Sistema de Gestão de Acesso Remoto a Bases de Dados em Intranet

Ricardo Murta, Miguel Sardinha, João Paulo Firmeza, Fernando Ramos

***Resumo*- Neste artigo é apresentado e descrito um sistema de gestão de acesso remoto a Bases de Dados em intranet.**

Este sistema baseia-se num editor visual de formulários, que posteriormente são utilizados no acesso a Bases de Dados via *Web*.

***Abstract*- This paper presents a management system for remote access to databases in Intranet. This system is a visual editor of forms, that are used to search database documents through the Web.**

I. INTRODUÇÃO [*]

A utilização de tecnologias *Internet* não tem de estar limitada a uma plataforma de informação mundial. Cada vez mais as organizações estão a chegar à conclusão de que algumas tecnologias que podem ser utilizadas com a *Internet* também o podem ser para criar um poderoso sistema empresarial de informação popularmente designado como *Intranet*.

Uma *Intrane*t pode ser imaginada como qualquer esquema de rede que utilize serviços normais TCP/IP dentro de uma rede de empresas.

As *Intranets* que estão a surgir actualmente consistem geralmente em servidores *Web* empresariais disponibilizados aos trabalhadores da empresa ao longo de uma LAN ou através de um acesso comutado privado. Através de ligações a Bases de Dados empresariais, servidores de ficheiros e repositórios de documentos, os servidores *Web* disponibilizam diversas formas de informação aos trabalhadores da empresa via um simples *front-end*: o familiar *browser Web*.

Os trabalhadores da empresa utilizam, assim, os seus *browsers Web* para aceder a uma série de páginas *Web* empresariais com ligações para documentos da empresa e dados escritos em HTML.

Com todas estas potencialidades surge a necessidade de criar uma ferramenta visual que permita a criação rápida e fácil de aplicações de acesso a Bases de Dados, baseadas em WWW para uso interno em organizações.

II. DESCRIÇÃO DO SISTEMA

O "Sistema de Gestão de Acesso Remoto a Bases de Dados em *Intranet*" (fig. 1), baseia-se num editor visual de formulários, que posteriormente são utilizados no acesso de Bases de Dados via Web. A edição dos *forms* pode ser realizada em qualquer máquina ligada a uma rede que utilize os serviços normais TCP/IP. Posteriormente são enviados através de Sockets para o servidor onde são colocados num directório pré-definido.

Assim, sempre que um utilizador solicita o URL de um formulário, através de um *browse*r, o servidor Web passa a solicitação para a ISAPI que executa o código C++ e gera uma página HTML dinâmica, remetendo-a para o servidor *Web*, que por sua vez a encaminha para o *browser*. O utilizador do sistema pode, deste modo, interagir com a Base de Dados, consultando os *forms* que lhe foram disponibilizados.

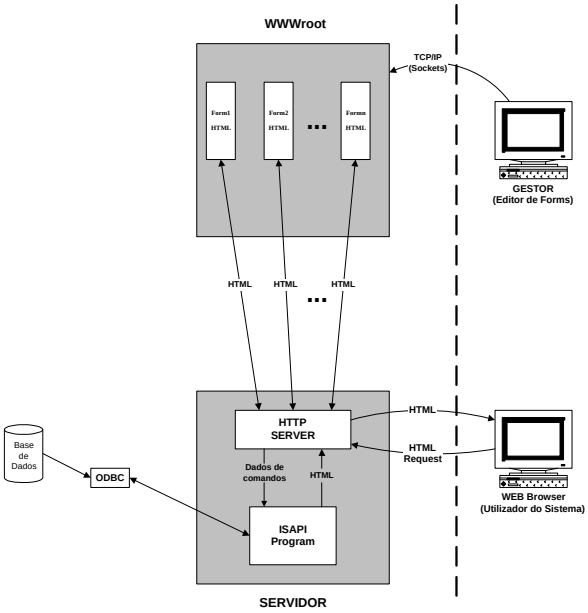

Fig. 1 - Arquitectura do Sistema

III. TECNOLOGIAS UTILIZADAS NA IMPLEMENTAÇÃO DO EDITOR

No desenvolvimento desta aplicação foram estudadas várias tecnologias tendo-se optado pela utilização das seguintes:

- Containers, ActiveX Control Container.
- ActiveX Controls.
- Sockets.

[*] Trabalho efectuado no âmbito da disciplina Projecto

A.. Containers, ActiveX Control Container

Uma aplicação *Container* é uma aplicação que pode incorporar objectos ligados ou embutidos nos seus próprios documentos. Os documentos gerados pela aplicação *Container* devem estar aptos a conter e editar componentes OLE, bem como os dados criados pela própria aplicação. A aplicação *Container* deve permitir aos utilizadores inserir novos objectos ou editar os existentes.

Um ActiveX Control Container, como o próprio nome indica, é um container que suporta controlos ActiveX e pode incorporá-los nas suas próprias janelas ou caixas de diálogo.

