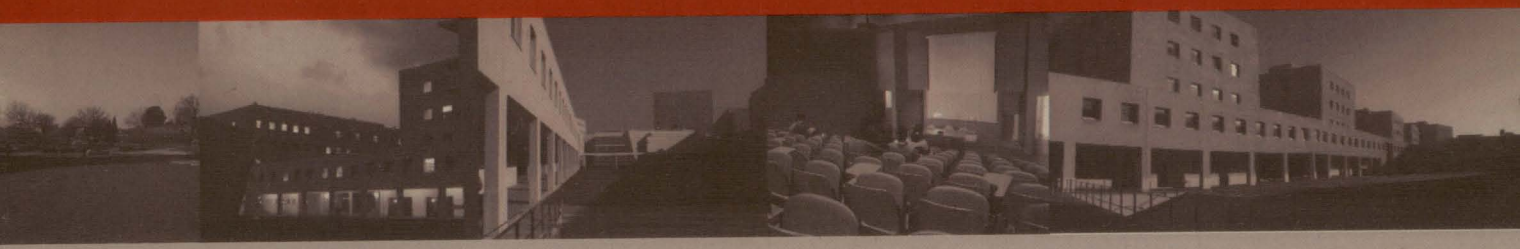
Universidade do Porto
Faculdade de Engenharia
FEUP



Cláudio Manuel Pinto da Silva

Migração de Aplicações para .NET

EGAPI - Equipamentos e Gestão para Aplicações Industriais, Lda.

LEIC
Porto, 2005

Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

Migração de aplicações para .NET na EGAPI – Equipamentos e Gestão para Aplicações Industriais, Lda.

Relatório do Estágio Curricular da LEIC 2004/2005

Cláudio Manuel Pinto da Silva

Orientador na FEUP: Prof. João Carlos Pascoal de Faria

Orientador na EGAPI: Eng. Rodrigo Jorge de Oliveira Maia e Queiroz Machado

Agosto de 2005

Resumo

O trabalho apresentado neste documento surge no âmbito do estágio por mim realizado na empresa EGAPI. Este consistiu na migração de aplicações comerciais já existentes para a tecnologia .NET da Microsoft®.

O primeiro projecto consistiu na migração da aplicação "Wine-Cellar Information System"™ (WIS), uma aplicação de gestão de adegas cooperativas, que se encontra actualmente em pleno funcionamento e que foi desenvolvida com a tecnologia Oracle Forms, para um amiente *Web*.

O segundo projecto consistiu na criação de um protótipo baseado na bilheteira da aplicação "Accessu"™, uma aplicação de controlo de acesso e presenças também desenvolvido em Oracle Forms, utilizando um *Smart Client*.

A tecnologia utilizada na implementação do projecto foi C#, ASP.NET, JavaScript, Oracle e SQL Server.

Agradecimentos

Oportunamente, faço aqui os meus sinceros agradecimentos a todos os que de algum modo contribuíram para a realização deste trabalho, em especial, às pessoas abaixo referenciadas.

Ao Eng. Rodrigo Jorge de Oliveira Maia e Queiroz Machado, meu orientador de estágio na empresa, e Sr. Joaquim Cruz pela oportunidade dada e apoio ao longo do estágio. A todos os restantes elementos da EGAPI, por me terem proporcionado uma excelente integração na empresa, quer ao nível profissional, quer pessoal.

Ao Prof. João Carlos Pascoal de Faria, meu orientador da FEUP, pela disponibilidade, aconselhamento ao longo do estágio e auxílio na construção deste relatório de estágio.

À Sara Ribeiro por todo o carinho, apoio e paciência que sempre demonstrou em todos os momentos.

Finalmente, agradeço à minha família e amigos pelo apoio dado em todas as ocasiões.

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

1 Introdução

Neste capítulo apresenta-se de uma forma sucinta o contexto no qual se desenvolve o projecto, em que consiste, motivação, objectivos e por fim a estrutura deste documento.

1.1 Apresentação da Instituição de Estágio

Nesta secção será apresentada a instituição onde foi realizado o estágio, descrevendo a sua história e o seu estado actual.

1.1.1 História da EGAPI

A criação da EGAPI – Equipamentos e Gestão para Aplicações Industriais, Lda., data do ano 1990, em Braga, com um capital social de 15.000€. A EGAPI surgiu como uma "software-house", cujo principal objectivo era o de fazer desenvolvimento para a área da indústria.

Assim, durante os dois primeiros anos de actividade, a sua equipa dedicou-se ao desenvolvimento e implementação de soluções verticais para áreas como o Controlo de Produção ou Gestão Comercial Integrada. Depressa se tornou numa empresa referência no sector de desenvolvimento de software, pela sua preocupação em dar cobertura às necessidades específicas de cada área de negócio em que intervém.

Com uma filosofia que a orienta para a especificidade das soluções que implementa, cresce de um modo sustentado, ocupando nichos de mercado muito pouco explorados até então.

Nascem assim produtos aplicacionais para a Indústria, sob a marca BDI – base de dados industrial, e para outras áreas, sob a marca EGAPI, sendo a sua dinâmica comercial exercida directamente ou através de uma rede de agentes especializados.

Para além do desenvolvimento aplicacional, cedo foi compreendido o futuro das redes de microcomputadores pelo que se geraram competências nesta área com a instalação de produtos Novell e Microsoft.

É também nesta altura que a actividade de "software-house" é coadjuvada com a de distribuidor de sistemas Bull e Zenith Data Systems.

Passado um par de anos, a vocação para a integração de sistemas, leva a EGAPI a ter um papel relevante no desenvolvimento aplicacional de sistemas intimamente ligados a processos de pesagem industrial.

Assim, em parceria com diversos fabricantes de básculas, dedica-se à ligação e controle aplicacional deste tipo de sistemas. A actividade nesta área obriga, obviamente, à especialização em diversas áreas de negócio, a maior parte das quais ligadas ao sector agro-industrial.

Embora o objectivo principal tenha sido o controle aplicacional integrado dos periféricos envolvidos em operações de pesagem, hoje a EGAPI possui um leque de soluções globais para áreas tão vastas como: Industrias Extractivas, Lagares de Azeite, Adegas ou Matadouros.

Nunca perdendo de vista a cobertura das especificidades funcionais de cada área de negócio, a EGAPI diversificou a criação de aplicativos diferentes, inclusive dentro do mesmo ramo, como por exemplo as Adegas da região do Douro e Alentejo e Matadouros de Bovinos e/ou

Suínos, aplicações essas que elevaram a empresa à qualidade de fornecedora de aplicações a nível nacional.

O ano de 1994 torna-se um marco importante para a EGAPI, ao iniciar a sua relação como parceira da ORACLE Corporation, adquirindo o estatuto de Value Added Reseller (VAR) e posteriormente Independent Software Vendor (ISV).

O desenvolvimento aplicacional sobre produtos do maior construtor mundial de bases de dados era um objectivo há muito ambicionado, e que, a partir dessa altura, irá constituir a principal actividade da "software-house". É assim concretizada a entrada no mercado no que se refere a projectos de média e grande dimensão, com recurso à utilização de Sistemas UNIX, Bases de Dados Relacionais e Ferramentas de 4ª geração.

Sempre ambicionando atingir novos sectores de uma actividade económica de feroz concorrência como é a do desenvolvimento aplicacional, em 1995 a EGAPI, em parceria com a SISCON (Sistemas de Controlo Lda.), desenvolve em ORACLE7/Developer 2000, todo o suporte aplicacional para um sistema global de Gestão e Controle de Acessos a recintos desportivos, parques de exposições, auditórios, ou qualquer actividade negocial cujo principal intuito seja o de Controlo de Acessos.

De então para cá, a EGAPI tem como principal trunfo o desenvolvimento de software, e aposta numa nova área de automação dentro da empresa, o desenvolvimento de hardware específico de apoio às diferentes aplicações que vão sendo criadas. Exemplo disso mesmo, são os produtos desenvolvidos para uma miríade de adegas espalhadas por todo o país, onde para além do sistema de informação propriamente dito, foi desenvolvido todo o hardware de apoio, em postos de auto-serviço, que realizam a recolha automática de dados tão diversos como os dados dos produtores (através da leitura de cartões ópticos), recolha automática do peso, recolha de dados provenientes de refractómetros, etc.

Acreditando que a relação entre empresas pode ser o modo mais eficaz de obter soluções globais e complementares, a EGAPI está actualmente associada da CPC – Companhia Portuguesa de Computadores Is Informática e Sistemas, Lda. tendo essa relação de parceria sido formalizada com a entrada da CPCis para o Pacto Social da EGAPI, com uma participação de 30%.

A CPCis é uma empresa do grupo CPC, o qual constitui hoje uma referência fundamental no sector informático português. VAR da HP, IBM e Digital, para equipamentos de médio e grande porte, assume-se também como um dos mais importantes VAR ORACLE, sendo nesta área que a coadjuvação de esforços entre a CPC-is e a EGAPI adquire maior relevo.

Finalizando, apresenta-se na Figura 1 o volume de vendas desde 1997 até 2001, com indicação do número de colaboradores.

