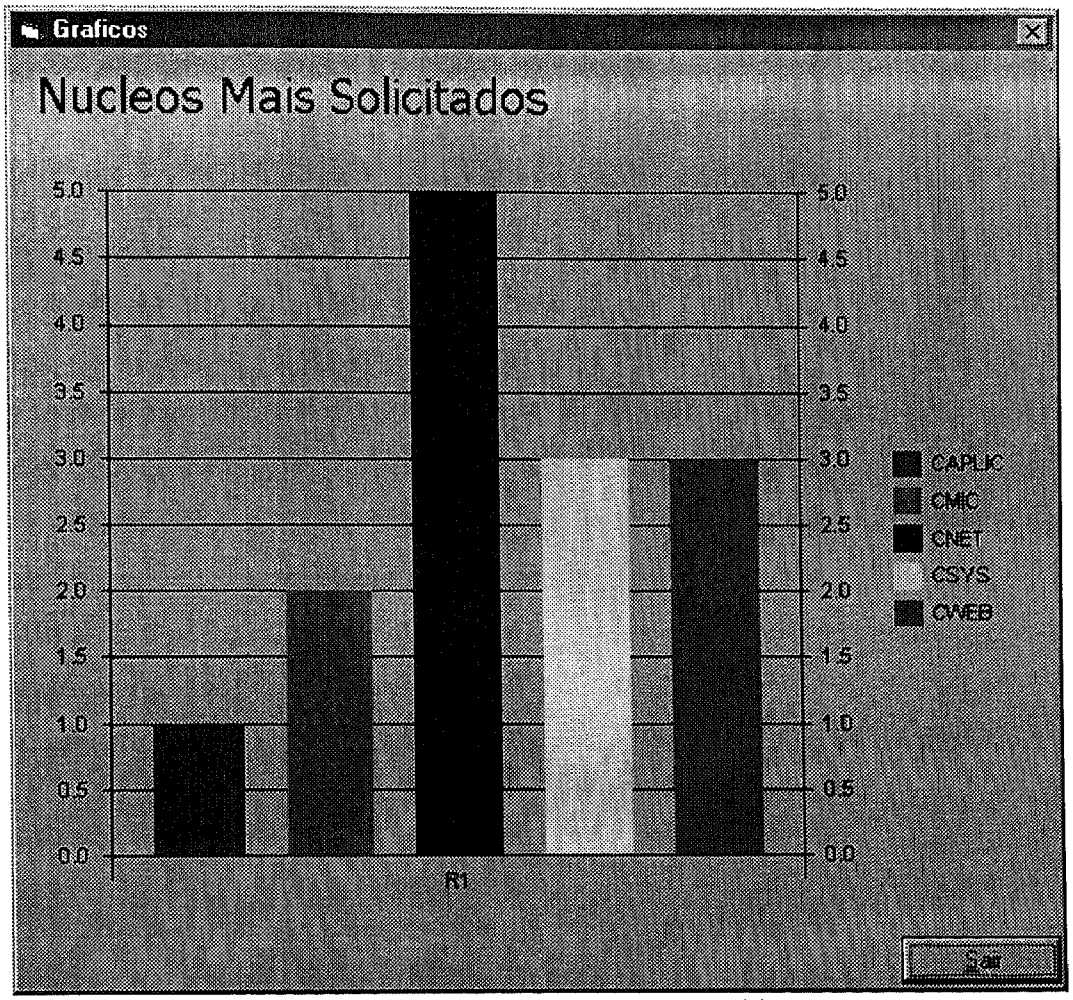

Figura IV-12 - Distribuição dos Problemas

Na figura IV-12 a janela possui como variáveis locais `Dim WithEvents Rs As ADODB.Recordset`; com a opção *WithEvents* para ser possível aceder aos eventos do *Recordset* e ao carregar a janela vai ser executado o seguinte código:

```
Private Sub Form_Load()
Set Rs = New ADODB.Recordset
Rs.CursorLocation = adUseClient; usa o cursor do lado do cliente
(opção por omissão)
Rs.Properties("Initial Fetch Size") = 1; coloca a propriedade Initial
Fetch Size a 1 para ser possível abrir o Recordset assincronamente
Rs.Open "Select count(*) as Result,Nucleo from problemas group by
Nucleo", CNN, adOpenKeyset, adLockOptimistic, adAsyncFetch; o
resultado da expressão SQL vai ser uma tabela com duas colunas, a
primeira contém o número de problemas atribuídos a cada núcleo e a
segunda vai ter a indicação do núcleo; a constataste adAsyncFetch
indica que o Recordset vai ser aberto assincronamente
```

```
If Not (Rs.State = adStateFetching) Then; verifica o estado do
Recordset, porque para além de ainda estar aberto pode também estar
em extracção
  FazGrafico; esta rotina consulta o Recordset e desenha o gráfico.
```

Através da constante *adAsyncFecth* verifica-se se a aplicação não vai parar à espera que os resultados retornem todos, como acontece na grande maioria dos casos. O que se passa é que, se os resultados forem todos extraídos, o método *Open* passa de assíncrono para síncrono e não há notificação por parte dos eventos que os dados já estão disponíveis ou que já estão disponíveis x de um total previsto y. Esta passagem normalmente acontece quando o *Recordset* retornado é rapidamente extraído e não contém mais de 50 registos como resultado. Se a abertura do *Recordset* for feita assincronamente existem duas funções que indicam o estado da extracção. A primeira indica, entre outras informações, a quantidade de registos extraídos em relação a um total previsto de registos. O código seguinte mostra como tratar esta situação:

```
Private Sub Rs_FetchProgress(ByVal Progress As Long, ByVal
MaxProgress As Long, adStatus As ADODB.EventStatusEnum, ByVal
pRecordset As ADODB.Recordset)
    Registos.Caption = "Extraidos: " & Progress & " do total " &
MaxProgress
    FazGrafico
End Sub.
```

Como parâmetros o procedimento indica o número de registos já extraídos, através da variável *Progress*, o número máximo de registos previsíveis, através da variável *MaxProgress*, e o estado do ligação, ou seja, se houve erros ou não, através da variável *adStatus* que é um apontador para o *Recordset* que foi alterado.

A função seguinte é invocada quando foram extraídos todos os registos:

```
Private Sub RS_FetchComplete(ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    ...
     FazGrafico ; faz o gráfico final
End Sub.
```

A rotina é invocada pelo ADO quando termina a extracção dos dados. Como parâmetros o procedimento utiliza uma variável que indica se houve erros que, no caso de existirem, residem na colecção de erros da conexão. As outras variáveis dão a mesma informação que a função *FetchProgress*.

Esta capacidade de extrair dados assincronamente permite uma certa interactividade com o utilizador visto que, como é demonstrado no exemplo, o gráfico vai sendo actualizado à medida que os dados vão sendo devolvidos pela fonte. Esta funcionalidade permite também que o utilizador realize outras operações enquanto os dados são extraídos visto que o processamento da aplicação não é afectado.

Outra relação entre duas tabelas surge no caso das categorias e dos tipos, porque os tipos de problemas estão directamente ligados às suas categorias.

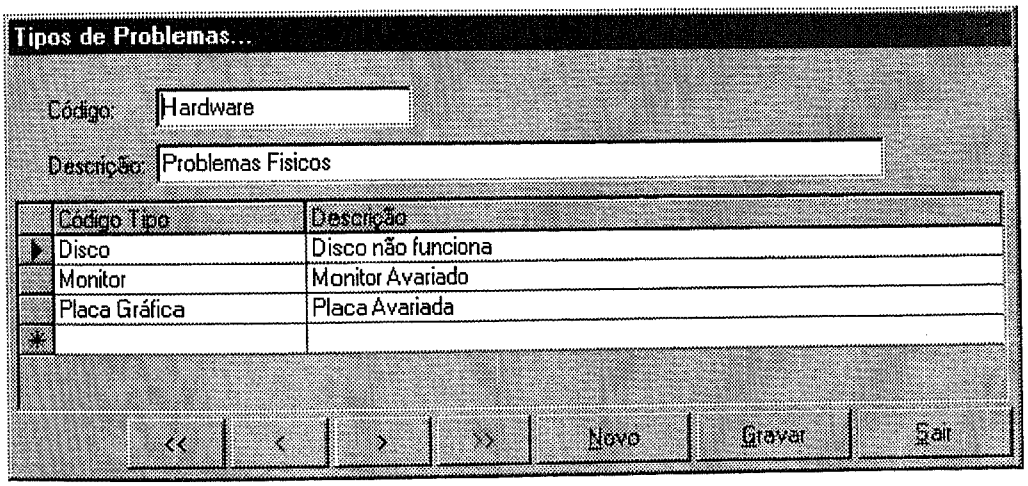

Figura IV-13 - Categoria e tipos de problemas

A figura IV-13 mostra que a categoria *Hardware* possui diferentes tipos de problemas, os quais podem ser problemas com os monitores, discos ou placas gráficas.

Esta janela possui como varáveis locais: `Private WithEvents Cat As ADODB.Recordset,` para a tabela Categoria, e `Private Tipo As New ADODB.Recordset,` para a tabela Tipo, e ao carregar a janela vai ser executado o seguinte código:

```
Private Sub Form_Load()
Set Cat = new Recordset
Cat.Open "Select * from Categorias", CNN, adOpenKeyset,
adLockOptimistic; abertura do Recordset
...
End Sub.
```

Neste caso a forma de actualizar o *Recordset* da tabela tipo que é uma restrição da tabela categoria é através do evento *MoveComplete*:

```
Private Sub Cat_MoveComplete(ByVal adReason As ADODB.EventReasonEnum,
ByVal pError As ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal
pRecordset As ADODB.Recordset)
If adReason = adRsnMoveFirst or adReason = adRsnMoveLast or adReason
= adRsnMoveNext or = adRsnMovePrevious then
If Tipo.State = adStateOpen Then Tipo.Close; no caso do Recordset
estar aberto então fecha
If Text1.Text = "" Then
Tipo.Open "Select * from Tipos where Categoria = ''", CNN,
adOpenStatic, adLockOptimistic
Else
    Tipo.Open "Select * from Tipos where Categoria = '" & Text1.Text
& "'", CNN, adOpenStatic, adLockOptimistic
End If
Set TDBGrid1.DataSource = Tipo
...
```

Este procedimento pode ser usado para fazer certos tipos de actualização e, nesses casos, actualiza o *Recordset* tipo (relação entre as tabelas "categorias" e "tipo" em que por cada categoria pode haver vários tipos). Como parâmetros o procedimento indica: a razão da movimentação; informação de erro se ocorreu algum durante a mudança de registo; uma variável de *status* para saber o estado, que pode ser do tipo *adStatusOk*, se foi feita a movimentação, ou *adStatusCancel*, se não houve movimentação, e um apontador para o *Recordset* onde foi feita a alteração.

Também existe o procedimento *WillMove* que é chamado no momento em que acontece a movimentação do *Recordset*.

4.2.5. Parte Cliente Web

Como já foi referido, a segunda parte do sistema é constituída por uma aplicação Web para poder dar informação aos utilizadores que reportaram os problemas. Com a utilização de um browser é possível consultar o estado de um determinado problema.

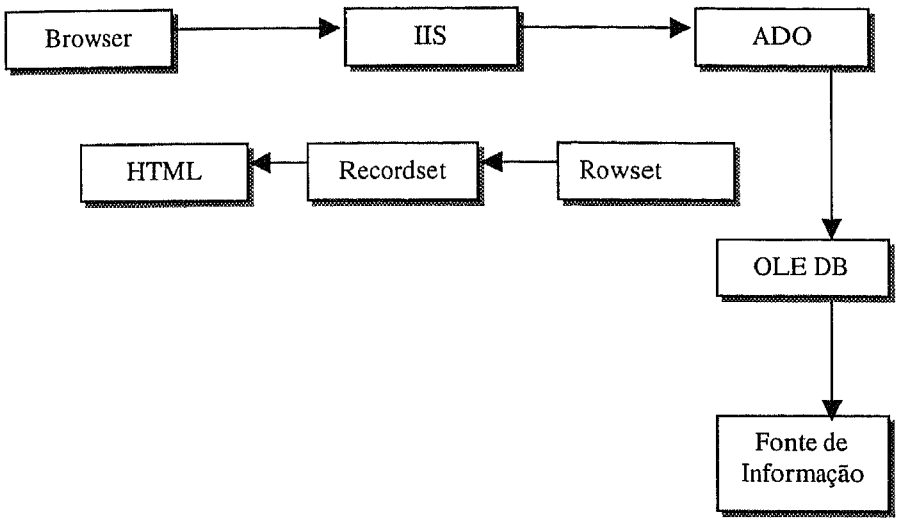

Figura IV-14 - Fluxo de Informação

A figura IV-14 mostra qual é o fluxo de informação quando um utilizador solicita uma determinada página Web. Como é possível verificar na ilustração o browser faz um pedido ao servidor Web e este, se necessitar de aceder a informação, fa-lo usando o ADO. Como este é um interface de alto nível, vai invocar os objectos de sistema para realizar esta função ou seja invoca os objectos OLE DB. Após a informação ser retirada da fonte pelo OLE DB, este envia-a ao ADO através de um *Rowset*. O nível ADO encapsula esse objecto num *Recordset* e disponibiliza a informação para o servidor Web. Após esta informação estar disponível, o servidor pode enviar a página para o cliente final.