B. ActiveX Controls

Os *ActiveX Controls* são versões actualizadas dos *OLE Controls*, podendo as suas funcionalidades fazer parte das páginas *Web*. Trata-se pois de componentes de *software* reutilizável desenvolvidos para satisfazerem várias necessidades, tais como acesso a Bases de Dados, monitorização de dados ou gráficos. Para além da sua portabilidade, os *ActiveX Controls* suportam capacidades que antes não estavam disponíveis em controlos comuns, tais como a compatibilidade com *ActiveX Containers*.

Em acréscimo, estes controlos suportam *ActiveX Automation*, que permite ao controlo expor as suas propriedades bem como um conjunto de métodos que podem ser chamados pelo utilizador do controlo.

B1. Componentes básicos

Os *ActiveX Controls* utilizam vários elementos que interagem eficientemente com o Control Container e o utilizador, tais como o envio de mensagens, e exposição de métodos e propriedades. O primeiro elemento reside na capacidade de enviar mensagens, chamados eventos, ao *Control Container*, e é suportada quando certas condições são satisfeitas. Estes eventos são utilizados para notificar o *Control Container* quando algo de importante ocorre no controlo. Pode-se enviar informação adicional ao *Control Container* acerca de um evento, atribuindo para o efeito parâmetros a esse mesmo evento.

Outro elemento é a capacidade de expor um conjunto de funções, chamados métodos, e atributos, chamados propriedades, ao utilizador do controlo. As propriedades permitem ao *Control Container* ou ao utilizador do controlo, manipula-lo de várias formas, isto é, o utilizador pode modificar a aparência do controlo, alterar certos valores do controlo, ou mesmo aceder a dados mantidos pelo Controlo. Esta interface é determinada pelo projectista do controlo. Assim quando um controlo é utilizado dentro de um *Control Container*, utiliza dois mecanismos para comunicar: expõe propriedades e métodos e dispara eventos. Na fig. 2 está demonstrado como estes dois mecanismos são implementados.

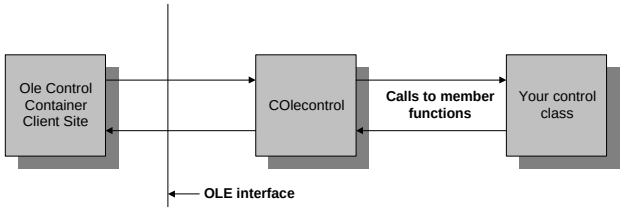

Fig. 2 - Implementação dos métodos e eventos do controlo

B2. Estado activo e inactivo do controlo

Um controlo tem dois estados básicos: activo e inactivo. Quando um controlo está activo, invoca a captura do rato, o *focus* do teclado e outros serviços da janela do seu *container*. Também é possível providenciar interacções com um controlo inactivo, bem como criar controlos que esperam até serem activados para criar uma janela.

Quando um controlo com janela fica activo, fica apto para interagir com o *Control Container*, o utilizador e o Windows. A fig. 3 ilustra a comunicação entre o *ActiveX Control*, o *Control Container* e o sistema operativo.

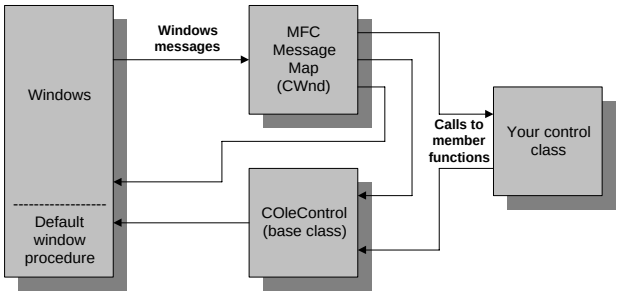

Fig. 3 - Comunicação entre o *ActiveX Control*, o *Control Container* e o sistema operativo.

B3. Controlos ActiveX implementados

Para a aplicação em questão, isto é, a criação de forms, foram desenvolvidos dois controlos, um para edição de texto, **Text Box**, e outro para memorização do campo da Base de Dados, **Edit Box**. O controlo *ActiveX* para a edição de texto, quando activo, permite ao utilizador do editor inserir texto e visualizá-lo no monitor. Para a memorização do campo da Base de Dados foi desenvolvido um controlo que tem por função guardar o campo da Base de Dados seleccionada. Não foram adicionadas quaisquer propriedades a qualquer um dos controlos.

B4. Potencialidades dos controlos ActiveX

Para se ter uma ideia das potencialidades desta tecnologia, achou-se por bem, referir algumas das propriedades que poderão ser adicionadas aos controlos por nós desenvolvidos. Assim para o caso do controlo **TextBox**, propriedades como, a alteração da *Font*, do *Size*, bem como a escolha da cor e estilo com que o texto digitalizado pelo utilizador do Editor é visualizado no *container*, podem, de uma forma não muito complicada, ser acrescidas a este controlo. Para o caso do controlo ActiveX **EditBox**, a alteração da aparência deste, em termos de relevo poderá ser adicionada.