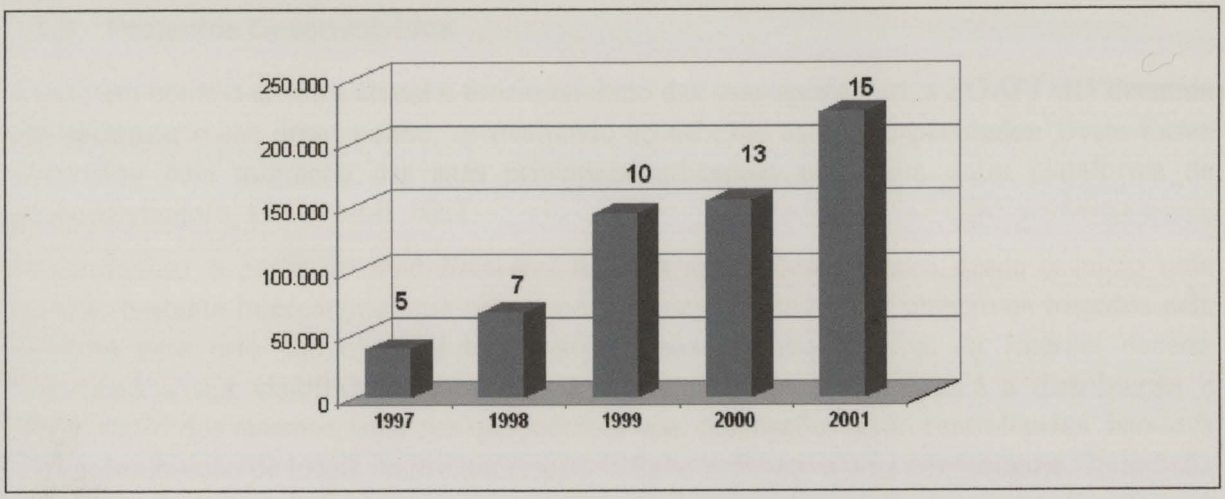


Figura 1 - Volume de Vendas em Euros e Nº de Colaboradores

1.1.2 A EGAPI Actualmente

Actualmente a EGAPI está numa fase de melhoramento e ajustamento dos seus principais produtos (aplicações) às necessidades actuais do mercado. Apesar da grande maioria dos produtos disponibilizados utilizarem tecnologia ORACLE, é aposta da empresa disponibilizar soluções para as áreas de negócio abrangidas por ela baseadas em tecnologias mais flexíveis, que permitam o desenvolvimento em multi-plataforma e uma eficiência superior em termos de desempenho e de *deployment.*

Presentemente, o número de colaboradores é 22, sendo a média de idades muito jovem, mais ou menos 30 anos. A nível interno a EGAPI apresenta o organograma da figura 2.

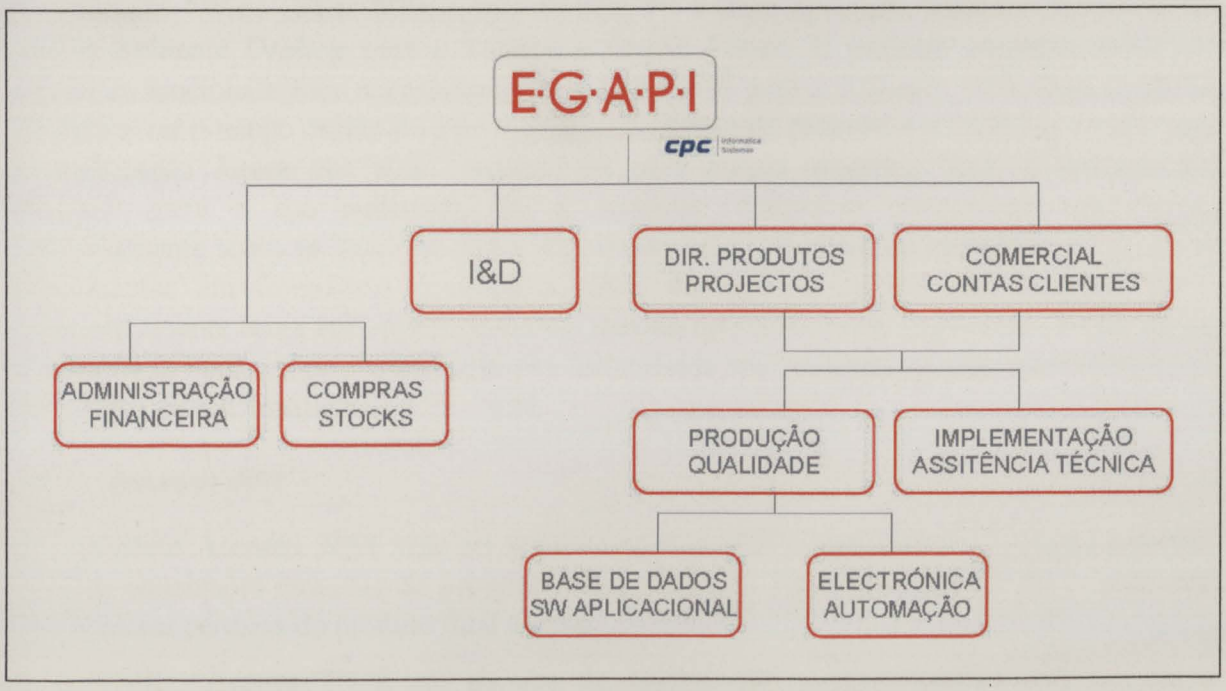


Figura 2 - Organigrama da EGAPI

1.2 Projectos Desenvolvidos

Tendo em conta o alcance actual e funcionamento das suas aplicações, a EGAPI está decidida em optimizar o seu desempenho, aproveitando ao máximo as suas capacidades. Desta forma enveredou pela migração das suas principais aplicações para uma outra plataforma de desenvolvimento, a Microsoft .NET.

Disponibilizar a partir de *Web Browsers* as suas aplicações, pareceu desde o início uma solução bastante interessante, que corresponderia perfeitamente aos objectivos traçados pela empresa para este "novo" produto. O acesso aos produtos através da Internet decerto aumentará a sua visibilidade comercial ao mesmo tempo que facilitará a distribuição e actualização dos mesmos, uma vez que todas as suas instalações serão centralizadas. Isto leva a uma diminuição de locais onde é necessário instalar software e uma consequente diminuição acentuada de custos de licenciamento, também devido à mudança de plataforma Oracle para Microsoft .NET.

Durante a realização do estágio integrei dois projectos que, apesar de envolveram a migração de aplicações de Oracle para .NET, estavam incluídos em áreas de negócio distintas. Em ambos os casos o estágio foi inserido na fase inicial dos projectos ao nível de construção e de testes das novas soluções.

Neste ponto irei descrever de um modo sucinto o âmbito de cada uma das aplicações, sendo que na secção respectiva de cada projecto, o irei detalhar mais pormenorizadamente.

1.2.1 Wine-Cellar IS .NET

O projecto Wine-Cellar IS .NET, foi o primeiro projecto da empresa a enveredar por este caminho.

A aplicação "Wine-Cellar Information System"™ é uma aplicação modular desenvolvida para o ambiente *Desktop* com a tecnologia Oracle Forms. O projecto consistiu então em migrar as funcionalidades suportadas pelo produto WIS para o ambiente *Web*. Com o intuito de minimizar o tempo requerido para o desenvolvimento do projecto e possibilitar ao máximo a reutilização futura dos seus componentes para outros projectos *Web*, a metodologia adoptada para a sua realização foi a seguinte: definiu-se previamente metadados, nomeadamente sobre as bases de dados suportadas pela aplicação, de maneira a ser possível implementar um formulário genérico a partir do qual se estima que se possa gerar automaticamente cerca 80% dos formulários simples que constituem a aplicação. Dessa forma dedicar-se-ia mais tempo à construção dos formulários mais específicos e/ou mais complexos em termos de funcionalidades e de visual.

1.2.2 Accessu .NET

Já o projecto Accessu .NET veio no seguimento do trabalho realizado no projecto anterior. Com as conclusões retiradas do projecto Wine-Cellar IS .NET, foi possível partir para este com maiores certezas do produto final a obter.

A aplicação "Accessu"™ é um sistema de informação e equipamentos para suporte a processos de bilhética, controle de acessos e de presenças, também desenvolvido em Oracle Forms para o ambiente *Desktop*. Neste projecto optou-se por desenvolver Smart Clients, uma tecnologia que permite, da mesma forma que uma página Web, uma fácil distribuição e actualização do produto, aliada a outras facilidades como o facto de poder trabalhar *offline* ou

ter uma interface gráfica superior, uma vez que o seu desenvolvimento é semelhante a uma aplicação Windows standard.

1.3 Organização e Temas Abordados no Presente Relatório

Este relatório está dividido em 4 capítulos.

O presente capítulo é uma pequena introdução, onde são apresentadas algumas ideias e conceitos que servem de base para a apresentação do projecto. São apresentados também os objectivos dos projectos que englobam este relatório.

Nos capítulos 2 e 3 são descritos os dois projectos em que estive envolvido. Cada um está dividido em diversas secções que permitem ter uma visão geral sobre o âmbito de cada projecto e o seu trabalho de desenvolvimento, bem como as previsões de trabalho futuro.

Finalmente, no capítulo 4 estão apresentadas as conclusões retiradas após a conclusão deste estágio.

2 Projecto Wine-Cellar IS .NET

Nesta secção detalha-se o âmbito do projecto Wine-Cellar IS.NET. Será feita inicialmente uma descrição da solução anterior, explicando sucintamente as suas funcionalidades e modo de funcionamento. De seguida apresentarei os requisitos que levaram a criação deste mesmo projecto e as suas condicionantes. Nos cinco pontos seguintes farei uma descrição detalhada de toda a nova solução, passando pela sua arquitectura e processo de desenvolvimento. Finalmente, nos dois últimos pontos descrevo o seu estado actual e perspectivas de trabalho futuro.

2.1 Solução Anterior

O "Wine-Cellar Information System"™ é um sistema aplicacional, modular, de gestão e equipamentos para suporte às indústrias de vitivinicultura, desenvolvido em Oracle Forms para o ambiente Desktop, sendo todos os relatórios e documentação oficial produzidos utilizando Oracle Reports.

Cada módulo da aplicação abrange um processo relevante para esta indústria: Viticultura, Vindima, Vinificação, Produção e Gestão Comercial. Cada um é independente e pode funcionar sem recorrer aos outros. De resto, todos os suportes aplicacionais foram desenvolvidos na perspectiva de sustentar de uma forma integrada todos os processos inerentes à indústria de vitivinicultura. Aliando a essa integração uma elevada parametrização do funcionamento da aplicação conseguiu-se obter um sistema único e perfeitamente integrado, possibilitando o ajustamento do produto ao utilizador final de uma forma simples e eficaz.