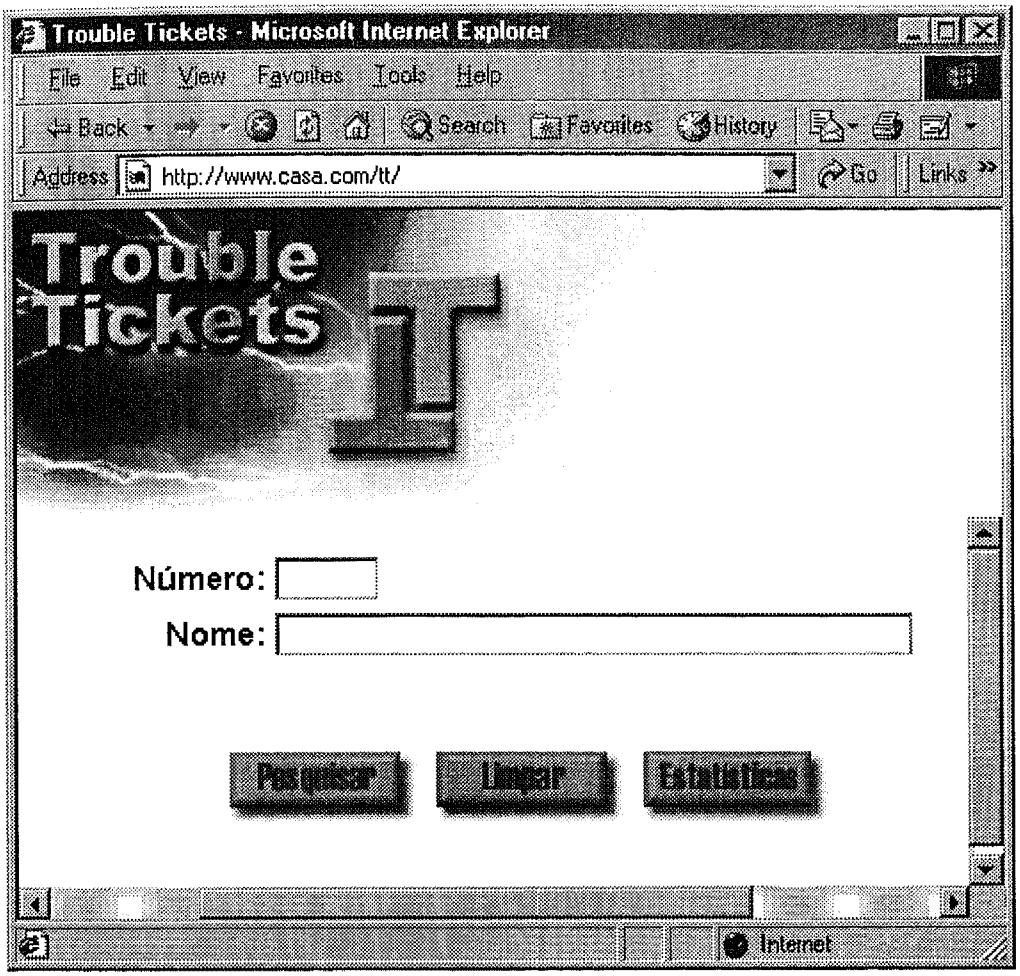

Figura IV-15 - Procura de TT utilizando a Web

Supondo por exemplo que um utilizador introduz um determinado número e prime o botão pesquisar (figura IV-15), a próxima página que vai encontrar (supondo que não houve qualquer erro) é a página com o nome do utilizador que reportou o problema, que foi devolvida pela fonte de informação. Esta página é gerada dinamicamente utilizando ASP e o acesso à fonte de informação é feito através da criação de uma conexão.

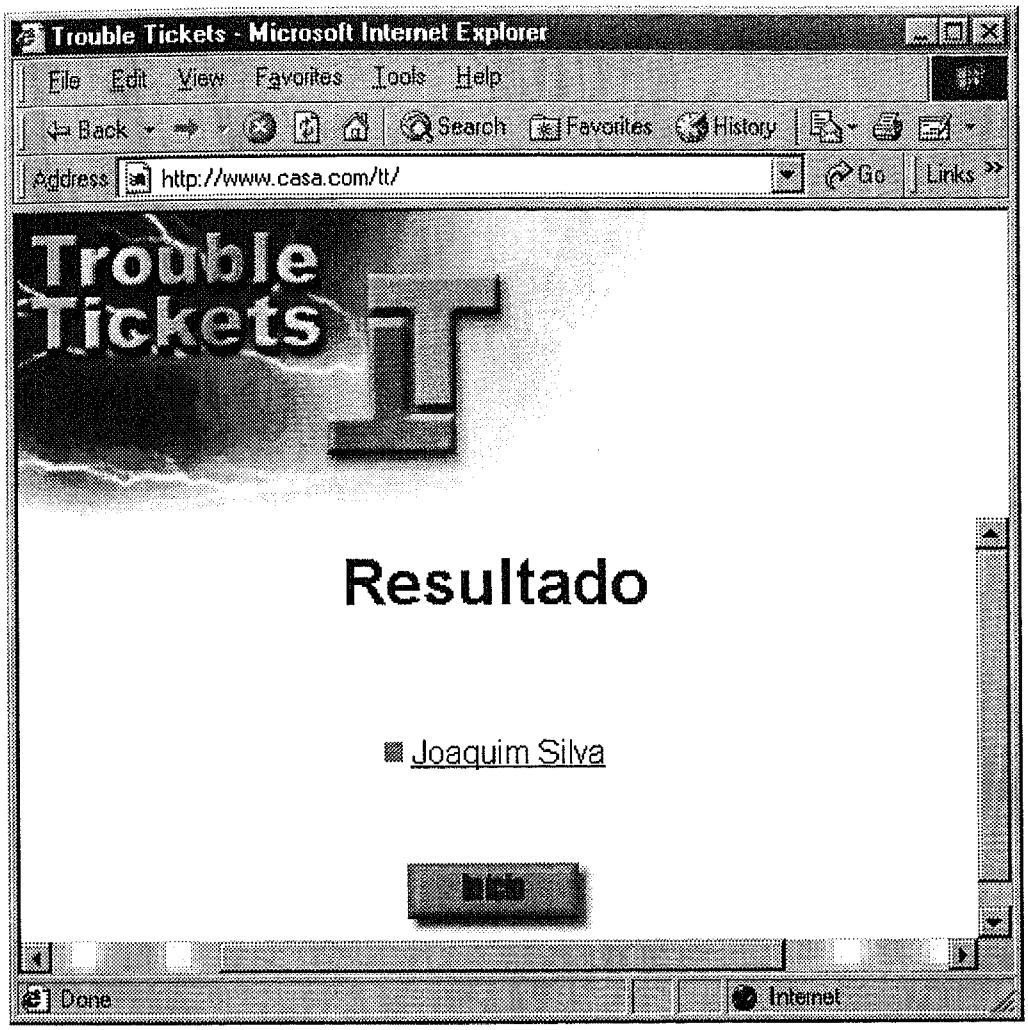

Figura IV-16 - Resultado de uma pesquisa na Web

Esta página (figura IV-16) foi criada através do seguinte código:

```
<!--#include file="common/ADOVBS.inc"-->; biblioteca para tratamento
de strings em VBScript
<!--#include file="common/IASUtil.asp"-->; biblioteca para a
utilização das constantes do ADO
<%
      strNumero = Request("P_NUMERO"); vai buscar o número que o
utilizador introduziu
      strNome = Request("P_NOME"); vai buscar o nome que o utilizador
introduziu
       Set cnTT = Server.CreateObject("ADODB.Connection"); cria uma
conexão
```

```
      Set rsTT = Server.CreateObject("ADODB.Recordset"); cria um
Recordset
      strCNString="Provider=SQLOLEDB.1;Persist Security Info=False;
User ID=sa;Initial Catalog=TT;Data Source=casa"; connection string
tal como na situação da primeira parte
      cnTT.Open strCNString; abre a conexão
      blnWhere = False
      strSQLTT = "select numero, nome from problemas"
      If (Not strNumero="") Then; constroi a expressão SQL
            strSQLTT = strSQLTT & " where "
            blnWhere = True
            strSQLTT = strSQLTT & " numero=" & strNumero
      End If
      If (Not strNome="") Then
      … construir SQL se for fornecido o nome.
      End If
      strSQLTT = strSQLTT & " order by numero"

%>
<html>
…
      rsTT.Open strSQLTT, cnTT; abrir o Recordset
…
            rsTT.MoveFirst
            While (CheckNextRS(rsTT))
                  rsTT.moveNext
…
            Wend
…
<% cnTT.Close %>; fechar a conexão
… concluir o html
```

Supondo que o utilizador prime o nome "Joaquim Silva" é devolvida uma nova página *html* onde se pode ver os dados do problema.

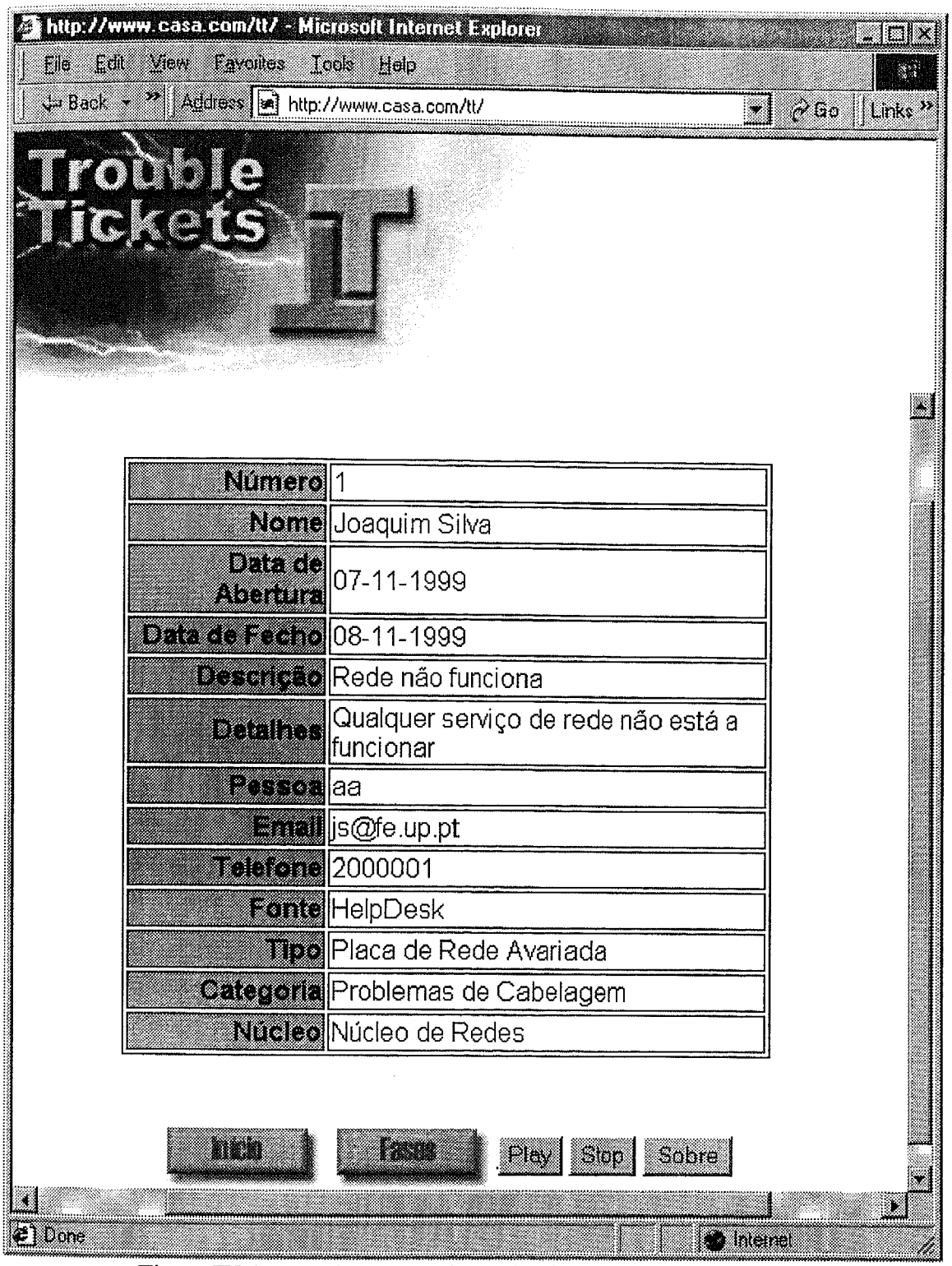

Figura IV-17 - Dados de um determinado TT usando a Web

Para gerar a página Web (figura IV-17) é necessário o seguinte código:

```
<%
    strNumero = Request("P_NUMERO"); saber qual é o numero do TT
```

```
    Set cnTT = Server.CreateObject("ADODB.Connection"); cria um
objecto Connection
      Set comdTT = Server.CreateObject("ADODB.Command"); Cria um
objecto Command
      Set rsTT = Server.CreateObject("ADODB.Recordset"); Cria um
objecto Recordset
      strCNString="Provider=SQLOLEDB.1;Persist Security Info=False;
User ID=sa;Initial Catalog=TT;Data Source=casa"; Connection string
      …
      cnTT.Open strCNString; abre a conexão
      comdTT.ActiveConnection = cnTT; atribui a conexão ao objecto
Command
      comdTT.CommandText = strSQLTT; atribui o SQL
      Set rsTT = comdTT.Execute; executa o comando e devolver o
resultado para um Recordset
…
       Escrever  o resultado do Recordset em html
html….
```