C. Sockets

Um Socket é o ponto terminal de uma comunicação, isto é, um objecto através do qual uma aplicação *Windows Sockets* envia e recebe pacotes de dados através duma rede. Um socket tem um tipo e está associado a um processo que está a correr, e pode ter um nome. Correntemente, os sockets só trocam dados com outros sockets no mesmo "domínio de comunicação" que usa o *Internet Protocol*.

C1. Modelo Cliente/Servidor

O modelo mais utilizado na construção de aplicações distribuídas é o modelo Cliente/Servidor. Neste modelo as aplicações Cliente solicitam serviços a uma aplicação Servidor. Isto implica uma assimetria na comunicação estabelecida entre o cliente e o servidor. Estes requerem um conjunto de convenções bem conhecidas antes do serviço ser enviado (e aceite). Este conjunto de convenções incluem um protocolo que deve ser implementado nos pontos terminais de uma conexão. Dependendo da situação, o protocolo pode ser simétrico ou assimétrico. Num protocolo simétrico, cada um dos lados pode funcionar segundo as regras master ou slave. Num protocolo assimétrico um dos lados é imutavelmente reconhecido como o master, sendo o outro lado o slave.

Não importa qual dos dois protocolos específicos é usado na obtenção de um serviço. Quando se acede a um serviço existe um "processo cliente" e um "processo servidor". Uma aplicação servidor normalmente está escrita num endereço pré-definido por um pedido de serviço. Isto é, um "processo servidor" permanece adormecido até uma conexão ser requerida por uma ligação cliente ao endereço servidor. Ao mesmo tempo o "processo servidor" acorda e serve o cliente, executando os pedidos requeridos pelo cliente.

IV. TECNOLOGIAS UTILIZADAS NA IMPLEMENTAÇÃO DA ISAPI

Uma ISAPI é uma DLL (Dynamic Link Library) que pode ser carregada e chamada por um servidor HTTP (Hipertext Transfer Protocol). Estas aplicações, também chamadas ISA (Internet Server Applications), são invocadas através de um *browser*, inserindo um URL (Universal Resource Locator), como por exemplo http://servidor/Aplic.dll.

A DLL corre no servidor e envia uma página HTML ao *browser* como resposta. Neste caso o URL identifica uma file que contém um programa em vez de código HTML, sendo este programa executado quando o utilizador activa o *link* contido no URL.

O diagrama da fig. 4 ilustra esta questão:

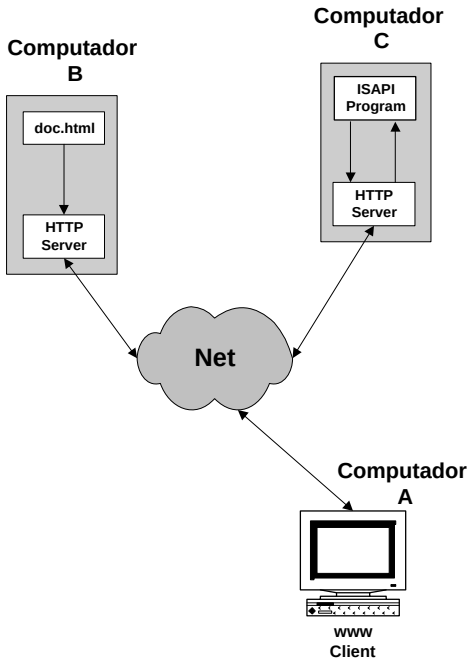

Fig. 4 - Exemplo de utilização de uma ISAPI

Neste diagrama um cliente utilizando um *browser* no Computador A, interage com dois servidores HTTP, um no Computador B e outro no Computador C.

O cliente carrega uma página HTML, doc.html, situado no servidor HTTP que corre no Computador B. Esta página contém um *link* para uma DLL, situada no Computador C. Esta DLL contém um programa que é executado quando o utilizador activa o *link.*

Quando o programa corre, cria um documento HTML, e envia-o para o cliente que o monitoriza no *browser.*

A. Utilização de ISAPI em detrimento de CGI

As ISAPI providenciam uma alternativa à utilização de CGI (Common Gateway Interface) e correm no mesmo espaço de endereço. As ISAs (Internet Server Applications) correm no mesmo espaço de endereços que o servidor HTTP e têm acesso a todos os recursos disponíveis no servidor HTTP. As ISAs têm a vantagem de não requererem a criação de processos adicionais e de não consumirem tempo de comunicação entre processos. As ISAs podem ser descarregadas se a memória for necessária para correr outros processos e permitem a utilização de múltiplos comandos só numa DLL.

Cada pedido CGI lança um processo novo, e cada pedido diferente está contido no seu próprio ficheiro executável, sendo carregado e descarregado em cada pedido.

B. Utilização das ISAPI na criação de páginas Web dinâmicas.

Para a criação de páginas Web dinâmicas, isto é, páginas que interagem com Bases de Dados, é necessário conectar a própria Base de Dados à página Web. A criação de uma ligação física directa para a transferência dos dados entre a *Internet* e as Bases de Dados não é só difícil como leva a riscos de segurança.