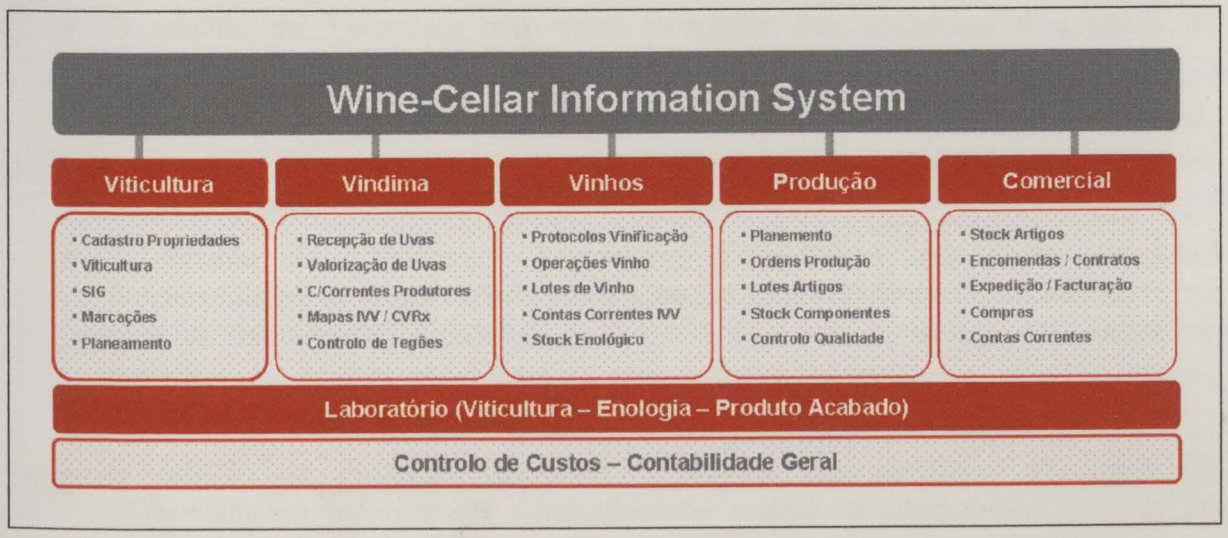


Figura 3 - Áreas da indústria de vitivinicultura suportadas pelo Wine-Cellar IS

O "Wine-Cellar Information System"™ integra à sua volta componentes tão diversas como as provenientes da recepção de vindima com o respectivo controlo de origem, processos/históricos de fermentação, lotes de vinificação, operacionalidade laboratorial sem lotes, etc. Contudo, a quantidade de dados a recolher tornaria o sistema "pesado" em termos

operacionais. Assim, cada uma das componentes é complementada com processos de "automação informatizada".

Este conceito é aplicado sobre os processos de recepção de vindima, vinificações com temperatura controlada, recolha de dados sobre protocolos de vinificação, linhas de engarrafamento e marcação automática de lotes engarrafados e embalados.

Nestes sistemas, são considerados todos os tipos de equipamentos existentes, directa ou indirectamente envolvidos na recolha ou controle de elementos relevantes, tais como: básculas, refractómetros, sondas de temperatura, sistemas de refrigeração, caudalímetros, rotuladoras, etc. Todas os interfaces com estes equipamentos são realizados por uma placa desenvolvida pela EGAPI, utilizando assim protocolos definidos pela empresa, facilitando a integração no sistema de informação.

No conceito de automação informatizada as componentes de automação incluídas no sistema herdam automaticamente pressupostos de parametrização existentes no sistema de informação – por exemplo, protocolos de vinificação, deixando portanto de ser encaradas como simples processos automáticos, sempre dependentes de operadores. Os dados resultantes dos diferentes processos são, naturalmente e automaticamente, inseridos no sistema de informação.

Todo o processo é implementado de forma a automatizar o mais possível todo o fluxo de informação desde a uva plantada numa parcela de uma vinha até ao produto final engarrafado e facturado a um cliente.

2.1.1 Funcionalidades

Passo de seguida a descrever brevemente as funcionalidades existentes em cada um dos módulos integrados no sistema aplicacional WIS:

- O módulo de Viticultura tem como principais funcionalidades o cadastro de propriedades, o processo de viticultura, as marcações e o planeamento de vindimas.
- O módulo de Vindima tem como principais funcionalidades a gestão de produtores, os processos de recepção de uvas, a valorização, as contas correntes dos produtores e o controlo de tegões.
- O módulo de Vinificação, Laboratório, Lotes/Cubas tem como principais funcionalidades os protocolos de vinificação, as fichas de fermentação e análises, os lotes de fermentação, o controlo de fermentações, a movimentação de vinhos, os tratamentos e adições enológicos, os lotes e depósitos, o laboratório e o stock de enológicos.
- O módulo de Produção tem como principais funcionalidades as linhas de engarrafamento, as ordens de engarrafamento, o stock de componentes, os lotes de produto acabado e o controlo de qualidade/custos.
- O módulo de Gestão Comercial tem como principais funcionalidades as encomendas/contratos, o stock de engarrafados, a expedição/facturação e as compras.

Todo este sistema foi desenhado para possibilitar aos seus utilizadores a recepção de castas seleccionadas e devidamente controladas nas vinhas, bem como vinificações planeadas e controladas desde a sua origem até ao produto final.

2.1.2 Limitações

Apesar da qualidade da gestão de informação que este sistema oferece, existem algumas contrariedades na arquitectura actual, das quais enumero as seguintes:

- Ainda que os formulários desenvolvidos em Oracle Forms estejam colocados num servidor único, é necessário instalar em todos os postos de acesso à aplicação as ferramentas de Oracle.
- Numa empresa que tenha diversos locais (p. ex.: uma sede no Porto e diversas quintas dispersas pelo Douro) é necessária uma base de dados em cada local para garantir que, por exemplo, a vindima não pare devido a uma falha de ligação. Isto cria a necessidade de uma actualização diária em todos os locais da empresa dos dados mestres, bem como consolidar os dados operacionais na base de dados central no final de cada dia.

2.2 Requisitos e Condicionantes do projecto

Uma vez que o projecto se centrava na migração de aplicações já existentes e em funcionamento em clientes, todos os requisitos funcionais foram herdados da solução anterior, adicionando, sempre que possível e fosse importante, novas funcionalidades. As funcionalidades já existentes foram também alvo de uma avaliação de forma a introduzir melhoramentos no seu funcionamento.

Já no que se refere aos requisitos não funcionais, que também transitaram da solução já existente, teve que ser dada uma atenção especial. Sendo uma migração para uma nova plataforma, ela só faz sentido se não existir uma degradação dos requisitos não funcionais, e se possível, serem melhorados. Desta forma, a interacção com o utilizador (toda a interface foi elaborada com o intuito de facilitar ao máximo a utilização por parte dos utilizadores, tendo em mente que nem todos os utilizadores têm as mesmas qualificações profissionais) e a performance (o sistema deve ser capaz de dar resposta às acções do utilizador em tempo útil e deve ser capaz de suportar diversos utilizadores em simultâneo) foram os pontos onde houve mais cuidado de garantir que esta nova aplicação será de facto uma evolução da sua predecessora.

Também o tempo de desenvolvimento foi tomado em conta. Para melhorar este ponto, optou-se por criar um gerador automático de formulários que iria permitir a criação de cerca de 80% de todos os formulários da aplicação, diminuindo assim o tempo necessário para a construção de todos os formulários e deixando mais tempo para a criação dos restantes, mais específicos da aplicação.

Existiu também uma preocupação em criar uma solução que seja independente da base de dados, isto é, que tanto possa trabalhar com uma base de dados com tecnologia Oracle ou com outra qualquer, como por exemplo SQL Server. Isto porque, apesar de uma base de dados Oracle ser extremamente fiável e com elevado desempenho, tem também um custo elevado, custo esse que para clientes mais pequenos poderá ser incomportável e possivelmente não necessitará de uma base de dados com o desempenho que a tecnologia Oracle permitem.

Além destes requisitos, as razões apontadas no ponto 1.2 - a visibilidade comercial de um produto com acesso através da Web e a redução de custos de licenciamento, que veio também

na sequência do parágrafo anterior - foram condicionantes importantes no desenvolvimento deste projecto.

2.3 Arquitectura da Nova Solução

Um dos grandes problemas da solução actual tem sido a necessidade de existir uma replicação dos dados em todos os locais da empresa onde o sistema era instalado e uma instalação de ferramentas Oracle em todos os postos de acesso ao mesmo. Tendo isto em mente, criou-se uma arquitectura que permitisse minimizar tal situação.

A disponibilização do acesso à aplicação através de um *Browser Web* foi a solução que sempre se mostrou mais interessante (o acesso através de Terminal Client mostrou-se algo lento e nem sempre mostrou o desempenho desejado). A opção pelo desenvolvimento da nova aplicação em ASP .NET tornou-se assim quase óbvia. O baixo custo de licenciamento, a actual implantação no mercado desta tecnologia da Microsoft e a facilidade de desenvolvimento e de acesso a informação relevante para o mesmo desenvolvimento foram fundamentos bastante fortes para a tomada desta opção.

Com esta solução pode-se facilmente realizar um *deployment* da aplicação em um qualquer cliente, seja de que dimensão for, uma vez que toda a instalação será feita num servidor central, podendo, logo de seguida, qualquer computador aceder à aplicação. Existe ainda possibilidade, para clientes pequenos, da instalação da sua aplicação ser feita em servidores da própria EGAPI, minimizando assim custos, tanto para o cliente como para a EGAPI, e continuando a manter uma qualidade de serviço superior.