Se um determinado problema tiver som é possível ouvi-lo, podendo o utilizador pressionar o botão "play" tal como mostra a figura IV-17. Esta funcionalidade é feita através da utilização do *Windows Media Player*. Para ser possivel esta funcionalidade é necessario o seguinte código:

```
<OBJECT ID="MediaPlayer1" WIDTH=0 HEIGHT=0
 CLASSID="CLSID:22D6f312-B0F6-11D0-94AB-0080C74C7E95">
 CODEBASE="http://www.microsoft.com/netshow/download/en/nsasfinf.cab
#Version=5,1,51,115">
 <Param Name="FileName" Value= <%= (rsTT("Media").getchunk) %>
<Param Name="ShowDisplay" Value="False">
</OBJECT>
<INPUT TYPE="BUTTON" NAME="BtnPlay" VALUE="Play">
<INPUT TYPE="BUTTON" NAME="BtnStop" VALUE="Stop">
<INPUT TYPE="BUTTON" NAME="BtnAbout" VALUE="Sobre">
<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnAbout_OnClick
   MediaPlayer1.AboutBox
End Sub
Sub BtnPlay_OnClick
```

```
    MediaPlayer1.Play
End Sub
Sub BtnStop_OnClick
    MediaPlayer1.Stop
    MediaPlayer1.CurrentPosition = 0
End Sub
-->
</SCRIPT>
```

Se o utilizador pretender visualizar as fases dos problemas pode premir o botão fases e uma nova página dinâmica será criada com a informação pretendida (figura IV-18).

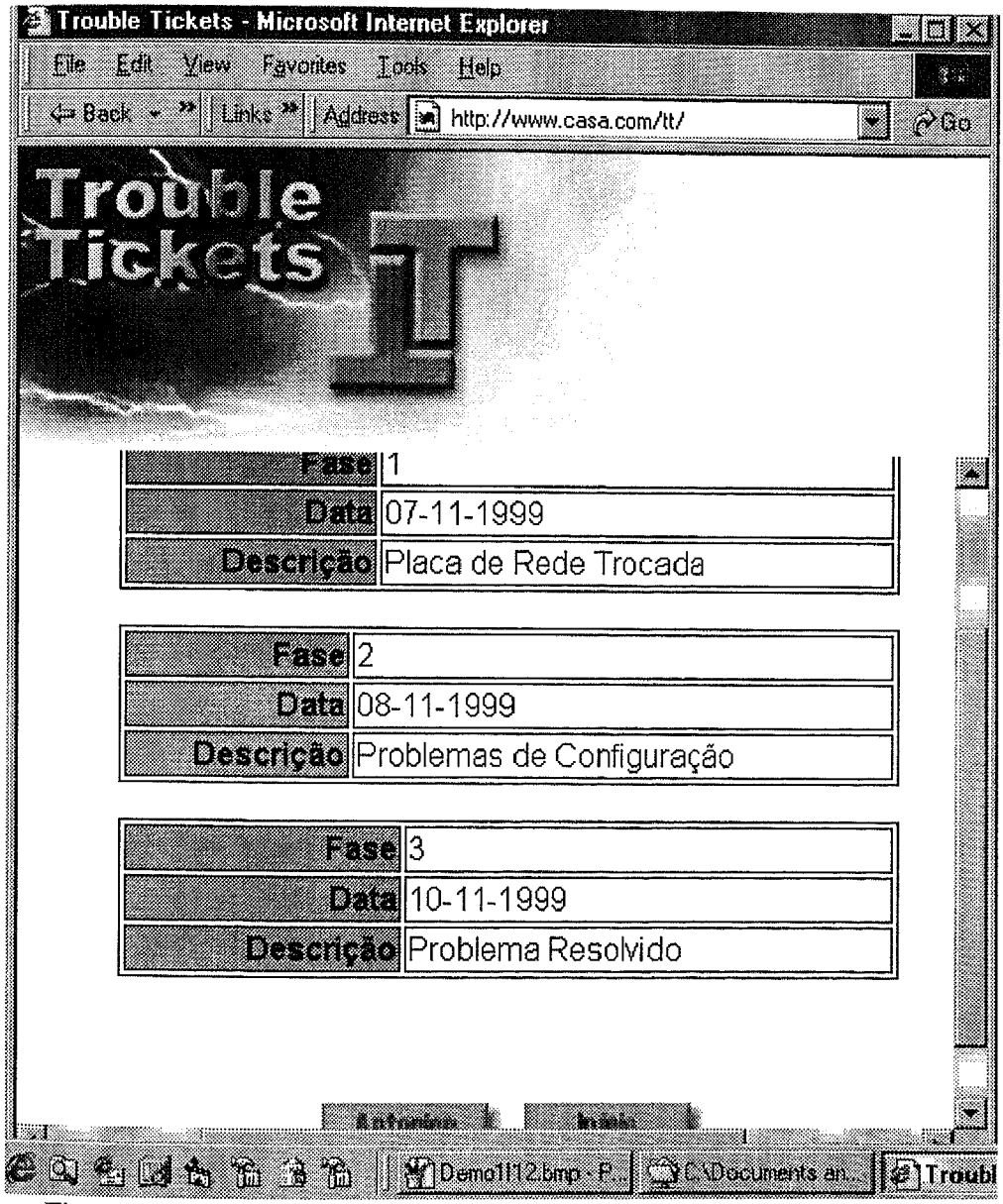

Figura IV-18 - Visualização das fases do um problema utilizando a Web

Para criar esta página (figura IV-18) é necessário o seguinte código:

```
...
      strNumero = Request("P_NUMERO"); vai buscar o número do TT
      Set cnTT = Server.CreateObject("ADODB.Connection"); cria uma
conexão
      Set rsTT = Server.CreateObject("ADODB.Recordset"); cria um
Recordset
      strCNString="Provider=SQLOLEDB.1;Persist Security Info=False;
User ID=sa;Initial Catalog=TT;Data Source=casa"; connection string
      cnTT.Open strCNString; abre a conexão
      strSQLTT = "select fase, descricao, data from fases"
      If (Not strNumero="") Then
            strSQLTT = strSQLTT & " where "
            strSQLTT = strSQLTT & " numero=" & strNumero
      End If

      strSQLTT = strSQLTT & " order by fase"

%>
<html>
...
            rsTT.MoveFirst; mover-se para o primeiro registo
            While (CheckNextRS(rsTT)); faz enquanto existirem linhas
... concluir o html
```

O utilizador tem ainda a possibilidade de ver quais os núcleos mais solicitados via Web, bastando para isso premir o botão estatística na página de abertura.

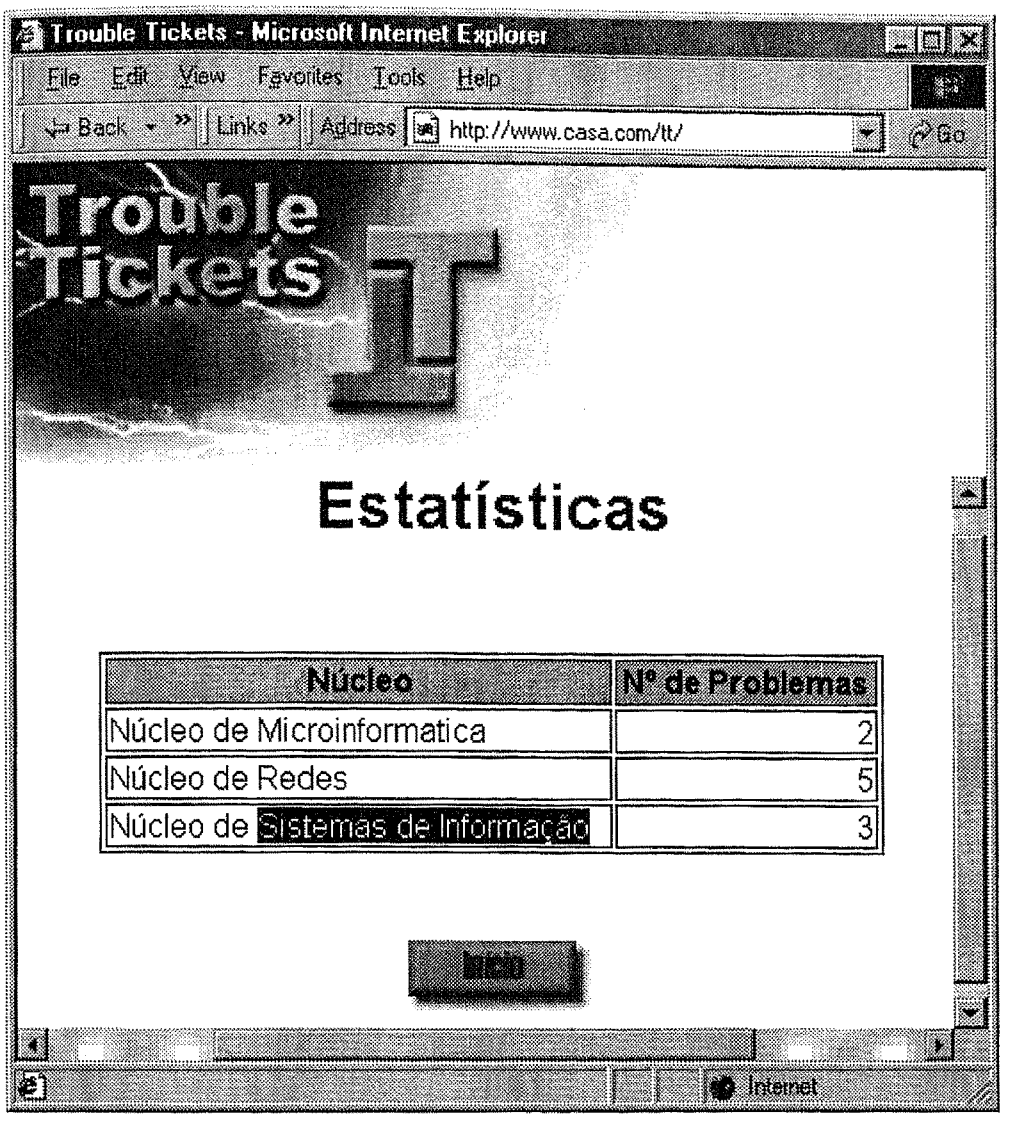

Figura IV-19 - Estatísticas via Web

Para construir esta página dinâmica (figura IV-19) é necessário o seguinte código:

```
...
        Set cnTT = Server.CreateObject("ADODB.Connection"); criar
uma conexão
        Set rsTT = Server.CreateObject("ADODB.Recordset"); criar um
Recordset
     strCNString="Provider=SQLOLEDB.1;Persist Security Info=False;
User ID=sa; Initial Catalog=TT;Data Source=casa"
     cnTT.Open strCNString; abrir a conexão
     strSQLTT = "select n.codigo as codigo,n.descricao as nucleo,
count(*) as total from problemas p, nucleo n where p.nucleo =
n.codigo group by n.codigo,n.descricao"; SQL para extrair os
resultados
```

```
%>
<html>
...
      rsTT.Open strSQLTT, cnTT
...
            rsTT.MoveFirst; movimenta para ao primeiro registo
            While (CheckNextRS(rsTT)); se existir registos
...
            rsTT.moveNext; move para o registo seguinte
            Wend
... concluir o html
```

4.2.6. Conclusão

Este exemplo serviu para ilustrar a utilização dos diferentes objectos do ADO em duas situações: numa aplicação em Windows e na Web através dos ASP.