A tecnologia utilizada foi o **Microsoft Internet Information Server (IIS)** que corre no sistema operativo **Microsoft Windows NT Server**. Para as Bases de Dados utilizou-se **Microsoft Access**, sendo o acesso feito através do **Open Database Connectivity (ODBC)**. Este providencia uma ligação abstracta entre as Bases de Dados e o *software* que requer esta ligação. Desde que exista um *driver* **ODBC** disponível no **DBMS (Database Management System)**a ligação pode ser feita utilizando o **IIS**.

C. Open Database Connectivity (ODBC)

As classes de acesso às Bases de Dados são implementadas com ODBC, as quais usam SQL (Structured Query Language) para realizar chamadas dinâmicas a dados de uma Base de Dados . O ODBC usa o SQL para comunicar com a *data source* através dos drivers ODBC. Estes drivers interpretam o SQL e traduzem-no, se necessário, para uso num formato de Base de Dados particular, como por exemplo o Microsoft Access.

C1. Data Source

Uma ODBC data source é um conjunto específico de dados; a informação requerida para aceder a esses dados, e a sua localização são descritas usando o DSN (Data Source Name), nome que as aplicações usam para solicitar a conexão à Data Source .

C2. Structured Query Language

Structured Query Language é uma forma de comunicar com uma Base de Dados relacional que permite definir, enviar *queries*, modificar e controlar dados. O uso da sintaxe SQL permite desenvolver uma forma de extrair *records* de acordo com o critério definido.

V. IMPLEMENTAÇÃO DO EDITOR

Para a implementação deste Editor, tiveram-se em conta vários aspectos de ordem prática, nomeadamente no que respeita à facilidade de utilização. Assim, teve-se como prioridade de implementação, a criação de um editor, todo ele orientado por objectos, permitindo deste modo ao utilizador do mesmo, a criação fácil e rápida de *Forms* , não requerendo quaisquer conhecimentos de programação. Quando se refere à orientação por objectos, centraliza-se esta questão, na simplicidade de criação e inserção de objectos no *Form* (isto é, a *View* do Editor), na fácil deslocação para qualquer parte do *Form*, bem como na sua alteração em termos de tamanho, selecção e no apagamento. A tecnologia utilizada para o efeito, como já foi referido anteriormente, é a **OLE** (Object Linking and Embedding). Recorrendo às classes das MFCs, que estão disponíveis para lidar com esta tecnologia, e utilizando as facilidades concedidas pelo *AppWizard* da Microsoft desenvolveu-se o Editor. Para dar resposta a estas questões, foi necessário adicionar várias funcionalidades, nomeadamente a possibilidade de lidar com vários objectos embutidos na aplicação, bem como a selecção, a activação, e a alteração das dimensões do objecto. Na fig. 5 está representado esse mesmo editor com o exemplo de um *form* referente à Base de Dados seleccionada.

Fig. 5 - Editor de *Forms*

A. Criação da página HTML a ser submetida para o servidor.

Quando no Editor, se "clicka" em **Creat Form** , o Editor gera uma página HTML, colocando-a por defeito, num directório pré-definido. Esta página que pode posteriormente ser enviada para o servidor, contém toda a informação necessária para que a *dll* que a processa, possa gerar o *Form* que é enviado para um *browser*. Assim, o Editor terá que ter a capacidade de interpretar a *View* do *container*, isto é, saber quantos objectos estão inseridos, quais os que representam texto, quais os que representam as variáveis dos campos das Bases de Dados, identificar que texto está associado a cada variável campo, bem como saber que Base de Dados o utilizador seleccionou. Com esta informação, é gerada uma página HTML, isto é, é criado código HTML de uma forma dinâmica, reflectindo o estado actual da *View* do *container*.

B. Submissão de Forms para o servidor

Para enviar um documento de um computador para outro, recorre-se, como já referido anteriormente, a *Sockets*. Para esta aplicação, é necessária uma estrutura , que permita do lado do cliente (máquina onde está a correr o Editor), aceder a um determinado directório, seleccionar um documento (*Form* elaborado pelo gestor), e submetê-lo para o servidor. Do lado do servidor, por seu turno, é necessário uma aplicação com uma estrutura idêntica, que esteja à "escuta", por ordem do utilizador, e com a tarefa de canalizar o documento para um directório específico, nomeadamente onde são inseridas as páginas com extensão html.

Foram, deste modo, adicionadas à aplicação três novas classes, a saber **CListenSocket**, **CMySocket** e **CSocketDlg**. Foram utilizadas as funções de notificação, pertencentes às classes de *Sockets*, de modo a que seja possível saber, aquando do estabelecimento de uma conexão, se esta foi feita com sucesso ou não.