No ponto seguinte irei descrever a arquitectura básica da nova solução.

2.3.1 Arquitectura

Como já foi referido antes, a facilidade de acesso à aplicação e a minimização da complexidade do *deployment* e actualização do produto foram pontos importantes no desenvolvimento desta nova solução. Sendo assim, a arquitectura desta nova solução foi criada de forma a garantir estas exigências. Desta forma ficou decidido que ela teria a estrutura apresentada a seguir.

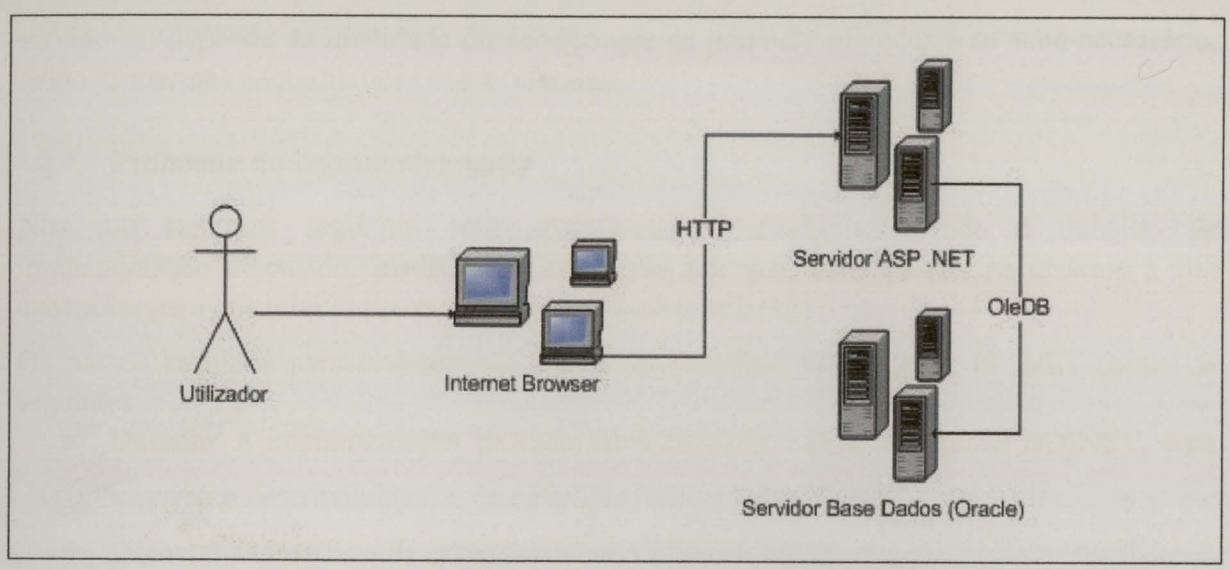


Figura 4 - Arquitectura básica da aplicação Wine-Cellar IS .NET

Passo de seguida a descrever cada um dos nós da arquitectura representada na imagem anterior.

a) Utilizador

Representa qualquer pessoa que possa interagir com o sistema.

b) Internet Browser

Uma vez que dois dos grandes objectivos eram a facilidade de acesso e distribuição da aplicação, optou-se por ter como "porta" de entrada uma *Browser* da Internet, uma vez que neste momento a utilização da Internet está massificada e já todos os computadores, com os sistemas operativos actuais, incluem uma qualquer versão destas aplicações. Desta forma não seria necessário a instalação de qualquer tipo de aplicação na maquina do utilizador, uma vez que apenas bastaria ele aceder a um endereço web (p. ex.: http://servidor/gescomnet/index.htm).

c) Servidor ASP .NET

Neste servidor, ou servidores, estariam alojadas todas as páginas web, que através de http seriam acedidas pelo utilizador. A utilização de um ou mais servidores aqui apenas está condicionado pela qualidade do serviço que se pretende oferecer. Sendo completamente transparente para o utilizador a utilização de mais do que um servidor (o utilizador apenas precisa de saber o endereço inicial da aplicação), colocar mais servidores em funcionamento permitiria uma melhoria em termos de tempo de resposta, fiabilidade, disponibilidade, entre outros requisitos não funcionais, mas como é óbvio aumentaria consideravelmente também o preço final.

d) Servidor Base Dados (Oracle)

Já neste último servidor estão instaladas as bases de dados. Em qualquer instalação desta nova solução deverão existir pelo menos duas bases de dados, no mesmo ou em diferentes servidores, a base de dados com os dados referentes à aplicação e a BDINET, a base de dados com os metadados. Nesta fase inicial continuou-se a utilizar tecnologia Oracle devido ao facto de a aplicação anterior ser já suportada por esta tecnologia, mas está previsto a possibilidade de utilizar outras tecnologias. Da mesma forma que o anterior, a utilização de um ou mais

servidores, depende da qualidade do serviço que se pretende oferecer e se ache necessário, tendo as mesmas contrapartidas antes referidas.

2.4 Processo de Desenvolvimento

Nos sub capítulos seguintes será apresentado detalhadamente todo o trabalho de implementação efectuado, justificando as opções tomadas e os passos necessários à sua execução, para que o leitor possa entender o trabalho realizado.

Os passos seguidos para o desenvolvimento do protótipo Wine-Cellar IS .NET foram os seguintes:

- Desenho e implementação de uma nova base de dados, designada BDINET, para suportar o desenvolvimento da aplicação Wine-Cellar IS .NET.
- Desenho da interface da aplicação Wine-Cellar IS .NET.
- Implementação da arquitectura genérica da aplicação Wine-Cellar IS .NET, que passa pela criação dos formulários genéricos da aplicação.

2.5 Desenho da base de dados de metadados (BDINET)

Inicialmente definiu-se cuidadosamente uma nova base de dados, designada BDINET, de suporte às aplicações desenvolvidas em ambiente .NET.

A construção desta base de dados foi imprescindível para se poder migrar as aplicações existentes da EGAPI para a plataforma .NET. Assim foi preciso definir nem mais nem menos do que 40 tabelas para a base de dados BDINET.

Vou apresentar de seguida apenas as tabelas mais importantes da BDINET.

As tabelas seguintes são tabelas com informação sobre as aplicações .NET desenvolvidas:

TABELA	DESCRIÇÃO
BDI_APLICACOES	Contém as aplicações existentes definidas em ambiente .NET
BDI_CONEXOES	Armazena as diversas ligações à Base de Dados que uma aplicação poderá ter, bem como os dados necessários para que a ligação seja estabelecida
BDI_MODULOS	Contém os módulos das aplicações
BDI_MENUS	Contém todos os itens dos menus aplicacionais e permite a geração dos menus por módulo
BDI_RELATORIOS	Contém todos os relatórios das aplicações
BDI_UTILIZADORES	Contém todos os utilizadores que podem trabalhar com as aplicações
BDI_PERFIS	Contém os perfis dos utilizadores
BDI_PERFIS_ACESSOS	Define os acessos para cada perfil

Tabela	Descrição
BDI_UTILIZADORES_APP_PERFIS	Contém os utilizadores pertencentes a um determinado perfil
BDI_EMPRESAS	Contém todas as empresas do cliente onde a aplicação está instalada
BDI_EMPRESAS_LOCAIS	Contém os locais (delegações) de cada empresa

Tabela 1 - Tabelas com informação sobre as aplicações .NET

As tabelas seguintes são tabelas com informação sobre os formulários que constituem as aplicações .NET:

Tabela	Descrição
BDI_FORMS	Contém todos os formulários declarados por aplicação .NET. Cada formulário tem definido a sua tabela de geração (tabela *Master*)
BDI_TABELAS_APLICACOES	Contém as tabelas pertencentes a cada aplicação.
BDI_TABPAI	Contém a relação pai/filhos entre as tabelas (relação *Master/Details*), isto é, contém todas as tabelas *Details* para cada tabela *Master*
BDI_OPERACOES	Contém todas as operações associadas aos formulários para cada aplicação
BDI_OPERACOES_ACESSOS	Garante o acesso a determinadas operações presentes num formulário a um determinado perfil

Tabela 2 - Tabelas com informação sobre os formulários da aplicação

As tabelas seguintes são as principais tabelas com informação sobre as tabelas das aplicações .NET:

Tabela	Descrição
BDI_TABELAS	Contém todas as tabelas das aplicações
BDI_CAMPOS	Contém todos os campos das tabelas aplicacionais
BDI_CAMPOS_RELACOES	Contém as relações de chave estrangeira entre os diversos campos
BDI_ACESSOS_CAMPOS	Contém os campos a que o perfil do utilizador não tem acesso. Assim podem-se gerar os formulários dinamicamente só com os campos a que o perfil do utilizador tem acesso

Tabela 3 - Tabelas com informação sobre as tabelas das aplicações

Interessa, antes de explicar em pormenor todo o processo de criação dos formulários da aplicação Wine-Cellar IS .NET, dar uma pequena descrição das tabelas BDI_FORMS, BDI_TABPAI, BDI_CAMPOS e BDI_CAMPOS_RELACOES, uma vez que estas fazem parte das tabelas mais utilizadas na construção dos formulários.