Esta ferramenta é bastante simples de usar e é bastante uniforme nas várias plataformas em que pode ser usada, não havendo alterações significativas na forma de utilização. Algumas funcionalidades não estão disponíveis para a construção de páginas Web, como por exemplo os eventos, mas a grande maioria das propriedades e métodos está disponível.

O exemplo pretendeu focar principalmente a forma como se faz uma conexão e como é que os resultados estão disponíveis para serem utilizados. Foi também demonstrado como é possível abrir uma conexão assíncrona com a fonte de informação e como é possível tratar os dados.

Pretendeu-se ainda testar a utilização de parâmetros e do objecto *Command* para chamar procedimentos da fonte de dados. Este tipo de solução é bastante mais rápido, do que passar o SQL e esperar pelos resultados.

4.3. Aplicações em 3 níveis

4.3.1. Introdução

A maioria das empresas de programação com uma posição significativa no mercado, que desenvolvem aplicações para o ambiente Windows têm como estratégia desenvolver também ferramentas para, de uma maneira mais fácil e fiável, construir as suas aplicações. Desta forma, as empresas de desenvolvimento de *software* contêm nas suas equipas grupos de programadores que implementam algumas áreas específicas das aplicações, sem terem por vezes o conhecimento exacto de como é que a aplicação funciona no seu todo. Uma dessas áreas pode ser o exactamente o controlo, a comunicação e a gestão dos dados de uma base de dados. Nesta área o ADO pode representar uma boa solução.

Este segundo exemplo pretende demostrar como se pode usar o ADO como método de acesso aos dados e ao mesmo tempo satisfazer as necessidades das aplicações. Este exemplo pretende demonstrar como é possível construir uma mecanismo para aceder aos dados com qualquer tipo de apresentação utilizando um mecanismo baseado em 3 níveis. A perspectiva da apresentação é essencialmente a das equipas de desenvolvimento.

4.3.2. Exemplo

Neste exemplo, vamos ver uma aplicação bastante simples baseada em 3 níveis (Figura IV - 20): nível de apresentação, nível lógico e o nível de comunicação com a fonte de informação. Cada um é implementado separadamente e como veremos, a separação destes serviços permite uma maior flexibilidade na construção da aplicação.

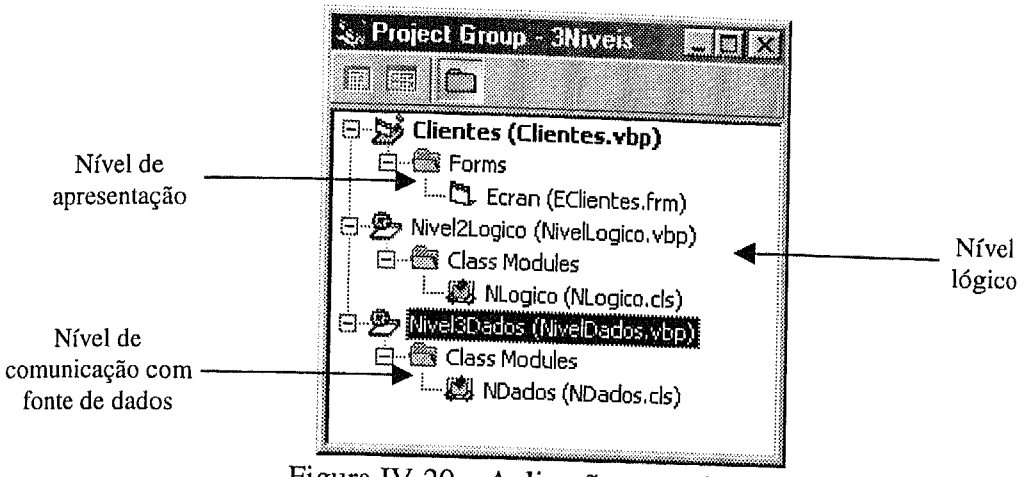

Figura IV-20 – Aplicações em três níveis

O exemplo consiste na extracção de informação de uma tabela de clientes de uma fonte de informação. Para se compreender melhor o funcionamento do exemplo, vamos seguir passo a passo o que vai acontecendo ao executar a aplicação. Quando que a aplicação começa, é apresentado ao utilizador um ecrã com uma lista e um botão (Figura IV-21) e, ao premir esse botão, a aplicação vai buscar à base de dados os nomes dos clientes e apresenta-os numa lista.

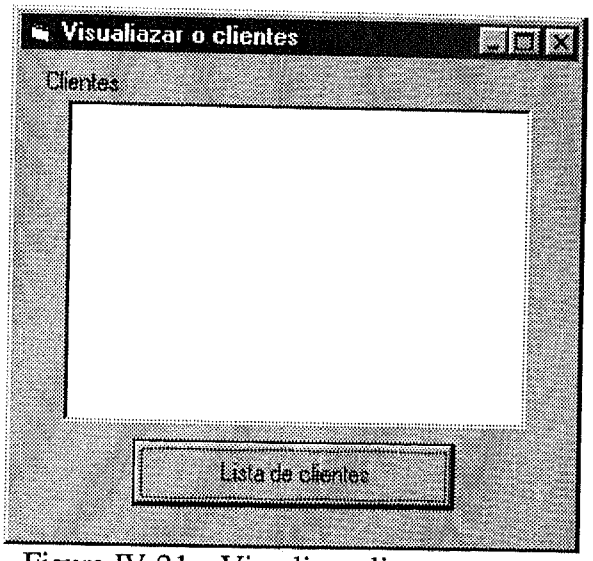

Figura IV-21 – Visualizar clientes

Quando é premido o botão "Lista de clientes", é executado o código da figura IV-22

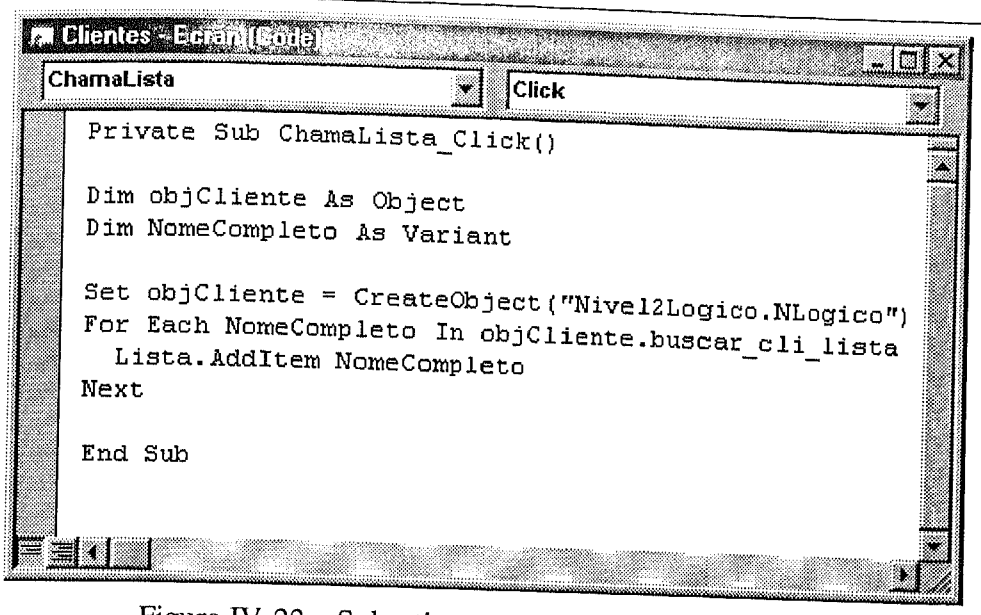

Figura IV-22 – Subrotina para carregar a lista de Clientes

Primeiro é usado o comando *Set* para criar uma ligação entre o objecto *objCliente* e o interface *Nivel2Logico* que contém a classe *Nlogico* (nível dois). Quando esta ligação estiver criada, será possível usar todas as propriedades e métodos desta classe.

Neste caso o objecto *objCliente* chama o método *buscar_cli_lista* que pertence à classe *NLogico*. Quando o método é invocado, o processamento passa para a interface *Nivel2Logico*, onde está implementada esta classe (Figura IV-23).

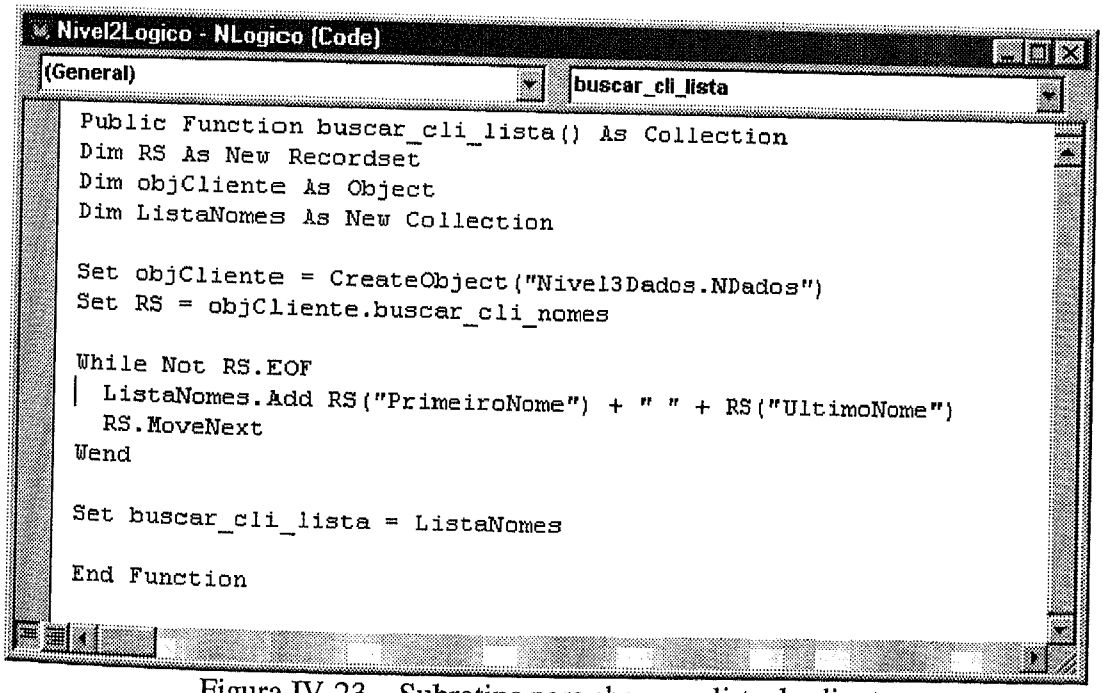

Figura IV-23 – Subrotina para chamar a lista de clientes

Como se pode ver o método *buscar_cli_lista* retorna uma colecção (um apontador para uma lista ligada no caso de ser implementado em C ou C++), que vai conter elementos com os nomes dos clientes.

Para preencher esta colecção vai ser criado um novo objecto a apontar para outro interface chamado *Nivel3Dados* com a classe *Ndados* (ou seja nível três). De seguida é invocado o método *buscar_cli_nome* (que é implementado pela classe *NDados*) para extrair da base de dados os nomes dos clientes (Figura IV-24).

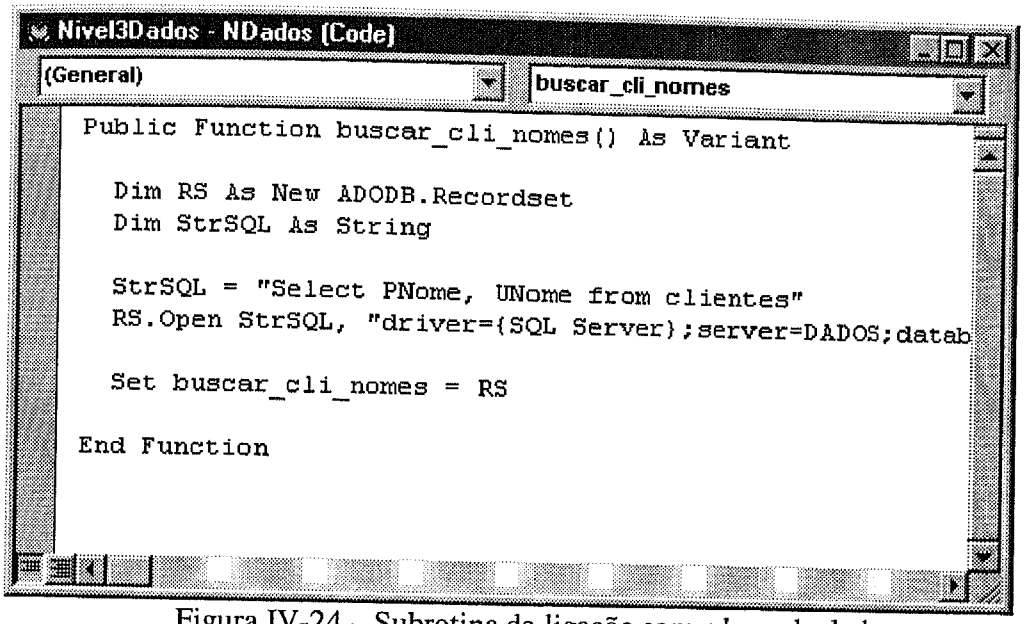

Figura IV-24 – Subrotina de ligação com a base de dados

O objecto *NDados* contém toda a especificação da base de dados. Centralizar toda a especificação de código de acesso à base de dados num único interface de comunicação de dados permite-nos alterar em qualquer altura a fonte de informação sem afectar os outros níveis, ou seja, o nível de apresentação e o nível lógico da aplicação. Este tipo de filosofia permite também criar aplicações escaláveis uma vez que as conexões com a base de dados podem ser abertas e fechadas em vez de permanecerem abertas longos períodos, como acontece nas aplicações com um único nível ou com dois níveis.