VI. IMPLEMENTAÇÃO DOS CONTROLOS *ACTIVEX*

No que respeita ao desenvolvimento destes controlos, recorreu-se também aqui, às facilidades do *AppWizard*, que é oferecido pela **Microsoft**. Na criação de *Forms*, utilizam-se basicamente *editboxs* para conter os campos das Bases de Dados, e um editor de texto, para digitalizar o texto que acompanha estas mesmas *editboxs*. Foram então desenvolvidos dois controlos *ActiveX*, para dar resposta a estas necessidades. Para a edição de texto, era necessário criar um controlo, **TextBox**, que permitisse digitalizar texto, memorizá-lo e visualizá-lo, mesmo estando este no seu estado inactivo. Para tal foram adicionadas algumas propriedades, utilizando as funções **GetText()** e **SetText()**, permitindo deste modo memorizar o texto inserido no controlo ActiveX pelo utilizador do sistema. Para a visualização do texto, contido no controlo ActiveX, no *container* do Editor, recorreu-se à função **InternalGetText()**, que como é perceptível, obtém o texto interno do controlo, sendo a tarefa de o tornar visível levada a cabo pela função **DrawText().**

No desenvolvimento do controlo **EditBox** teve-se em conta , que este teria um aspecto de uma *editbox*, quando estivesse no seu estado inactivo, e tal como o controlo para a edição de texto, deveria memorizar o texto lá inserido. No que respeita à memorização, o processo é idêntico ao do controlo **TextBox**. Já para o caso do seu aspecto, recorreu-se à função **FillRect()**, para modificar cor da área que ocupa, é utilizada a função **DoSuperclassPaint()**, que confere ao controlo o aspecto que se deseja no seu estado inactivo.

VII. IMPLEMENTAÇÃO DA ISAPI

Para a implementação deste sistema, desenvolveram-se três *DLLs* (**Form.dll**, **User.dll** e a **Data.dll**). Para restringir o acesso ao sistema, a um número de utilizadores (por exemplo, só aos funcionários de uma determinada empresa), desenvolveu-se a **User.dll**. Esta comparara os dados inseridos pelo utilizador com os dados contidos numa Base de Dados, gerando uma página HTML no caso de os dados fazerem parte desta Base de Dados, ou enviando uma mensagem de erro caso contrário.

A **Form.dll** tem por função, actualizar os *links* de todos os *Forms* submetidos para o servidor por parte do gestor do sistema, isto é, cria uma página HTML com todos os nomes dos *Forms* bem como os *links* aos mesmos, permitindo deste modo ao utilizador do sistema, através de um simples *click* aceder ao *Form* criado pelo gestor. Para a reprodução dos *Forms*, com os dados já contidos nas *EditBox*, em páginas HTML, foi desenvolvida uma

terceira *dll*, a **Data.dll**. Esta contém duas funções, tendo cada uma delas uma estrutura de dados associada, referente a cada uma das Bases de Dados utilizada para a realização dos *Forms* . É nesta *dll* que se faz toda a interpretação dos dados que são submetidos para o servidor (a Base de Dados utilizada, os campos utilizados na criação do *Form*, bem como o texto inserido que antecede as *EditBox*)

A. Restrição de acesso ao sistema

Para se poder compreender o mecanismo utilizado na restrição de acesso ao sistema, vamos recorrer à ISAPI desenvolvida para o efeito, **User.dll**. Utilizando o *Internet Explorer* e inserindo o URL **http://jps.inesca.pt/inesca.html**, o utilizador do sistema tem, assim, acesso à primeira página ilustrada na fig.6.

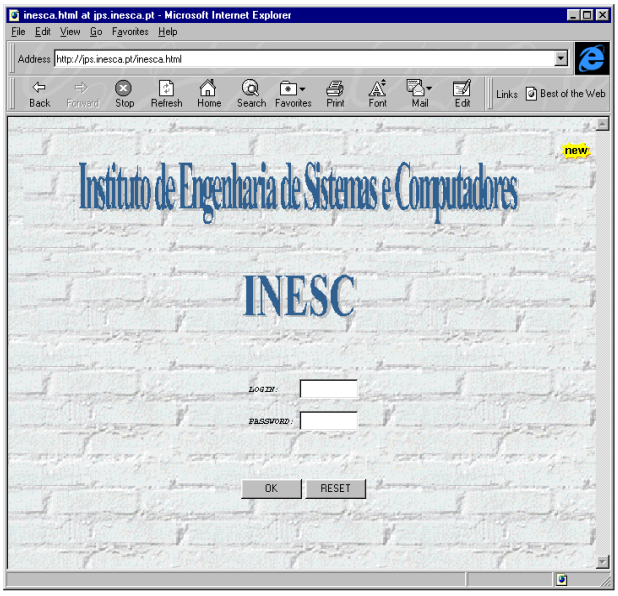

Fig. 6 - Página de acesso ao Sistema

O utilizador insere o *Login* e a *Password* , "clickando" em OK estas *strings* são enviadas para o servidor que executa a função **Default** pertencente à DLL **User**.

Para se poder comparar os dados inseridos pelo utilizador, é necessário aceder à Base de Dados. Para tal utiliza-se uma função que faz um *query* à Base de Dados utilizando para o efeito o **ODBC** e a estrutura de dados (***Recordset***) criada para acolher os dados da Base de Dados.