A tabela BDI_FORMS possui 7 campos descritos a seguir:

Campo	Descrição
FM_CODIGO	Representa o código do formulário
FM_APP_ID	Representa a identificação da aplicação
FM_DESIGNACAO	Representa a designação do formulário
FM_DIC_INDICE	Representa o índice de tradução
FM_DIC_LNG_CODIGO	Representa o código da língua
FM_GERADA	Indica se o formulário é gerado dinamicamente ou não
FM_TA_TAB_ID	Representa a identificação da tabela *Master* do formulário

Tabela 4 - Descrição da tabela BDI_FORMS

A tabela BDI_TABPAI possui 4 campos descritos a seguir:

Campo	Descrição
PAI_TA_APP_ID	Identifica a aplicação
PAI_TA_TAB_ID_PAI	Identifica a tabela *Master* (pai)
PAI_TA_TAB_ID_FILHO	Identifica a tabela *Detail* (filho)
PAI_ORDEM	Representa a ordem da tabela *Detail*

Tabela 5 - Descrição da tabela BDI_TABPAI

A tabela BDI_CAMPOS possui 23 campos descritos a seguir:

Campo	Descrição
CMP_TAB_ID	Representa a tabela à qual o campo pertence
CMP_CODIGO	Representa o código do campo
CMP_NATUREZA	Representa a natureza do campo, ou seja o seu tipo de dados (varchar, number, date, etc.)
CMP_TAMANHO	Representa o tamanho do tipo de dados associado ao campo
CMP_DESCRICAO	Representa a descrição do campo (na língua original)

CAMPO	DESCRIÇÃO
CMP_DIC_INDICE	Representa o índice para tradução
CMP_DIC_LNG_CODIGO	Representa o código da língua
CMP_PAPEL	Representa o papel do campo na tabela (PK - Primary Key, FK - Foreign Key, AU – Auxiliar, FS - Foreign key Sem descrição e não associado à campos adicionais, FS {digito} - Foreign key Sem descrição mas associado à campos adicionais, AD {digito} - ADicional)
CMP_ORDEM	Representa a ordem do campo na tabela
CMP_OBRIGATORIO	Indica se o campo é obrigatório ou não
CMP_FUNCAO_VALIDACAO	Representa a função para validar o campo
CMP_OBJECTO_ECRAN	Representa o objecto para manipular o campo (*Spinner*, *Combobox*, etc.)
CMP_PARAMETRO_OBJECTO	Representa os parâmetros necessários para o objecto
CMP_VISIVEL	Indica se o campo é visível ou não, isto é, se é ou não mostrado no formulário
CMP_SEQUENCIAL	Indica se o campo é sequencial ou não
CMP_SEQUENCIA	Representa o nome da sequência a utilizar para determinar o próximo valor do campo no caso de ser sequencial
CMP_FNC_ID	Representa a função usada pelo campo
CMP_LABEL	Representa a descrição resumida do campo
CMP_DIC_INDICE_LAB	Representa o índice de tradução da *label* associada ao campo
CMP_MIGRACAO	Indica se o campo é para migrar ou não
CMP_TAMANHO_PX	Indica o tamanho do campo em pixeis definido para a sua apresentação no formulário
CMP_MASCARA	Representa a máscara usada para formatar o campo
CMP_TIPO	Representa o tipo do campo se este for especial (F - Flag, L - Local, LF - Local de Filtragem, DC – Data de Criação, DA – Data de Alteração, UC – Utilizador de Criação, UA – Utilizador de Alteração)

Tabela 6 - Descrição da tabela BDI_CAMPOS

A tabela BDI_CAMPOS_RELACOES possui 5 colunas descritas a seguir:

CAMPO	DESCRIÇÃO
CR_CMP_TAB_ID	Identifica a tabela com relações
CR_CMP_CODIGO	Representa o código do campo da tabela com relações
CR_CMP_TAB_ID_REL	Identifica a tabela relacional
CR_CMP_CODIGO_REL	Representa o código do campo relacional
CR_RELACAO	Identifica de maneira única a relação, de maneira a ser possível determinar as relações distintas para a mesma tabela relacional

Tabela 7 - Descrição da tabela BDI_CAMPOS_RELACOES

No anexo B pode encontrar-se o diagrama desta base de dados.

2.6 Desenho da Interface

Uma vez definidos os metadados necessários à criação dos formulários genéricos automaticamente, prosseguiu-se com a construção do *layout* da aplicação Wine-Cellar IS .NET, ou seja, da sua interface ou GUI *(Graphical User Interface)*. O logótipo e muitos dos ícones usados na barra de ferramenta do produto WIS foram reaproveitados para o *layout* da sua nova versão em ambiente *Web*.

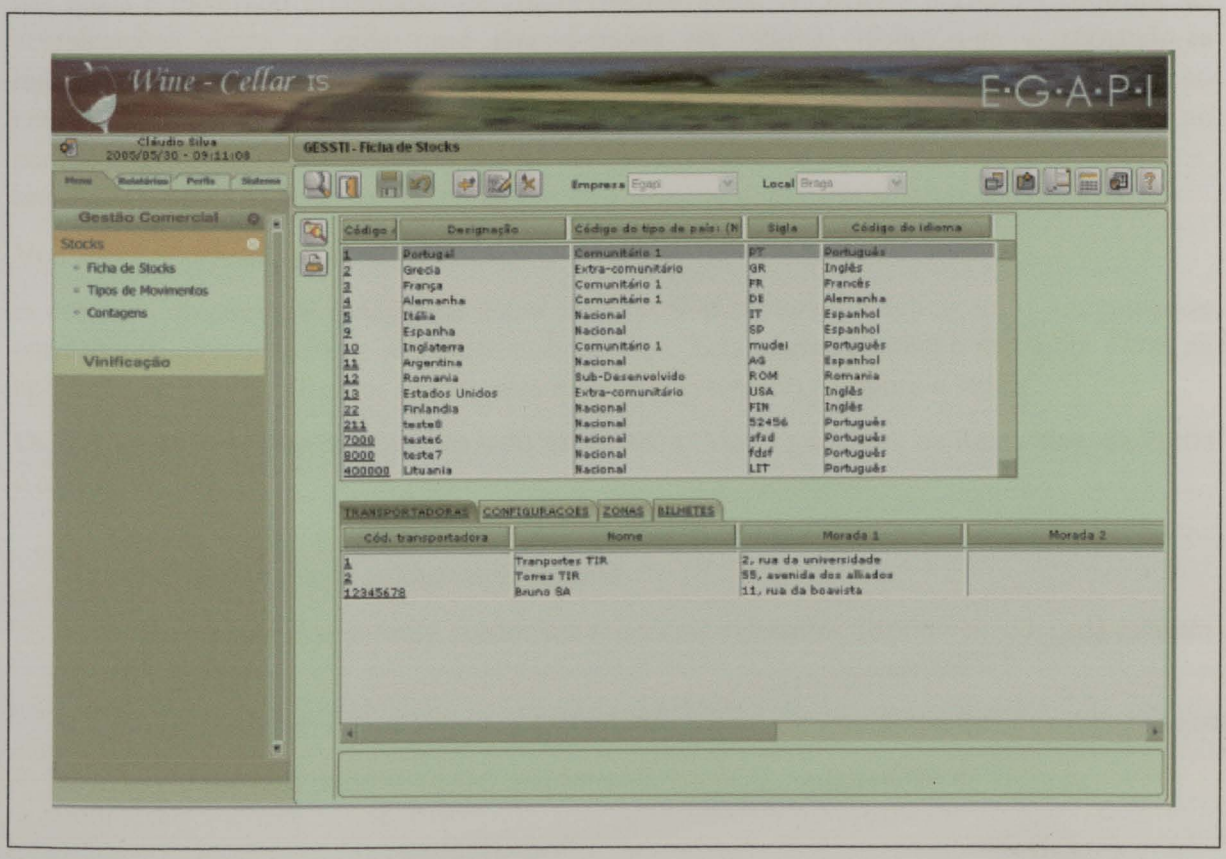


Figura 5 - Layout do Wine Cellar IS .NET

Na página ASP.NET principal da aplicação, designada inicio.aspx, encontra-se definido o "esqueleto" do *layout*.

No cabeçalho é possível visualizar uma imagem com o logótipo do produto Wine-Cellar da EGAPI. Imediatamente a seguir ao logótipo, aparecem no canto esquerdo da página campos informativos com o nome do utilizador activo (o que entrou na aplicação) e a data/hora actual do servidor. O botão observado à esquerda dessa informação permite esconder o menu da aplicação.

Figura 6 – Logótipo do Wine Cellar

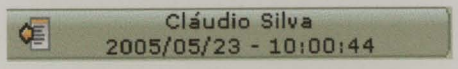


Figura 7 – Utilizador e Data/Hora

No resto da página, onde se encontra o essencial da aplicação, foram usadas diversas *frames* nas quais é mostrado o conteúdo de outras páginas aspx. As *frames* permitem controlar as actualizações feitas a cada uma das divisões da página, isolando-as e tornando-as independentes entre elas. Por exemplo é possível fazer um *refresh* apenas à página relativa ao conteúdo principal, fazendo o redireccionamento adequado do *target* da *frame* ligada ao conteúdo principal. Resumindo, as *frames* estão associadas às partes não estáticas da página, ou seja às partes que precisam de actualização constante durante as sessões dos utilizadores.

Vou agora apresentar as divisões da página principal que usam essas *frames*.

À esquerda da página é usada uma *frame* para mostrar o menu da aplicação; o seu conteúdo corresponde ao conteúdo da página menuflash.aspx. Esta página contém o menu em *Flash* da aplicação na qual o utilizador entrou; este menu é apresentado na figura a seguir.

Os diferentes itens disponibilizados pelo menu são o Menu principal, os Relatórios, os Perfis e o Sistema.

- O item Menu permite aceder aos vários formulários para manutenção de dados pertencentes a cada módulo aplicacional.
- O item Relatórios permite aceder aos relatórios existentes disponíveis em cada módulo aplicacional.
- O item Perfis permite aceder aos formulários para definição de perfis.
- O item Sistema permite aceder aos formulários para configuração do sistema.

Figura 8 - Menu em Flash

A construção do menu necessitou do uso de múltiplas tabelas da BDINET tais como BDI_MENUS, BDI_MODULOS, BDI_MODULOS_RELATORIOS, BDI_RELATORIOS.

À direita da página existem cinco *frames*, cujo conteúdo passo agora a descrever.

No topo, uma *frame* contém a página tituloform.aspx e mostra o título do formulário actual. A tabela da BDINET usada é a BDI_FORMS.