A função *buscar_cli_nome* simplesmente envia uma expressão SQL para uma fonte de informação (neste caso SQL Server, mas pode ser qualquer outra), para ser

processada e retornar o resultado para o objecto que invocou o método, ou seja, para o objecto *NLogico*.

Alterações das especificações dos dados ou optimizações das fontes de dados podem agora ser feitas no nível de dados, sem afectar os níveis lógico e de apresentação. Estes níveis só usam dados provenientes do nível de dados.

Quando o objecto *NDados* obtiver a informação, passa-a ao objecto *NLogico* na forma de *Recordset* e, nesta altura, o objecto *NLogico* pode tratar os dados. Ao terminar esse tratamento, os dados são novamente devolvidos (neste caso sob a forma de uma colecção) ao nível da apresentação. Finalmente os dados podem ser apresentados ao utilizador (figura IV-25). Neste exemplo, o nível de apresentação percorre a colecção e apresenta a informação ou seja os nomes dos clientes numa *listbox*, sem saber como foram tratados nem que métodos foram usados para extrair a informação da fonte.

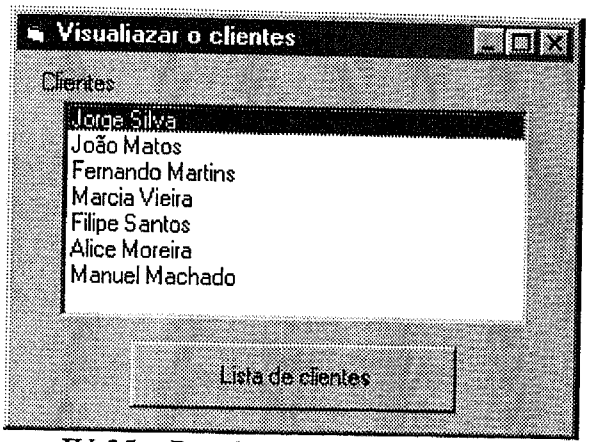

Figura IV-25 – Resultado do pesquisa de clientes

Os níveis de apresentação, lógico e de dados comunicam entre si através do *Component Object Model* (COM), mecanismo este que permite aos programadores concentrarem todo o trabalho na implementação das funcionalidades em cada um dos três níveis.

O *Distributed* COM (DCOM) permite aos programadores desenvolverem componentes que podem ser distribuídos por várias máquinas, podendo obter maior escalabilidade, e permite a reutilização de objectos. Um mecanismo normalmente

utilizado para alojar estes componentes é o *Microsoft Transaction Server*, mas não é obrigatório que assim seja.

Existem alguns benefícios na construção da aplicação usando várias camadas de código. Primeiro, é possível minimizar a distribuição de *software*, porque só é necessário fornecer actualizações quando estas são feitas ao nível da apresentação, nunca sendo necessário fazer distribuição de *software* quando as alterações se situam nos níveis lógico ou de dados. Segundo, se todos os acessos à fonte de informação são feitos no nível de dados, não é necessário instalar motores da base de dados, nem *drivers* ou qualquer *software* de acesso a dados nos clientes, o que reduz significativamente os custos associados com a manutenção do sistema. Uma terceira vantagem que pode ter algum impacto numa abordagem de três níveis é o facto de ser possível utilizar o nível lógico e o nível de dados em diferentes tecnologias de apresentação: pode ser criado um exemplo utilizando a Web para o nível da apresentação construído no *Visual Interdev* (Figura IV-26), e utilizar a parte lógica existente.

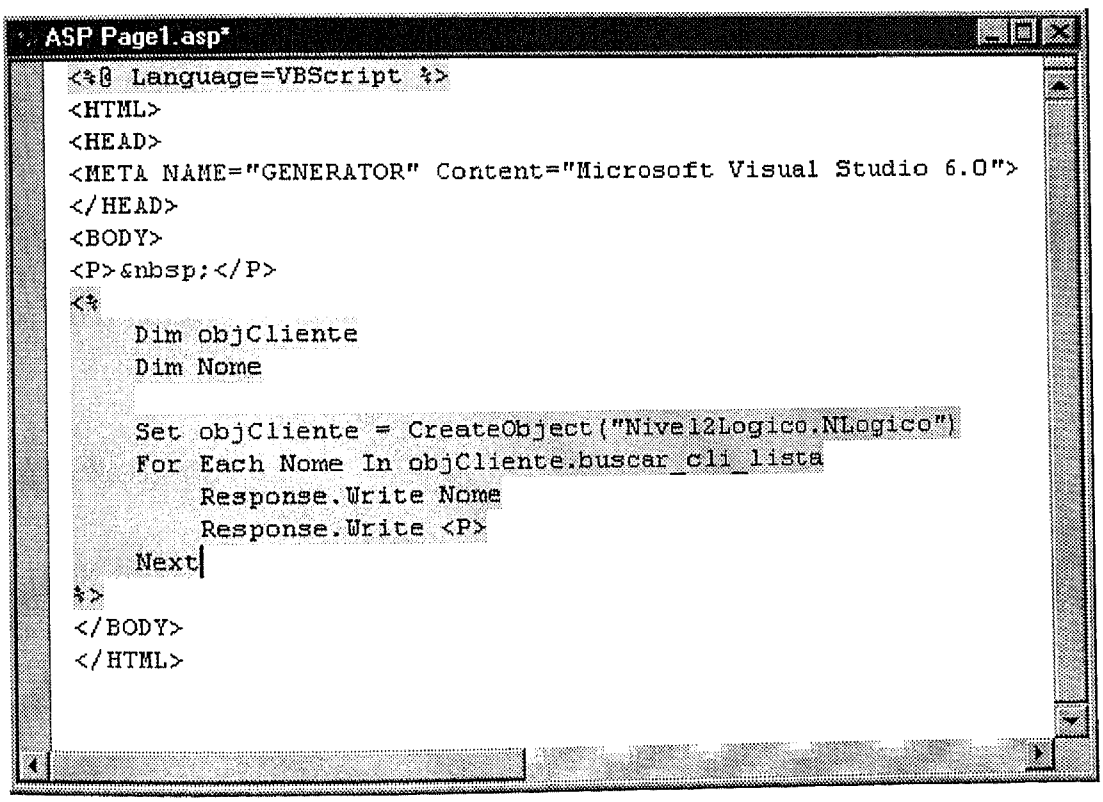

Figura IV-26 – Exemplo da tecnologia com a utilização de ASP

Este página usa o objecto *NLogica* para obter informação sobre os nomes dos clientes. Com algumas alterações no código elaborado em *Visual Basic* no nível de apresentação, é possível criar em *VBScript* do lado do servidor um mecanismo para aceder ao objecto *NLogico*. Quando a aplicação corre, cria simplesmente o objecto, invoca o método *buscar_cli_lista* e mostra os nomes dos clientes num browser, utilizando o método *Write* do objecto *Responde* existente nas *Active Server Pages* (ASP).

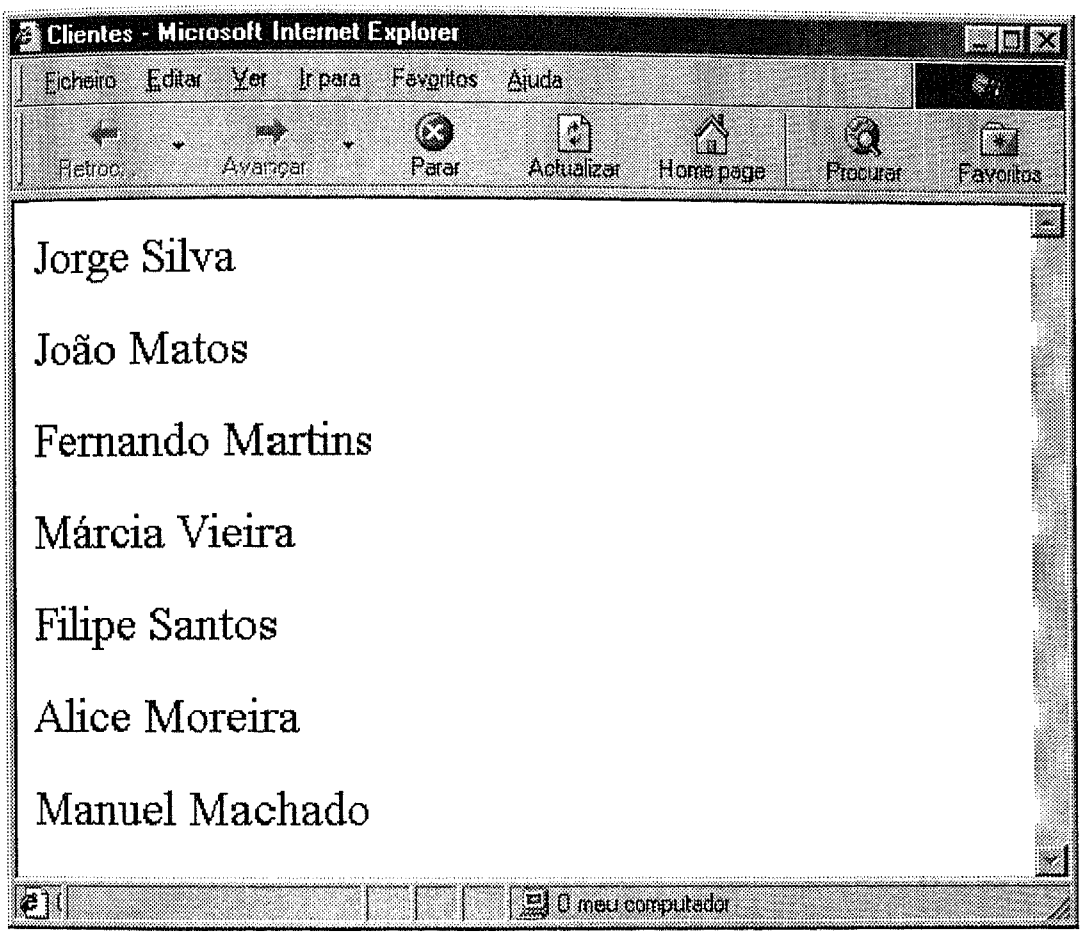

Figura IV-27 – Resultado da utilização de ASP

Quando se cria o nível de apresentação usando ASP ou DHTML, o código é processado pelo servidor Web e é enviado para o browser (Figura IV-27).

É também possível reutilizar os níveis lógico e de dados em aplicações tais como Microsoft *Word* ou *Excel*. No próximo exemplo foi criada uma aplicação simples em *Excel* que usa mais uma vez o objecto *NLogica* para obter os nomes dos clientes e colocá-los numa folha de cálculo.

O código necessário para que se possa usar o objecto *NLogica* foi construído em *Visual Basic for Applications* e é bastante idêntico ao código em *Visual Basic*.

Depois de correr uma macro em VBA (Figura IV-28), a lista de clientes é mostrada na folha de cálculo (Figura IV-29).