A primeira tarefa a realizar para obter os dados da Base de Dados, será a de abrir uma conexão à mesma. Foi utilizada para o efeito a função **OpenEx()** pertencente à classe **Cdatabase** que é uma nova adição às MFC 4.2. Tal como a familiar **CDatabase::Open(),** a função **OpenEx()** abre na mesma uma conexão à Base de Dados. A razão pela qual foi utilizada foi a de se ter feito uso do parâmetro **CDatabase::noOdbcDialog**. Se fosse utilizada a função regular **Open(),** e para o caso dos parâmetros inseridos no *Form* não especificarem informação suficiente para realizar a conexão **ODBC** (por exemplo, se deixar algum dos campos vazios), a MFC vai automaticamente abrir uma caixa de dialogo para pedir mais informação ao utilizador. Embora isto seja útil para uma aplicação normal, não o será em aplicações para *Web*.

Uma vez feita a conexão, está-se pronto a utilizar os dados. Se o l*ogin* e *password*, estiverem contidos nesta Base de Dados, então é gerada uma página HTML. Esta página contém informação visível ao utilizador explicando em que consiste este sistema, bem como um "*form action*" (quando o utilizador "clicka" em **Forms,** um *link* é estabelecido para uma segunda *dll*, **Forms.dll**, que por sua vez vai gerar outra página HTML).

Assim, o resultado da execução desta função, no caso de o utilizador do sistema ter permissão para aceder ao mesmo, é uma página HTML (fig. 7).

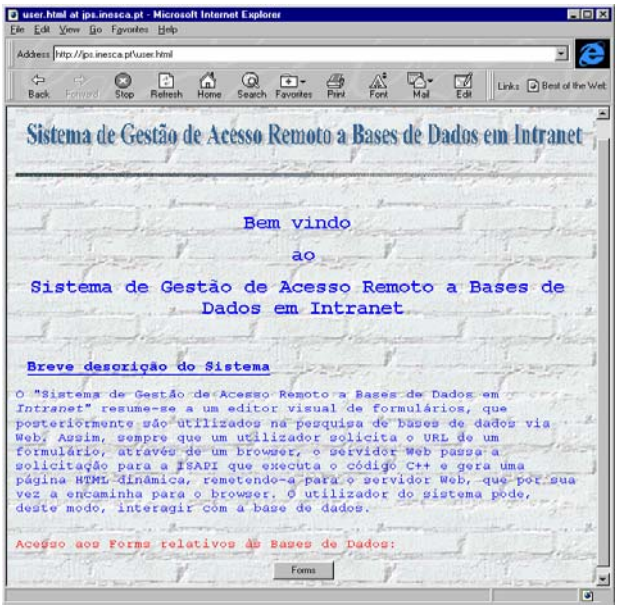

Fig. 7 - Página relativa à descrição do sistema

B. Monitorização de todos os Forms numa página Web

Nesta fase tornou-se clara a necessidade de criar uma página Web que contivesse informação de todos os *Forms* já submetidos para o servidor. Através desta informação (nome dos *Forms*), deveria ser possível ao utilizador do sistema, de uma forma simples, aceder ao respectivo *Form*.

O processo para que tal se torne possível, é a colocação dos nomes dos *Forms* como *hyperlinks* (isto é, através de um "click" no nome do *Form*, o *link* é estabelecido). A *dll* que realiza esta tarefa, **Forms.dll,** como já foi referido anteriormente, contém uma função, **Default()**, onde é feita a pesquisa ao directório em que se encontram os *Forms* e os monitoriza como *hyperlinks* numa página *Web*.

Recorrendo a código HTML, este nome é reproduzido na página HTML que vai ser gerada pela *dll*, como *hyperlink*.

O aspecto visual desta página é ilustrado na fig. 8.

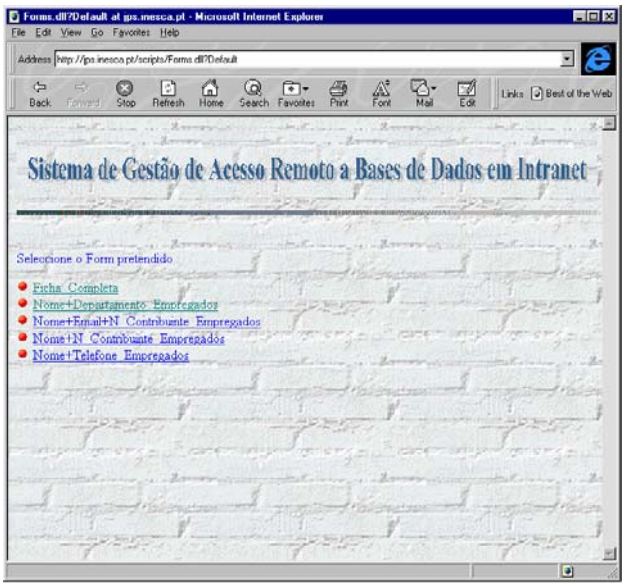

Fig. 8 - Monitorização de todos os *Forms* disponíveis

C. Reprodução dos Forms em páginas Web

Para que todo este sistema fosse funcional, isto é, a ciação de *Forms* ser realizada de uma forma dinâmica, e a reprodução dos mesmos em páginas HTML, não ter que exigir a intervenção do programador, é necessário criar uma função que permita dar resposta a estas mesmas necessidades. Quando se cria um *Form*, este tem que conter informação suficiente a respeito do mesmo, para que desta forma seja reproduzido eficazmente numa página HTML. A *dll* terá de saber interpretar toda esta informação.