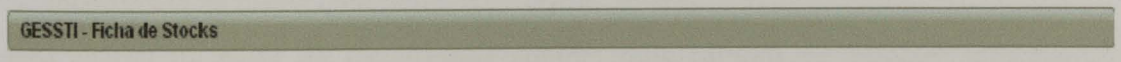


Figura 9 – Título do formulário

Imediatamente a seguir existe uma *frame* que contém a página tarefas.aspx e mostra uma barra de ferramentas horizontal das tarefas associadas aos formulários da aplicação. As tabelas da BDINET mais usadas são: BDI_OPERACOES, BDI_OPERACOES_ACESSOS, BDI_UTILIZADORES_APP_PERFIS.

Todos os formulários da aplicação têm em comum esta barra de ferramentas. Esta serve para efectuar as normais operações de consulta, inserção, etc. Além disso contém outras funcionalidades, que permite à aplicação um ambiente mais integrado.

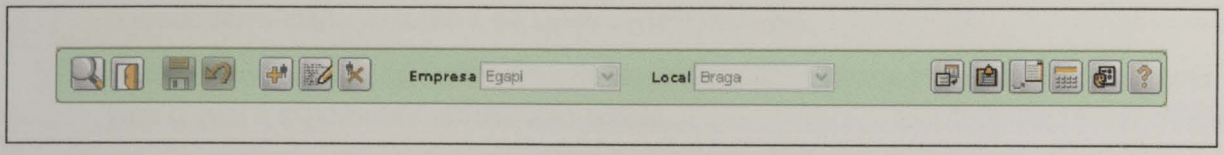


Figura 10 – Barra de tarefas

Interessa aqui descrever quais as funcionalidades atribuídas a cada botão, pois no desenvolvimento dos formulários é necessário ter em conta as suas funcionalidades.

Na parte esquerda da barra de ferramentas estão os botões que permitem ao utilizador efectuar operações sobre os dados e que só se encontram activos (menos o botão sair) quando o formulário actual é um formulário para manutenção de dados. Todos eles após um clique chamam funções definidas no formulário actual (visível). Por exemplo, quando se clica no botão de inserção é chamada uma sub rotina inserir previamente definida na página ASP.NET associada ao formulário actual.

Passo a explicar a funcionalidade de cada botão, pela ordem que aparecem na Figura 10:

- Consultar: permite ao utilizador pesquisar por qualquer campo do formulário o registo pretendido.
- Sair: permite ao utilizador sair do formulário em que se encontra.
- Confirmar: esta operação só está activa quando o utilizador previamente clica num destes botões: consultar, inserir, alterar ou remover, e serve para confirmar as acções efectuadas.
- Cancelar: esta operação está activa nas mesmas condições da anterior e serve para cancelar as acções efectuadas.
- Inserir: quando o utilizador clica no botão associado à esta funcionalidade é criado um novo registo.
- Alterar: esta funcionalidade permite ao utilizador alterar os dados.
- Remover: serve para que o utilizador possa remover os registos que deseja, para isso clica sobre eles, sendo nesse instante verificado se a sua remoção é possível. A remoção de um determinado registo pode não ser possível efectuar devido à integridade de dados com registos de outras tabelas.

A parte central da barra de ferramentas contém dois campos de carácter informativo: a empresa e o local escolhidos pelo utilizador.

A parte direita da barra contém os botões que não lidam directamente com os dados, que são meramente auxiliares e que passo agora a descrever pela ordem que aparecem na imagem:

- Este botão permite mudar o tipo de objecto usado para representar a tabela principal no formulário actual para manutenção de dados; ao clicar no botão o tipo de objecto passa de *Grid* a Formulário e vice-versa consoante o tipo de objecto actualmente usado.
- Este botão está associado ao processo de criação de novas listagens a partir dos dados mostrados no formulário actual para manutenção de dados: um primeiro clique no

botão inicia o processo de criação das listagens abrindo um formulário onde o utilizador selecciona os parâmetros para a nova listagem e um segundo clique termina o processo iniciado gerando a listagem correspondente.

Botão de exportação, que abre uma nova janela onde o utilizador escolhe o programa para o qual é exportada a informação actual.

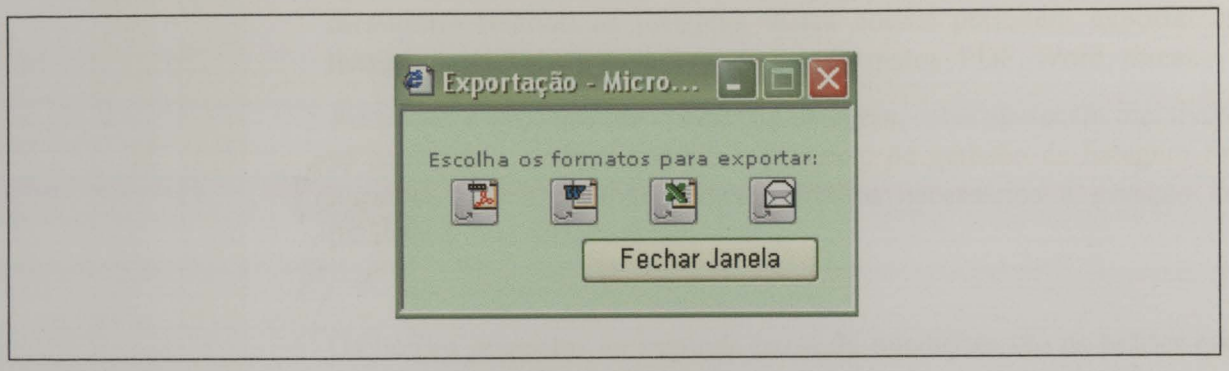


Figura 11 – Exportação

Este botão abre uma calculadora (implementada em flash).

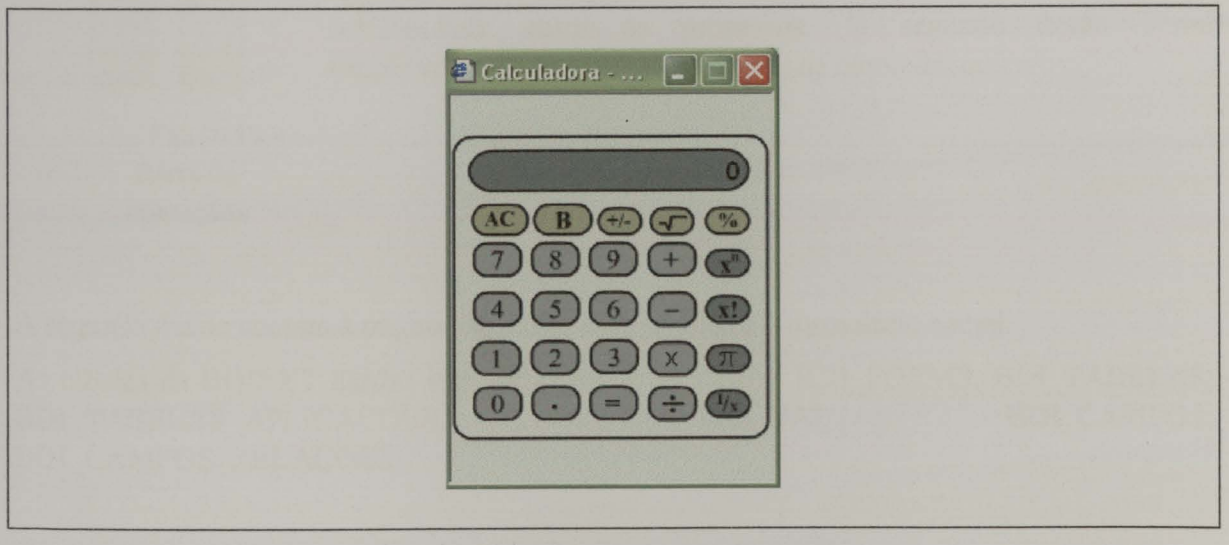


Figura 12 – Calculadora

Este botão abre um formulário com as preferências actuais do utilizador.

Botão de ajuda chamando um ficheiro HTML Help compilado (.chm).

Por baixo da *frame* com a barra de tarefas existem outras três *frames*.

A primeira *frame* contém a página operacoes.aspx e mostra uma barra de ferramentas vertical das operações associadas ao formulário actual.

As tabelas da BDINET principalmente usadas são BDI_OPERACOES, BDI_OPERACOES_ACESSOS, BDI_UTILIZADORES_APP_PERFIS.

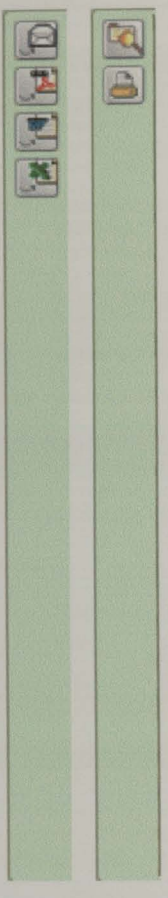


Figura 13 – Barra de operações

Vou descrever as funcionalidades atribuídas a cada botão representado nas barras de operações ilustrativas apresentadas na figura 18.

Os botões presentes na primeira barra de operações são os botões que aparecem na barra de operações apenas quando o formulário actual é o formulário relativo às listagens. Estes botões permitem exportar as listagens geradas por e-mail ou para os formatos: PDF, Word, Excel.

Ao iniciar o processo de criação da listagem, estes aparecem inactivos, só passando a activos quando o processo de geração da listagem for possível, isto é, quando todos os dados necessários a geração da listagem forem introduzidos.

Os botões presentes na segunda barra de operações são os botões que aparecem nas barras de operações associadas aos formulários para manutenção de dados.

O primeiro permite a funcionalidade *Drag & Relate* ao abrir o formulário relacionado com o formulário actual, garantindo assim a continuidade entre os processos. O segundo botão permite simplesmente imprimir informação tais como documentos.

A segunda *frame* contém a página conteudo.aspx e mostra o formulário actual.