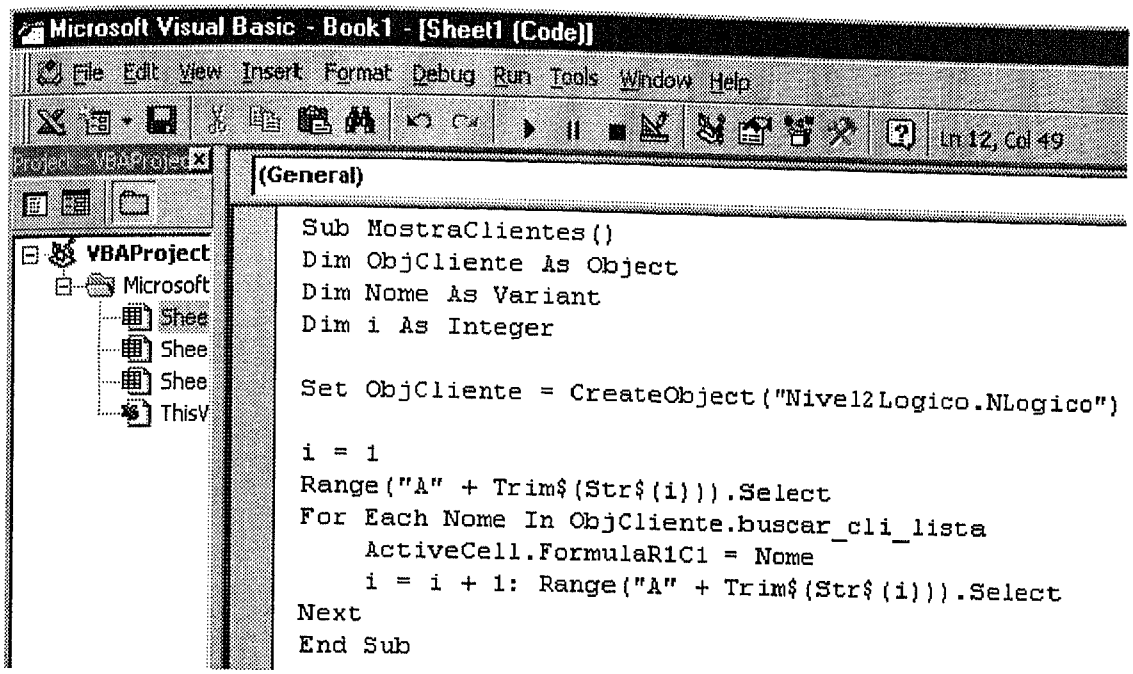

Figura IV-28 – Utilização da tecnologia no Excel

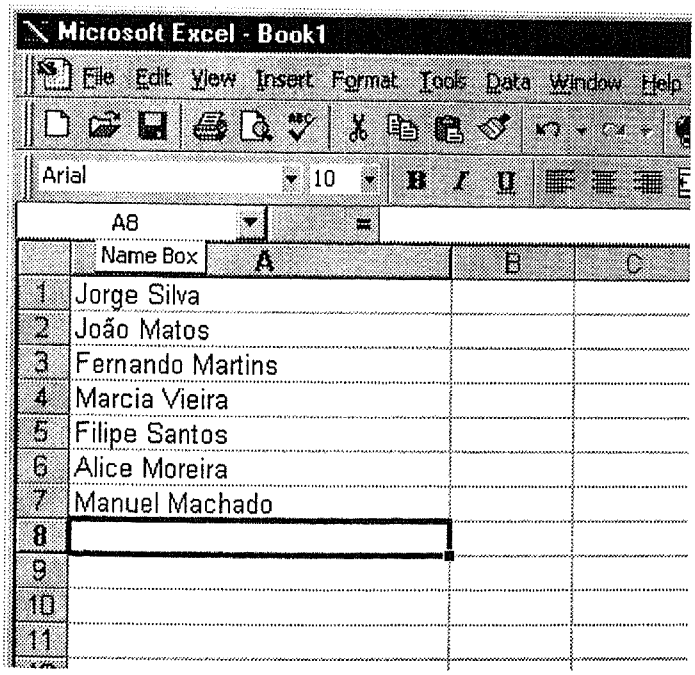

Figura IV-29– Resultado da utilização no Excel

4.3.3. Conclusão

O exemplo não demonstra as capacidades do ADO, mas sim onde, como, e em que situações o ADO pode e deve ser aplicado.

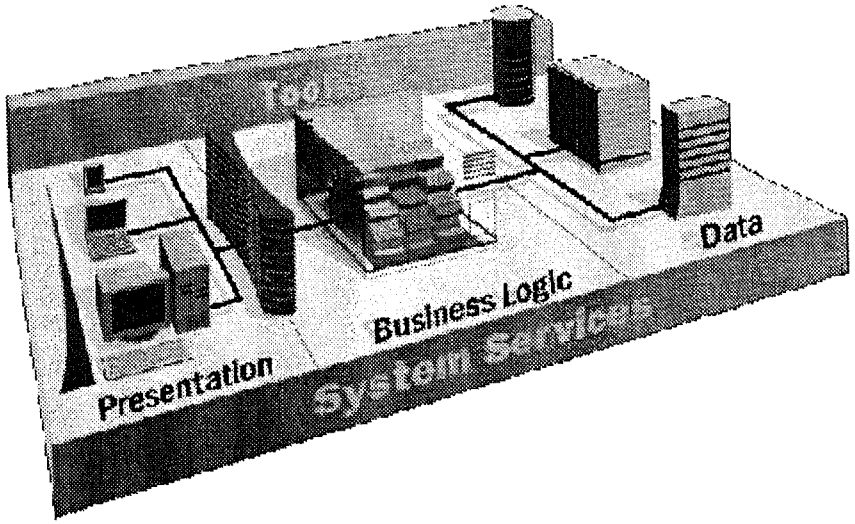

Figura IV-30 – Aplicações por níveis

Nesta demonstração, vimos uma aplicação simples que usa três níveis diferentes. Foram também abordados os benefícios da separação dos níveis de apresentação, lógico e de dados (figura IV-30). Finalmente foi mostrada a capacidade de reutilizar os objectos criados nos níveis lógico e de dados, recorrendo a diferentes maneiras de mostrar os dados ao utilizador.

4.4. ADO e Multimédia

Não existe uma relação directa entre o ADO e a Multimédia já que, como já foi referido, o ADO é uma ferramenta para fazer o interface entre uma determinada aplicação ou uma página Web e uma determinada fonte de informação. No entanto é possível usar o ADO para receber informação da fonte e disponibilisá-la sob as formas já referidas. Na actual versão do ADO (2.1) não foi dada especial atenção à multimédia, mas existe a capacidade de extrair da fonte de informação dados binários para as mais diversas utilizações, tais como imagens, som ou vídeo. As fontes de informação e mais concretamente as bases de dados relacionais, possuem um tipo de

dados para armazenar informação binária e são esses campos que são usados para armazenar a informação pretendida em formatos conhecidos. A versão 2.5 do ADO já disponibiliza um tipo de objecto para melhorar a manipulação de dados binários.

O exemplo seguinte mostra como é possível extrair uma imagem armazenada numa base de dados Oracle para uma página Web usando o ADO como método de acesso à fonte de informação e o um servidor de páginas Web.

```
<%@ LANGUAGE="VBSCRIPT" %>
<%
Response.Expires = 0 ; Limpar a informação de cabeçalho do HTTP.
Response.Buffer = TRUE
Response.Clear
Response.ContentType = "image/gif" ; Modificar o HTTP Header para
receber uma imagem
Dim strTemp
Set oConn = Server.CreateObject("ADODB.Connection")
oConn.Open "DSN=ORACLE_DB;UID=userid;PWD=password;" ; Estabelecer a
conexão à fonte de dados
sSQL = "Select ID, IMG from IMAGE where ID = 1" ; SQL para
seleccionar qual o campo da tabela IMAGE que contém
Set oRS = Server.CreateObject("ADODB.Recordset")
oRS.Source = sSQL
oRS.ActiveConnection = oConn
oRS.Open
strtemp = oRS("IMG") ; atribuir a imagem
Response.BinaryWrite(strTemp) ; inserir na página a imagem
Response.End
oRS.Close
Set oRS = nothing
oConn.Close
Set oConn = nothing
%>
```

Uma outra aplicação onde pode existir um relacionamento entre o ADO e a multimédia seria um misturador de música. Como é sabido quanto mais próxima forem as batidas das músicas mais fácil se torna a mistura e menos perceptível é para quem as houve. Neste caso o ADO pode ser usado com uma dupla função: ler o

stream de música e ler informação sobre as batidas das músicas. Se no primeiro caso a informação é prioritária, no segundo caso a pesquisa pode ser assíncrona para libertar a aplicação para funções que têm de ser realizadas em tempo real. Nesta aplicação o método de acesso disponibiliza informação para a componente multimédia da aplicação e informação para a componente funcional da aplicação. Este é um exemplo onde o método de acesso está relacionado com a multimédia.

No mercado podem ser encontrados outros exemplos de aplicações multimédia que usem acessos a fontes de dados. Existem leitores de CDs de música que identificam o número de faixas de um CD, o tamanho das músicas e procuram em bases de dados o nome do artista ou grupo e os nomes das músicas. Um sítio onde se pode aceder para fazer este tipo de consulta é http://www.cddb.com.

Existe na Internet um sítio que usa ADO, OLE DB e SQL Server e contém uma componente elevada de multimédia: http://terraserver.microsoft.com. Este sítio, permite ao utilizador visualizar superfícies terrestres com bastante precisão. Todas as imagens estão armazenadas numa base de dados relacional e o utilizador através da Internet pode navegar quase pelo mundo inteiro.

O facto do ADO ser um *ActiveX* permite que este seja utilizado em qualquer aplicação multimédia (que tenha suporte para a funcionalidade *ActiveX*) que pode ser usada localmente, numa Intranet ou através da Internet. Se a aplicação multimédia necessitar de aceder a dados o ADO é um bom mecanismo de acesso.

V. CONCLUSÃO

O trabalho descrito nesta tese versou duas vertentes complementares: a primeira era o estudo de tecnologias de acesso a dados que permitissem o acesso a mais do que um tipo de fonte de dados; a segunda era a construção de um sistema de *Trouble Tickets* que usasse e tirasse partido das novas funcionalidades dessas tecnologias.

O segundo objectivo deste trabalho era este sistema de *Trouble Tickets* que deveria proporcionar uma melhor gestão dos problemas num ambiente colaborativo de trabalho. O sistema obtido como resultado final do trabalho pauta-se por ser um sistema simples de usar e prático e que consegue gerir diferentes tipos de problemas com diferentes grupos de pessoas. Além do objectivo inicial, o sistema contém informação que pode ser usada para eliminar fontes de erros, mesmo antes destes acontecerem. Permite ainda fazer análises do grupo das pessoas mais solicitadas para resolução de problemas, analisar dos grupos de utilizadores que mais usam o sistema e conhecer tempos de resolução de problemas, entre outros. A possibilidade de armazenar áudio na fonte de informação torna a descrição e a interpretação do problema mais fácil, visto que é mais atractivo descrever ou ouvir o problema através da fala além de ser possível transmitir certo tipo de emoções por parte dos utilizadores a quando da descrição do problema.

O primeiro objectivo do trabalho foi o estudo de várias tecnologias de acesso a fontes de informação. Um factor importante para a escolha dessas tecnologias foi a possibilidade de comunicação com um número significativo de fontes de informação disponíveis no mercado, bem como a possibilidade de ser usada por várias linguagens de programação.

Com vista à elaboração do trabalho começou por fazer-se um levantamento das tecnologias mais usadas e efectuar uma comparação entre o ADO e essas tecnologias que de alguma maneira contribuíram para o aparecimento do ADO. Feita uma análise às ferramentas anteriores ao ADO, nomeadamente ao DAO e RDO, concluiu-se que o

ADO seguiu e aproveitou grande parte das melhores funcionalidades das tecnologias anteriores.

Outro factor importante a favor da tecnologia escolhida foi o facto desta pertencer ao *Universal Data Access*. Na tese foi abordada esta situação e pretendeu-se explicar que algumas potencialidades do ADO não resultam directamente da tecnologia mas sim de todos os mecanismos que a suportam.

Uma tecnologia que não poderia deixar de ser referida é o JDBC. Esta tecnologia tem condições para ser a tecnologia do futuro mas, no presente, e tal como foi referido, ainda tem algumas limitações em relação ao ADO. Se o facto de ser multiplataforma confere ao JDBC um estatuto invejável, que dificilmente outra tecnologia poderá beneficiar desta característica. Já existem fabricantes de disponibilizam mecanismos de acesso OLE DB e ADO para plataformas não Microsoft, mas não é, ou ainda não é, tão portável como o JDBC.

Após situar a tecnologia de acesso às diferentes fontes e de a comparar com outras, foi necessário analisar com algum pormenor as principais capacidades de manipulação dos dados através do seu modelo de objectos. Uma característica importante do ADO é a capacidade de manipular a estrutura da fonte, ou seja, a capacidade de alterar ou fazer ajustes à estrutura da fonte de informação através do ADOX. Outra característica bastante inovadora é a capacidade de acesso a fontes de informação multidimensional. É também referida na tese a possibilidade de aceder a fontes de informação multidimensional e destacadas algumas capacidades inovadoras em relação a outros métodos de acesso, onde sobressai a possibilidade de extracção assíncrona de dados e o *Data Shaping*.