Para se ter uma ideia da complexidade de todo este processo, basta referir que a função que se encarrega de dar resposta a estas questões, terá que identificar qual a Base de Dados que o gestor do sistema seleccionou, que campos é que utilizou na criação dos *Forms*, qual o texto inserido pelo utilizador do Editor e a que campo está associado, bem como permitir ao utilizador do *Form*, via *Web*, pesquisar a Base de Dados utilizando para o efeito um motor de pesquisa, que no nosso caso, faz a procura por nome, podendo esta, ser estendida aos restantes campos. Estas foram as questões que tivemos que dar resposta.

Esta *dll*, **Data.dll**, contém duas funções, **Read()** e a **Read1()**, cada uma delas referente a uma das duas Bases de Dados utilizadas para a criação de *Forms*, podendo como é obvio, este número ser aumentado. O princípio de funcionamento destas duas funções é o mesmo, variando a Base de Dados e o número de campos a que acede.

A melhor forma de explicar o princípio de funcionamento destas funções, será a de recorrer a um exemplo prático. Assim, quando se cria um *Form* e se submete para o servidor, a página criada tem o aspecto ilustrado na fig. 9.

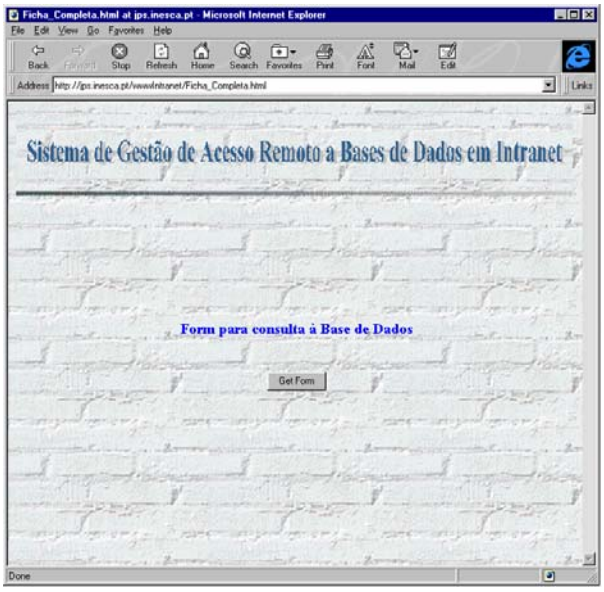

Fig. 9 - Página de acesso à Base de Dados

A que corresponde o código:

```
<HTML>
<BODY BACKGROUND="/bg.gif">
<FORM ACTION="/scripts/data.dll?Read1"
METHOD="POST">
<INPUT TYPE=HIDDEN NAME="texto1" VALUE="Nome: ">
<INPUT TYPE=HIDDEN NAME="campo1" VALUE="name">
<INPUT TYPE=HIDDEN NAME="texto2"
VALUE="Departamento:">
<INPUT TYPE=HIDDEN NAME="campo2"
VALUE="department">
<INPUT TYPE=HIDDEN NAME="texto3" VALUE="Nº de
Contribuinte:">
<INPUT TYPE=HIDDEN NAME="campo3" VALUE="number">
<INPUT TYPE=HIDDEN NAME="texto4" VALUE="Email:">
<INPUT TYPE=HIDDEN NAME="campo4" VALUE="email">
<img src="/Sistema_Gest1.gif"align=botten width=680
height=95><br>
<img src="/grad.gif"align=botten width=680
height=5><br><br><P>
<h3 ALIGN=center><font color="#0000FF">
<B>Form para consulta a Base de
Dados</B></h3><br><br>
<p><INPUT TYPE=HIDDEN NAME="strin"> <P
ALIGN=center>
<INPUT TYPE=SUBMIT VALUE="Get Form">
</BODY>
</HTML>
```

O utilizador *Web* ao "clickar" **Get Form**, vai submeter para a função **Read1()**, as variáveis que contêm o texto inserido pelo gestor (texto1,2,3,4), bem como o nome do campo a que está associado (campo1,2,3,4). Neste caso, o