As tabelas da BDINET usadas são por exemplo as tabelas BDI_FORMS, BDI_TABELAS, BDI_TABELAS_APLICACOES, BDI_TAB_PAI, BDI_CAMPOS, BDI_CAMPOS_RELACOES.

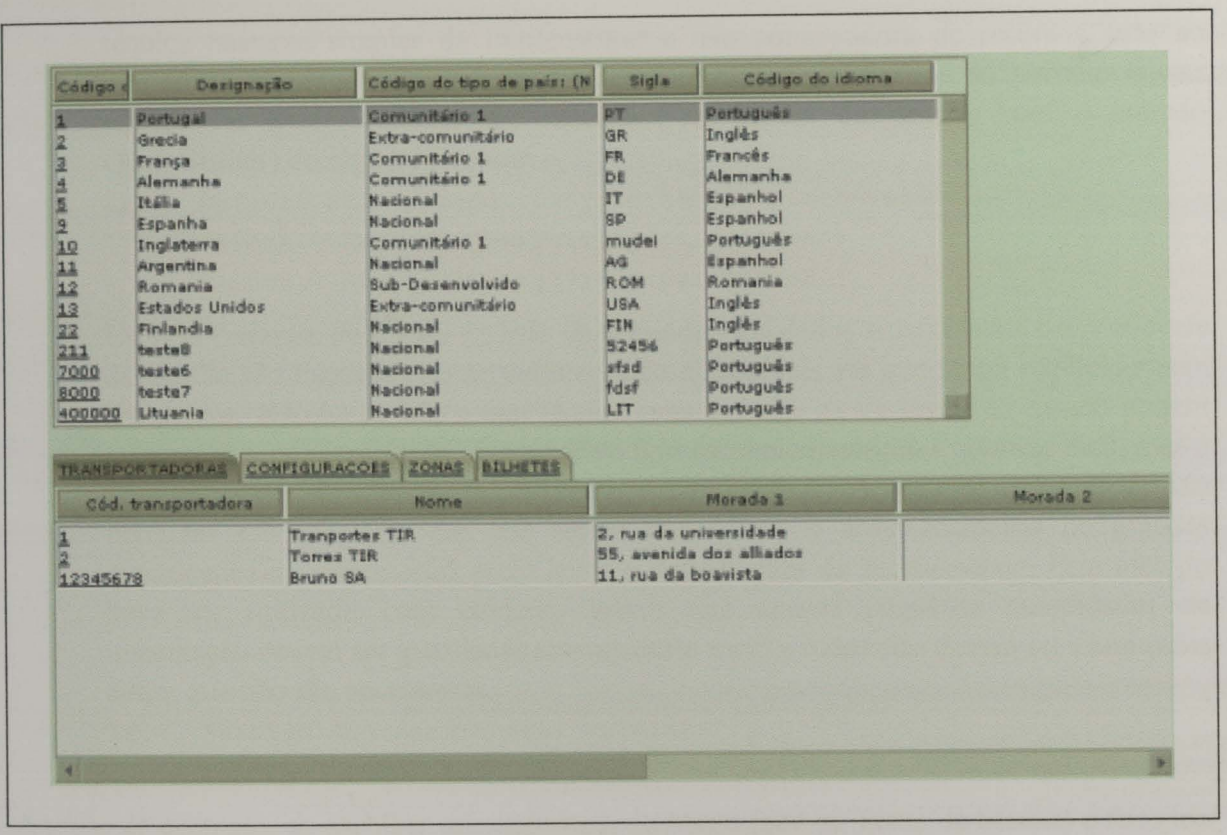


Figura 14 – Conteúdo principal com o formulário actual

A terceira *frame* contém a página mensagem.aspx e permite mostrar ao utilizador as mensagens informativas, de erro e de aviso resultantes das operações efectuadas sobre o formulário actual.

As tabelas da BDINET usadas são por exemplo as tabelas BDI_MENSAGENS e BDI_TIPOS_MENSAGENS.

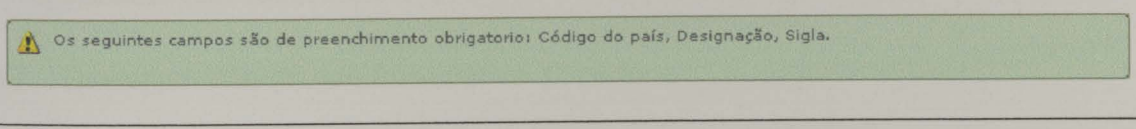


Figura 15 – Mensagens

Todas as páginas .aspx atrás descritas e relativas à uma parte específica da página principal da aplicação (o menu, a barra de tarefas, a barra de operações, o conteúdo principal com o formulário actual, o titulo do formulário, as mensagens) precisam de comunicar e interagir umas com as outras, embora sendo independentes. Vou agora precisamente explicar como é feita a comunicação entre as várias páginas .aspx contidas nas *frames* da página principal inicio.aspx.

A comunicação entre as páginas .aspx requer a passagem de parâmetros de uma página para outra. A passagem desses parâmetros pode ser feita de duas maneiras:

- Usando *Query Strings* do lado do cliente – As *Query Strings* são pedaços de informação que são adicionados no final do URL de uma página. A utilização desta técnica permite passar informação do tipo (variável=valor). Demonstrou ser uma

técnica bastante simples de implementar e não consumidora de recursos, mas ao mesmo tempo mostrou ser uma técnica muito limitada porque só permite enviar strings. As *Query Strings* são recebidas na página de destino com o método QueryString() do objecto *Request*.

```
Ex:   Default.aspx?parametroA=valorA&parametroB=valorB;
      valorA = Request.QueryString(parametroA);
      valorB = Request.QueryString(parametroB);
```

- Usando variáveis de sessão do lado do servidor – Os dados armazenados nas Variáveis de Sessão são mantidos no servidor e estão disponíveis até que sejam explicitamente removidos. Não há limites a esta técnica uma vez que as Variáveis de Sessão podem armazenar qualquer tipo de objecto. Este tipo de implementação é também fácil, mas é necessário ter cuidado com a utilização abusiva pois, uma vez que as Variáveis são mantidas em memória até que sejam removidas, o armazenamento de grandes quantidades de informação pode levar a degradações de *performance*. Este método deve ser utilizado com cuidado, sendo que apenas pequenas quantidades de informação devem ser guardadas através deste meio e todas elas devem ser eliminadas assim que não são necessárias.

```
Ex:   Session.Add(parametroA, valorA);
      valorA = Session.Contents[parametroA];
```

A título de exemplo, ao escolher um dado item do Menu (por ex. Ficha de Stocks) é feito um redireccionamento do *target* da *frame* designada Conteúdo para a página conteudo.aspx à qual é passado um parâmetro: o código do formulário associado ao item escolhido (`conteudo.aspx?fm_codigo=GESSTI`).

Depois, a partir da informação presente na tabela BDI_FORMS da BDINET, a página conteudo.aspx trata de redireccionar a mesma *frame* para a página .aspx específica cujo nome é o código do formulário – caso o formulário não seja gerado dinamicamente (`gessti.aspx`) – ou então para a página .aspx genérica – caso o formulário seja gerado dinamicamente – adicionada dos parâmetros seguintes: o nome da tabela e do *owner* correspondentes (`default.aspx?table_sel=STOCKS&owner=GESCOM`).

2.7 Desenvolvimento dos formulários genéricos

Depois da construção do *layout*, prosseguiu-se com a implementação da arquitectura genérica da aplicação.

Esta passa pela criação de dois formulários genéricos:

- O formulário genérico para manutenção de dados, usado na geração dinâmica da maioria dos formulários simples da aplicação.
- O formulário genérico para criação de novas listagens.

2.7.1 Formulário genérico para manutenção de dados

O formulário genérico para manutenção de dados permite apresentar os dados de uma dada tabela aplicacional – na forma de *Grid* ou de Formulário consoante o tipo de objecto escolhido – e, possivelmente, os dados relativos às tabelas relacionadas com esta tabela principal. A página ASP.NET que implementa este formulário designa-se default.aspx e apresenta-se como uma página parametrizada. Os parâmetros são a designação da tabela

principal a representar e o *owner* dessa tabela. Os valores associados a esses parâmetros são recebidos pela página através de *Query Strings*.

O formulário genérico para manutenção de dados pode ser do tipo *Master* ou *Master/Details* consoante a tabela principal a representar.

Entenda-se por formulário do tipo *Master*, o formulário cuja tabela principal não está ligada a outras tabelas da aplicação. Nesse caso o formulário genérico representa apenas a informação da tabela principal ou tabela *Master* (na forma de uma *Grid* ou de um *Formulário*) e as funcionalidades operacionais existentes na barra de tarefas se aplicam apenas a esta tabela.

Se a tabela principal (tabela *Master*) estiver ligada a outras tabelas aplicacionais, então estas últimas (designadas tabelas *Details*) são representadas através de *Grids* situadas por baixo da *Grid* ou do *Formulário* representando a tabela *Master*. Neste caso obtém-se um formulário do tipo *Master/Details* e as funcionalidades operacionais existentes na barra de tarefas aplicam-se tanto à tabela *Master* como às suas tabelas *Details*, mas só uma delas pode ser requisitada num dado instante (esta corresponde a tabela previamente seleccionada). Mais, se houver mais do que uma tabela *Detail* associada a tabela *Master* então só é representada a informação de uma delas num dado instante; para mudar de tabela *Detail* criou-se um painel de separadores em que cada separador tem a designação da tabela *Detail* à qual este dá acesso.

Toda a informação necessária sobre a relação pai/filhos entre a tabela *Master* e as tabelas *Details* correspondentes encontra-se na tabela BDI_TABPAI da base de dados BDINET. Lá existe, para cada tabela *Master*, a identificação das suas tabelas *Details*.