Na tese, é descrita a implementação do sistema de *Trouble Tickets*, e é apresentado com particular destaque o modo como é feito o acesso à fonte de informação usando o ADO e como este pode disponibilizar um *stream* de som entre a fonte de dados e a aplicação.

O trabalho realizado permitiu demonstrar como é que esta tecnologia se pode enquadrar na implementação de *software* construído em vários níveis, onde o ADO assume um papel preponderante no acesso aos dados.

Refere-se também a contribuição do ADO para as aplicações multimédia. Sem dúvida que o ADO é uma excelente ferramenta para o acesso a múltiplas fontes de informação nomeadamente a informação multimédia. Com a utilização do OLE DB via ADO é possível:

- Aceder à maioria de fontes de informação com uma única e robusta tecnologia;
- Beneficiar da mais recente técnica de desenvolvimento;
- Minimizar a aprendizagem;
- Reduzir o custo de implementação e manutenção.

A grande desvantagem do ADO é, sem dúvida, a sua implementação fora dos sistemas operativos da Microsoft ser ainda pouco visível, embora muitas empresas de *software* tenham bibliotecas para o fazerem. A grande vantagem do JDBC é a maior desvantagem do ADO.

Na futura versão do ADO (2.5), aparece um novo objecto no modelo de objectos, intitulado *Stream*. Este objecto tem como função manipular um campo de um registo que contenha ao mesmo tempo informação binária e texto, possuindo para esse efeito um conjunto de métodos adequados. Esta funcionalidade pode tornar-se bastante importante para o tratamento de E-Mails ou para o tratamento de páginas Web onde exista no mesmo campo uma mistura de texto e imagens. Este novo objecto ainda em fase de desenvolvimento pode beneficiar, por exemplo, o arquivo e tratamento de páginas Web ou outro tipo de aplicações onde exista o binómio informação binária e texto. Este novo objecto pode apresentar algumas vantagens no acesso a dados nas aplicações multimédia.

A escolha do método de acesso para as aplicações ainda não é muito fácil, não pelas fontes de informação que vão ser usadas, pois existem em número significativo e para diferentes tipos de dados, mas pelos sistemas onde vão ser implementadas as

aplicações cliente e pelas limitações que as próprias fontes têm em relação aos mecanismos de acesso.

Existem também os aspectos comerciais das empresas produtoras de informação. Estas não fornecem todas as funcionalidades aos mecanismos de acesso externos para assim poderem mais facilmente venderem os seus próprios produtos de acesso e, indirectamente, criar dependências dos seus clientes.

Como trabalho futuro seria bastante interessante fazer evoluir a aplicação dos *Troubles Tickes* de forma a que esta pudesse suportar vídeo quando surgisse um novo problema, já que o suporte para gravação e leitura na fonte de informação está implementado. Esta nova funcionalidade além de possibilitarem aos técnicos uma visão (som já é possível) do problema. Podia-se pensar na criação de um pequeno componente que teria como função analisar o som inserido ou o som que provem da imagem para a resolução automática de problemas. Uma nova funcionalidade que poderia ser acrescentada seria a inclusão no sistema das bibliotecas do *NetMeeting* para que quando o técnico tentasse resolver o problema estivesse a interagir com o utilizador que reportou o problema.

VI. REFERÊNCIAS BIBLIOGRÁFICAS

[Brus99] Put OLAP and ADO MD to Work, Andrew J. Brust, VBPJ Agosto 1999, vol. 9 no. 9

[Hamil96A] Section 2.1, "A SQL level API", *JavaSoft JDBC: A Java SQL API*, Graham Hamilton & Rick Cattell, Version 1.01, August 8, 1996.

[Hamil96B] Section 2.2, "SQL Conformance", *JavaSoft JDBC: A Java SQL API*, Graham Hamilton & Rick Cattell, Version 1.01, August 8, 1996.

[Laza98] Microsoft Strategy for Universal Data Access, David Lazar Maio 1998 em: Http://msdn.microsoft.com/library/ backgrnd/ html/ msdn_udastrat.htm (última visita 21/09/1999)

[MacM99] Ease Data Access with ADOX, Jeffry P. McManus, VBPJ, Junho de 1999 vol. 9 no. 7

[Micr97A] DAO Object Model for ODBCDirect Workspaces, Microsoft 1997 em: http://msdn.microsoft.com/library/psdk/daosdk/dadi254l.htm (última visita 21/09/1999)

[Micr97B] DAO Object Model for Microsoft Jet Workspaces, Microsoft 1997 em: http://msdn.microsoft.com/library/psdk/daosdk/dadi252t.htm (última visita 21/09/1999)

[Micr98] Understanding the RDO Object Model, Microsoft 1998 em: http://msdn.microsoft.com/library/devprods/vs6/vstudio/vsentpro/veconunderstandingrdoobjectmodel.htm (última visita 21/09/1999)

[Micr99A] ADOX Object Model, Microsoft 1999 em: http://msdn.microsoft.com/library/psdk/dasdk/adod8rho.htm (última visita 21/09/1999)

[Micr99B] ADO MD Object Model, Microsoft 1999 em: http://msdn.microsoft.com/library/psdk/dasdk/adom2vxo.htm (última visita 21/09/1999)

[Micr99F] ADO Dynamic Properties, Microsoft 1999 em: http://msdn.microsoft.com/library/psdk/daosdk/mdad87qr.htm (última visita 21/09/1999)

[Sieb99] ADO 2.0's new Recordset Feature, Dianne Siebold, VBPJ Janeiro 1999, vol. 9 no. 1

[Will99] Implementing ADO with Various Development Languages Don Willits, Microsoft Data Access Group July 1998 em: http://msdn.microsoft.com/library/techart/msdn_adorosest.htm (última visita 21/09/1999)

VII. BIBLIOGRAFIA

Sítios Internet

Empresa	Links
Microsoft	Http://msdn.microsoft.com http://www.microsoft.com/access/ http://www.microsoft.com/data http://www.microsoft.com/data/ado/default.htm http://www.microsoft.com/data/oledb/ http://www.microsoft.com/ntserver/nts/default.asp http://www.microsoft.com/sna/default.asp http://www.microsoft.com/sql/default.asp http://www.odbcsdk.com/Products/OAOleDbSDK/OpenAccess.asp
15seconds	Http://www.15seconds.com
Geppetto's Workshop	Http://www.geppetto.com/
IBM	Http://www.as400.ibm.com/clientaccess/oledb/
ISG	Http://www.isgsoft.com/
Sagent Technology	Http://www.sagenttech.com/index.html
Object Design, Inc.	Http://www.odi.com/
Metawise	Http://www.metawise.com/
Sequiter	Http://www.sequiter.com/
Simba Technologies	Http://www.simba.com/
X-Tension	Http://www.x-tension.com/
Merante	Http://www.merant.com/
Outros	Http://www.activeserverpages.com http://www.able-consulting.com/tech.htm http://ourworld.compuserve.com/homepages/Ken_North/homepage.htm http://www.itmweb.com/ http://www.devx.com/default.asp http://www.webtechniques.com/archives/1999/08/data/

Livros

SQL Server 7 – Beginner's Guide

Dusan Petkovic

Osborne, 1999

SQL Server 7 – Developer's Guide
Michel Otey, Paul Conte
Osborne, 1999

Hichhikers's Guide to Visual Basic and SQL Server, Sixth Edition
William R. Vaught
Microsoft Press, 1998

Inside COM
Dale Rogerson
Microsoft Press, 1999

Programming Microsoft Interdev 6.0
Nicholas D. Evans, Ken Miller e Ken Spencer
Microsoft Press, 1999

Inside Distributed COM
Guy Eddon; Henry Eddon
Microsoft Press, 1999

ActiveX™ Controls Inside Out, Second Edition
Adam Denning
Microsoft Press, 1999

Revistas

Visual Basic Programmer's Journal de Janeiro , Fevereiro, Março, Abril, Junho, Julho e Agosto de 1999

VIII. ANEXOS

7.1. Anexo A - Empresas fornecedoras de interfaces OLE DB

Tipo de Dados/ Plataforma	Fornecedor OLE DB	Empresa
Unisys® OS 2200 e ClearPath IX Systems	AIS UniAccess	Applied Information Sciences, Inc.
DB2/400 AS/400	Acceler8-DB e DataGate/400	ASNA
DB2/400 (Microsoft Windows NT Server)	HiT OLE DB Server/400 and Developer Edition	HiT Software, Inc.
DB2/400 (Microsoft Windows NT Workstation)	HiT OLE DB/400 and Developer Edition (Developer Edition contains additional toolkits)	HiT Software, Inc.
DB2/400 AS/400	Cliente Windows 95/NT SDK	IBM
Fornecedor OLEDB para: Fontes de dados relacionais- Oracle, Sybase, Informix , SQL Server, DB2, Rdb, Ingres, Red Brick, Non-Stop SQL/MP, Non-Stop SQL/MX Fontes de dados não relacionais- CISAM, RMS, DBMS, MUMPS, Jasmine, Adabas, Enscribe, ficheiros de texto Fontes de dados genericas- ODBC, OLEDB (Intel Win 95, Intel NT, HP Unix, Sun Unix, Data General Unix, IBM RS/6000 Unix, Digital Alpha Unix, Digital Alpha OpenVMS, Digital Alpha NT, Tandem NSK)	ISG Navigator	ISG
Família IBM DB2, Informix, Microsoft SQL Server, OpenIngres, Oracle, Sybase Adaptive Server 11.5 and System 10/11; inclui também suporte para Oracle e DB2	SequeLink OLE DB Edition	Merant

em IBM OS/390		
AS/400 and VSAM (Microsoft Windows NT 4.0 Workstation, Windows NT 4.0 Server or Windows 95)	MetaWise DP	MetaWise
LDAP version 2, NetWare 4, NetWare 3, and Windows NT directories (Intel NT, Alpha NT)	Microsoft Active Directory	Microsoft
HTML, texto, documentos Microsoft Office, SQL (Intel NT, Alpha NT)	Microsoft Index Server	
Fornecedor Microsoft Access (Intel NT, Alpha NT)	Microsoft Jet Database Engine	Microsoft
Fornecedor Microsoft SQL Server (Intel NT, Alpha NT)	Microsoft SQL Server	Microsoft
Fornecedor ODBC (Intel NT, Alpha NT)	Microsoft OLE DB SDK	Microsoft
VSAM e AS/400 (Intel NT, Alpha NT)	Microsoft SNA Server	Microsoft
Bases de dados ObjectStore	ObjectStore Active Toolkit 1.5	Object Design, Inc.
OLAP Microsoft SQL Server, Oracle, Red Brick, e Sybase (Intel NT)	Sagent Data Mart Server	Sagent Technology
XBASE (FoxPro, Clipper and dBASE) (Windows 95, 3.1, NT, CE, DOS, OS/2, Macintosh, e UNIX incluindo as plantaformas Solaris, SunOS, HP/UX, AIX, SCO, Linux)	CodeBase OLE DB Data Provider	Sequiter
OSS systems	Data Explorer	X-Tension

7.2. Anexo B - Empresas fornecedoras de produtos e serviços OLE DB

Tipo de Dados/ Plataforma	Produtos e serviços OLE DB	Empresa
Oracle, Sybase, Informix , SQL Server, DB2, Rdb, Ingres, Red Brick, Non-Stop SQL/MP, Non-Stop SQL/MX, CISAM, RMS, DBMS, MUMPS, Jasmine, Adabas, Enscribe, ficheiros de texto, ODBC data, OLEDB data (Intel Win 95, Intel NT, HP Unix, Sun Unix, Data General Unix, IBM RS/6000 Unix, Digital Alpha Unix, Digital Alpha OpenVMS, Digital Alpha NT, Tandem NSK)	ISG Navigator (Multi-Platform Distributed Query Processor)	ISG
Dados SQL e não SQL MAPI tais como e-mail, Microsoft Exchange, Lotus Notes e Lotus cc:Mail (Windows 95, NT)	DataDirect Reflector (Query Processor)	Merant

7.3. Anexo C - Empresas fornecedoras de ferramentas OLE DB

Tipo de Dados/ Plataforma	Ferramentas OLE DB	Empresa
Toolkit para desenvolvimento em rápido OLE DB	OpenAccess OLE DB SDK	Automation Technology, Inc. (ATI)
Dynamic OLE DB para OLAP MDX (Windows 95, NT)	AntMDX	Geppetto's Workshop
Contém OLE DB SDK para construção nativa em OLE DB e ODBC SDK (Windows NT)	SimbaProvider™ Professional Edition	Simba Technologies
Toolkit para construção OLE DB para OLAP (Windows 95, NT)	SimbaProvider™ for OLAP	Simba Technologies