gestor do sistema utilizou todos os campos da Base de Dados seleccionada, que são quatro. O processo utilizado para gerar a página *Web*, que reflecte este *Form* , resulta de uma série de comparações complexas, que são realizadas com vista a identificar qual o campo com que se deve inicializar a *EditBox,* e associar a esta o texto respectivo. Para o motor de pesquisa é utilizado a variável *strin*, a qual contém o nome que se quer pesquisar, inserido pelo utilizador, com que é feita a procura. A título de exemplo, se o gestor do sistema só tivesse utilizado dois dos campos da Base de Dados para realizar este *Form* , o código produzido pelo Editor poderia ter o seguinte aspecto (caso fossem estes os campos e texto escolhidos) :

```
<HTML>
<BODY BACKGROUND = "/bg.gif">
<FORM ACTION="/scripts/data.dll?Read1"
METHOD="POST">
<INPUT TYPE=HIDDEN NAME="texto1" VALUE="Nome do
Contribuinte : " >
<INPUT TYPE=HIDDEN NAME="campo1" VALUE="name">
<INPUT TYPE=HIDDEN NAME="texto2" VALUE=" Nº de
Contribuinte:">
<INPUT TYPE=HIDDEN NAME="campo2" VALUE=" number ">
<INPUT TYPE=HIDDEN NAME="texto3" VALUE="vazio ">
<INPUT TYPE=HIDDEN NAME="campo3" VALUE="vazio ">
<INPUT TYPE=HIDDEN NAME="texto4" VALUE="vazio">
<INPUT TYPE=HIDDEN NAME="campo4" VALUE="vazio">
<img src="/Sistema_Gest1.gif"align=botten width=680
height=95><br>
<img src="/grad.gif"align=botten width=680
height=5><br><br><p>
<h3 ALIGN=center><font color="#0000FF">
<B>Form para consulta a Base de
Dados</B></h3><br><br>
<p><INPUT TYPE=HIDDEN NAME="strin">
 <P ALIGN=center>
<INPUT TYPE=SUBMIT VALUE="Get Form">
</BODY>
</HTML>
```

Como se pode verificar as variáveis que contêm o texto inserido pelo gestor (texto3,4) e as que contêm o nome dos campos utilizados (campo3,4) são preenchidas com o valor “vazio” . Quando estes valores são submetidos para a função da *dll* , esta vai interpretar que só existem dois campos a criar na página HTML, identificando para o efeito quais e que texto a associar a cada um deles.

Estas funções, **Read()** e **Read1()**, para além de reproduzir os *Forms* e adicionar um motor de pesquisa, criando ainda na página HTML, um *link* para a página que contém os *Forms* que podem ser consultados pelo utilizador do sistema, permitindo deste modo aceder a outros *Forms* que estejam disponíveis.

A fig. 10 mostra a página HTML gerada pela *dll*, no caso da utilização da totalidade dos campos.

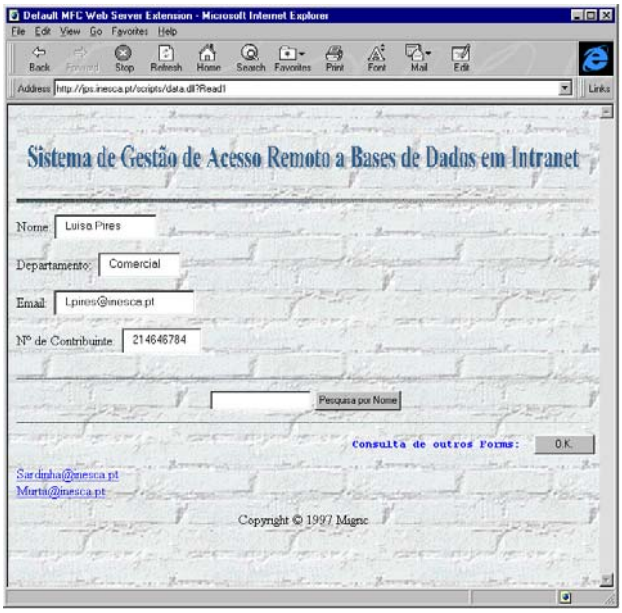

Fig.8 - Reprodução de um *Form* numa página *Web*

VIII. CONCLUSÕES

Foi implementada a estrutura base dum sistema de acesso remoto a Bases de Dados que permite ao utilizador final, a fácil realização de *Forms* e sua utilização em páginas *Web*.

A utilidade deste sistema, numa empresa, torna-se óbvia, tendo em conta que grande parte da informação contida numa Base de Dados, não é necessária nem utilizada nos vários departamentos, que a integram. Assim, fazendo uso deste sistema, é permitido ao gestor do mesmo, direccionar para cada departamento a informação que achar mais relevante, criando *Forms* específicos, para serem utilizados pelos vários trabalhadores que integram estes mesmos departamentos.

REFERÊNCIAS

[1] Ivor Horton, “Beginning Visual C++ 4”, Wrox Press Ltd, 1996.

[2] Ben Ezze, “The Revolutionary Guide to Visual C++”, Birmingham, Wrox Press Ltd, 1994.

[3] Keneth L. Spencer, Upper, “NT Server Management and Control”, Saddler River, Prentice Hall, 1996.

[4] Patrice Bonner, “Network programming with Windows Sockets”, Upper Saddle River (NJ), Prentice Hall, 1996.

[5] Bob Quinn, “Windows Sockets Network Programming”, Reading (Mass), Addison-Wesley, 1996.

[6] Tom Savora, “Using HTML”, Indianapolis (IN) Que, 1995.