OLE DB para OLAP MDX (Windows 95, NT)	MDX Parser for OLE DB for OLAP	X-Tension

7.4. Anexo D - Objectos ADO em métodos OLE DB

Objecto ADO		**método OLE DB**
Command object		
	Method	
	Cancel	Nenhum
	CreateParameter	Nenhum
	Execute	**ICommand::Execute**
	Property	
	ActiveConnection	Nenhum
	CommandText	**ICommandText::GetCommandText** e **ICommandText::SetCommandText**
	CommandTimeout	**IcommandProperties::SetProperties** (DBPROP_COMMANDTIMEOUT)
	CommandType	Nenhum
	Name	Nenhum
	Prepared	**ICommandPrepare::Prepare** e **ICommandPrepare::Unprepare**
	State	Nenhum
	Collection	
	Parameters	Get: **IcommandWithParameters::GetParameterInfo** ou DBSCHEMA_PROCEDURE_PARAMETERS schema rowset Set: **IcommandWithParameters::SetParameterInfo**
	Properties	**IDBProperties::GetPropertyInfo**, **IcommandProperties::GetProperties** e **IcommandProperties::SetProperties**
Connection object		
	Method	
	BeginTrans	**ITransactionLocal::StartTransaction**
	Cancel	Nenhum
	Close	Nenhum
	CommitTrans	**ITransaction::Commit**
	Execute	**ICommand::Execute** ou **IOpenRowset::OpenRowset**
	Open	**IDBInitialize::Initialize** e **IDBCreateSession::CreateSession**
	OpenSchema	**IDBSchemaRowset::GetRowset**
	RollBack	**ITransaction::Abort**
	Event	
	BeginTransComplete	Nenhum
	CommitTransComplete	Nenhum
	ConnectComplete	Nenhum

Disconnect	Nenhum
ExecuteComplete	Nenhum
InfoMessage	Nenhum
RollbackTransComplete	Nenhum
WillConnect	Nenhum
WillExecute	Nenhum
Property	
Attributes	**ITransactionLocal::StartTransaction**
CommandTimeout	**IcommandProperties::SetProperties** (DBPROP_COMMANDTIMEOUT)
ConnectionString	Nenhum
ConnectionTimeout	**IDBProperties::SetProperties** (DBPROP_INIT_TIMEOUT)
CursorLocation	Nenhum
DefaultDatabase	**IDBProperties::GetProperties** (DBPROP_CURRENTCATALOG) e **IDBProperties::SetProperties** (DBPROP_CURRENTCATALOG)
IsolationLevel	**ITransactionLocal::StartTransaction**
Mode	**IDBProperties::GetProperties** (DBPROP_INIT_MODE) e **IDBProperties::SetProperties** (DBPROP_INIT_MODE)
Provider	**ISourcesRowset::GetSourcesRowset**
State	Nenhum
Version	Nenhum
Collection	
Errors	**IerrorRecords**
Properties	**IDBProperties::GetPropertyInfo**, **IDBProperties::GetProperties** e **IDBProperties::SetProperties**
Error object	
Method	
Nenhum	Nenhum
Property	
Description	**IErrorRecords::GetErrorInfo**
HelpContext	**IErrorRecords::GetErrorInfo**
HelpFile	**IErrorRecords::GetErrorInfo**
NativeError	**IErrorRecords::GetCustomErrorObject** e **ISQLErrorInfo::GetSQLInfo**
Number	Nenhum
Source	**IErrorRecords::GetErrorInfo**
SQLState	**IErrorRecords::GetCustomErrorObject** e **ISQLErrorInfo::GetSQLInfo**
Collection	
Nenhum	Nenhum
Field object	
Method	
AppendChunk	**ISequentialStream::Write**
GetChunk	IStream::Seek e Istream::Read,

		ILockBytes::ReadAt ou **ISequentialStream::Read**
	Property	
	ActualSize	**IAccessor::CreateAccessor** e **IRowset::GetData**
	Attributes	**IColumnsInfo::GetColumnInfo**
	DataFormat	**IColumnsInfo::GetColumnInfo**
	DefinedSize	**IColumnsInfo::GetColumnInfo**
	Name	**IColumnsInfo::GetColumnInfo**
	NumericScale	**IColumnsInfo::GetColumnInfo**
	OriginalValue	**IRowsetUpdate::GetOriginalData**
	Precision	**IColumnsInfo::GetColumnInfo**
	Type	**IColumnsInfo::GetColumnInfo**
	UnderlyingValue	**IRowsetRefresh::GetLastVisibleData** ou **IRowsetResynch::GetVisibleData**
	Value	**IAccessor::CreateAccessor** e **IRowset::GetData** e **IRowsetChange::SetData**
	Collection	
	Properties	**IDBProperties::GetPropertyInfo**, **IRowsetInfo::GetProperties** e **IRowsetInfo::SetProperties**
Parameter object		
	Method	
	AppendChunk	Nenhum
	Property	
	Attributes	Get: **IcommandWithParameters::GetParameterInfo** ou DBSCHEMA_PROCEDURE_PARAMETERS schema rowset Set: **IcommandWithParameters::SetParameterInfo**
	Direction	Get: **IcommandWithParameters::GetParameterInfo** ou DBSCHEMA_PROCEDURE_PARAMETERS schema rowset Set: **IcommandWithParameters::SetParameterInfo**
	Name	Get: **IcommandWithParameters::GetParameterInfo** ou DBSCHEMA_PROCEDURE_PARAMETERS schema rowset Set: **IcommandWithParameters::SetParameterInfo**
	NumericScale	Get: **IcommandWithParameters::GetParameterInfo** ou DBSCHEMA_PROCEDURE_PARAMETERS schema rowset Set: **ICommandWithParameters::SetParameterInfo**
	Precision	Get: **ICommandWithParameters::GetParameterInf**

		o ou DBSCHEMA_PROCEDURE_PARAMETERS schema rowset Set: **ICommandWithParameters::SetParameterInfo**
	Size	Get: **ICommandWithParameters::GetParameterInf** **o** ou DBSCHEMA_PROCEDURE_PARAMETERS schema rowset Set: **ICommandWithParameters::SetParameterInfo**
	Type	Get: **ICommandWithParameters::GetParameterInf** **o** ou DBSCHEMA_PROCEDURE_PARAMETERS schema rowset Set: **ICommandWithParameters::SetParameterInfo**
	Value	**IAccessor::CreateAccessor e** **ICommand::Execute**
Collection		
	Properties	Nenhum
Record object		
Method		
	Cancel	Nenhum
	Close	Nenhum
	CopyRecord	**IScopedOperations::Copy**
	DeleteRecord	**IScopedOperations:Delete**
	GetChildren	**IBindResource::Bind**
	MoveRecord	**IScopedOperations::Move**
	Open	**IBindResource::Bind**
Property		
	ActiveConnection	Nenhum
	Mode	Nenhum
	ParentURL	**IRow::GetColumns**
	RecordType	Nenhum
	Source	Nenhum
	State	**IDBAsynchStatus::GetStatus**
Collection		
	Fields	Nenhum
	Properties	Nenhum
Recordset object		
Method		
	AddNew	**IRowsetChange::InsertRow**
	Cancel	Nenhum
	CancelBatch	**IRowsetUpdate::Undo**
	CancelUpdate	Nenhum
	Clone	**IRowsetLocate**
	Close	**IAccessor::ReleaseAccessor,** **IRowset::ReleaseRows**

	CompareBookmarks	Nenhum
	Delete	**IRowsetChange::DeleteRows**
	Find	Nenhum
	GetRows	**IAccessor::CreateAccessor**, **IRowsetLocate::GetRowsAt**, **IRowset::GetNextRows**, e **IRowset::GetData**
	GetString	Nenhum
	Move	**IRowsetLocate::GetRowsAt** ou **IRowset::GetNextRows**
	MoveFirst	**IRowsetLocate::GetRowsAt** ou **IRowset::RestartPosition**
	MoveLast	**IRowsetLocate::GetRowsAt**
	MoveNext	**IRowsetLocate::GetRowsAt** ou **IRowset::GetNextRows**
	MovePrevious	**IRowsetLocate::GetRowsAt** ou **IRowset::GetNextRows**
	NextRecordset	**IMultipleResults::GetResult**
	Open	**IOpenRowset::OpenRowset** ou **ICommand::Execute**
	Requery	**IOpenRowset::OpenRowset** ou **ICommand::Execute**
	Resync	**IRowsetRefresh::RefreshVisibleData**
	Supports	**IRowsetInfo::GetProperties**
	Save	Nenhum
	Seek	Nenhum
	Update	**IRowsetChange::SetData** e/ou **IRowsetUpdate::Update**
	UpdateBatch	**IRowsetUpdate::Update**
Event		
	EndOfRecordset	Nenhum
	FetchComplete	Nenhum
	FetchProgress	Nenhum
	FieldChangeComplete	Nenhum
	MoveComplete	Nenhum
	RecordChangeComplete	Nenhum
	RecordsetChangeComplete	Nenhum
	WillChangeField	Nenhum
	WillChangeRecord	Nenhum
	WillChangeRecordset	Nenhum
	WillMove	Nenhum
Property		
	AbsolutePage	Get: **IRowsetScroll::GetApproximatePosition** Set: **IRowsetScroll::GetRowsAtRatio** ou **IRowsetLocate::GetRowsAt**
	AbsolutePosition	Get: **IRowsetScroll::GetApproximatePosition** Set: **IRowsetScroll::GetRowsAtRatio** ou **IRowsetLocate::GetRowsAt**
	ActiveCommand	Nenhum
	ActiveConnection	**IDBInitialize::Initialize** e **IDBCreateSession::CreateSession**
	BOF	Nenhum
	Bookmark	**IAccessor::CreateAccessor** e

		IRowsetLocate::GetRowsAt
	CacheSize	*cRows* argument to **IrowsetLocate::GetRowsAt** ou **IRowset::GetNextRows**
	CursorLocation	Nenhum
	CursorType	**ICommandProperties::SetProperties**
	DataMember	Nenhum
	DataSource	Nenhum
	EditMode	**IRowsetUpdate::GetRowsStatus**
	EOF	Nenhum
	Filter	**IRowsetUpdate::GetPendingRows** e **IRowsetLocate::GetRowsByBookmark** OR **IRowsetView::CreateView** e **IViewChapter::OpenViewChapter** e **IViewFilter::SetFilter**
	Index	Nenhum
	LockType	**ICommandProperties::SetProperties**
	MarshalOptions	Nenhum
	MaxRecords	**IOpenRowset::OpenRowset** (DBPROP_MAXROWS) ou **ICommandProperties::SetProperties** (DBPROP_MAXROWS)
	PageCount	**IRowsetScroll::GetApproximatePosition**
	PageSize	Usa-se posições absulotas PageCount e AbsolutePage
	RecordCount	**IRowsetScroll::GetApproximatePosition**
	Sort	Nenhum
	Source	Nenhum
	State	Nenhum
	Status	**IRowsetUpdate::GetRowStatus**
	StayInSync	Nenhum
	Collection	
	Fields	**IColumnsInfo::GetColumnInfo**
	Properties	**IDBProperties::GetPropertyInfo**, **IRowsetInfo::GetProperties** e **IRowsetInfo::SetProperties**
Stream object		
	Method	
	Cancel	Nenhum
	Close	Nenhum
	CopyTo	IStream::CopyTo
	Flush	Nenhum
	LoadFromFile	Nenhum
	Open	**IBindResource::Bind**
	Read	IStream::Stat, IStream::Read
	ReadText	Nenhum
	SaveToFile	Nenhum
	SetEOS	IStream::Seek, IStream::SetSize
	SkipLine	IStream::Seek, IStream::Read
	Write	IStream::Write
	WriteText	Nenhum

	Property	
	Charset	Nenhum
	EOS	IStream::Stat, IStream::Seek
	LineSeparator	Nenhum
	Mode	Nenhum
	Position	IStream::Seek
	Size	IStream::SetSize
	State	**IDBAsynchStatus::GetStatus**
	Type	Nenhum

7.5. Anexo E - Colecções ADO em métodos OLE DB

ADO collection		**OLE DB method**
Errors collection		
	Method	
	Clear	Nenhum
	Refresh	Nenhum
	Property	
	Count	Nenhum
	Item	Nenhum
Fields collection		
	Method	
	Append	Nenhum
	Delete	Nenhum
	Refresh	Nenhum
	Update	**IRowSchemaChange::AddColumns** e **IRowSchemaChange::DeleteColumns**
	Property	
	Count	Nenhum
	Item	**IRow::Open**, **IRowChange::SetColumns**, **IRow::GetColumns**, **IColumnsInfo::GetColumnInfo**
Parameters collection		
	Method	
	Append	Nenhum
	Delete	Nenhum
	Refresh	Nenhum
	Property	
	Count	Nenhum
	Item	Nenhum
Properties collection		
	Method	
	Refresh	Nenhum
	Property	
	Count	Nenhum