2.7.1.1 Tipo Master

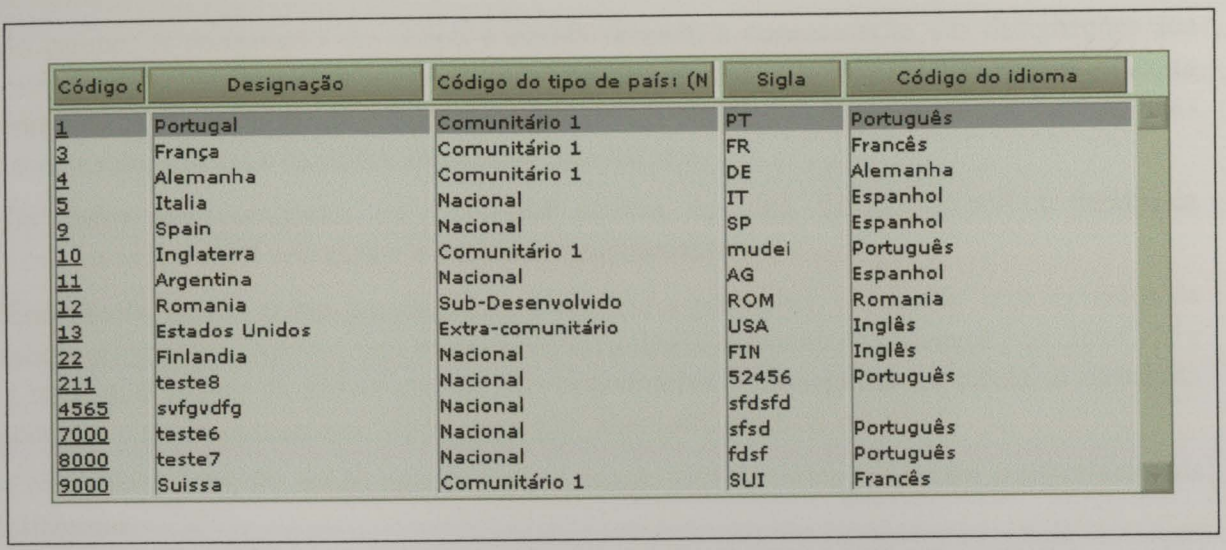
Nesta secção são descritos os passos seguidos na construção do formulário do tipo *Master*, ou seja, do formulário em que a tabela principal não é referenciada em outras tabelas aplicacionais.

Vou descrever a construção deste tipo de formulário conforme o tipo de objecto escolhido para representar a tabela principal: uma *Grid* ou um *Formulário*.

a) Usando uma *Grid*

Para representar a tabela principal na forma de uma grelha ou *Grid*, usou-se dois tipos de controlos ASP: o controlo *Table* e o controlo *DataGrid*. Ambos são criados dinamicamente porque o nome da tabela principal e do *owner* são passados como parâmetros à página default.aspx (`table_sel` e `owner`) e logo não são conhecidos à partida.

As designações dos campos da tabela principal são mostradas na forma de botões, que constituem as células do controlo *Table*, enquanto que os dados da tabela são mostrados no controlo *DataGrid*. A *Table* serve então de cabeçalho à *DataGrid*.



Código	Designação	Código do tipo de país: (N	Sigla	Código do idioma
1	Portugal	Comunitário 1	PT	Português
3	França	Comunitário 1	FR	Francês
4	Alemanha	Comunitário 1	DE	Alemanha
5	Italia	Nacional	IT	Espanhol
9	Spain	Nacional	SP	Espanhol
10	Inglaterra	Comunitário 1	mudei	Português
11	Argentina	Nacional	AG	Espanhol
12	Romania	Sub-Desenvolvido	ROM	Romania
13	Estados Unidos	Extra-comunitário	USA	Inglês
22	Finlandia	Nacional	FIN	Inglês
211	teste8	Nacional	52456	Português
4565	svfgvdfg	Nacional	sfdsfd	
7000	teste6	Nacional	sfsd	Português
8000	teste7	Nacional	fdsf	Português
9000	Suissa	Comunitário 1	SUI	Francês

Figura 16 – Controlos *Table* e *DataGrid* associados ao tipo de objecto Grid

Ao carregar a página default.aspx (evento `page_load`), os metadados associados à tabela principal são guardados num *DataSet* chamado `metadados`. Esses metadados são os registos da tabela BDI_CAMPOS da base de dados BDINET correspondentes à tabela principal, filtrados pelo campo CMP_VISIVEL e ordenados pelo campo CMP_ORDEM. Dessa forma obtêm-se todos os campos visíveis da tabela seleccionada, ordenados.

Ao preencher o *DataSet* `metadados` ficam identificados ao mesmo tempo os campos a mostrar no formulário, uma vez que cada registo do *DataSet* corresponde a um campo da tabela principal. Podemos então construir as células do controlo *Table*, assim como as colunas do controlo *DataGrid*.

Relativamente à construção das células da *Table*, a cada registo do *DataSet* `metadados` corresponde uma nova célula na tabela *Table* com um botão, e a cada um desses botões está associada a funcionalidade seguinte: a ordenação dos dados apresentados na *DataGrid* pelo campo associado ao botão de forma ascendente ou descendente consoante se clica uma ou duas vezes no botão.

Relativamente à construção das colunas da *DataGrid*, a propriedade *AutoGenerateColumns* da *DataGrid* está previamente definida como *false*, uma vez que as suas colunas são construídas dinamicamente. Para cada registo dos metadados cria-se uma nova coluna na *DataGrid* do tipo *ButtonColumn*, se o campo correspondente ao registo tem o papel de chave primária, ou do tipo *BoundColumn* caso contrário. Os dados presentes nas colunas do tipo *ButtonColumn* aparecem na forma de um botão do tipo *link* associado ao comando *Select*. Dessa forma, a selecção de uma linha da *DataGrid* fica associada ao clique num desses *links* presentes na linha. Mais, se existir uma mascara de formatação a aplicar a um campo então atribui-se a *string* de formatação correspondente a mascara à propriedade *DataFormatString* da coluna da *DataGrid* correspondente ao campo.

Depois de criar a *DataGrid* é preciso preenchê-la com os dados da tabela seleccionada. Esses dados são guardados num *DataSet* designado `dados`. Esses dados são os registos da tabela aplicacional, cuja designação corresponde ao valor do parâmetro `table_sel` e que pertence à base de dados cuja designação corresponde ao valor do parâmetro `owner`. No entanto só são seleccionados os campos da tabela definidos como visíveis nos metadados e, para os campos com o papel de chave estrangeira, substitui-se o código da chave estrangeira pela designação

do campo. A designação do campo é obtida fazendo a concatenação das designações dos campos auxiliares presentes na tabela relacionada. Os dados guardados no *DataSet* resultam então de uma instrução SQL complexa que faz a junção da tabela principal com as suas tabelas relacionadas e que filtra os campos não visíveis.

Os dados são carregados na *DataGrid* através de uma função complexa designada `bind_datagrid` que vou passar a descrever sucintamente.

Esta função recebe como argumentos a *DataGrid* a preencher, o *DataSet* com os dados da tabela principal, o *DataSet* com os metadados da tabela, o campo de ordenação da *DataGrid* e a respectiva ordem de ordenação, as chaves primárias e estrangeiras da tabela, o *owner* da tabela, e o modo operacional actual (inserção, alteração, procura, etc.).

Consoante o valor do modo operacional, a função `bind_datagrid` tem comportamentos diferentes:

- Assim, em modo de selecção (modo em que não é preciso alterar os dados, portanto modo sem necessidade de edição de linhas da *DataGrid*), a função limita-se a criar uma *DataView* (vista) a partir do *DataSet* com os dados, ordenada pelo campo/ordem de ordenação, e depois a preencher a *DataGrid* com a vista em questão.

```
DataView myView = new DataView();
myView = dados_table.Tables[0].DefaultView;
myView.Sort = SortExpression + " " + SortMode;
DataGrid.DataSource = myView;
DataGrid.DataBind();
```

Código	Designação	Código do tipo de país: (N	Sigla	Código do idioma
1	Portugal	Comunitário 1	PT	Português
3	França	Comunitário 1	FR	Francês
4	Alemanha	Comunitário 1	DE	Alemanha
5	Italia	Nacional	IT	Espanhol
9	Spain	Nacional	SP	Espanhol
10	Inglaterra	Comunitário 1	mudei	Português
11	Argentina	Nacional	AG	Espanhol
12	Romania	Sub-Desenvolvido	ROM	Romania
13	Estados Unidos	Extra-comunitário	USA	Inglês
22	Finlandia	Nacional	FIN	Inglês
211	teste8	Nacional	52456	Português
4565	svfgvdfg	Nacional	sfdsfd	
7000	teste6	Nacional	sfsd	Português
8000	teste7	Nacional	fdsf	Português
9000	Suissa	Comunitário 1	SUI	Francês

Figura 17 – *Grid* em modo de selecção

- Caso o modo seja de alteração, o valor da propriedade *EditItemIndex* da *DataGrid* é previamente instanciado com o índice da linha que queremos editar para que depois, ao preencher a *DataGrid* com a vista descrita anteriormente, aparece editada a linha pedida para editar antes de se chamar a função, isto é a linha para actualizar os dados. A seguir actualiza-se a linha editada, modificando se necessário o tipo do controlo presente em cada célula da linha editada (o controlo criado por omissão para edição é a *TextBox*) para o tipo pretendido. A certas células adicionam-se também botões que trazem funcionalidades complementares (botões que abrem uma janela com um

calendário para as datas ou com uma lista de valores para as chaves estrangeiras, etc.). Assim, por exemplo para as células correspondentes aos campos com o papel de chave estrangeira, a *TextBox* é substituída por uma *ComboBox* preenchida dinamicamente a partir dos dados da tabela relacionada com o campo, e é adicionado à célula um botão de LOV (*List Of Values*) que permite escolher o item numa grelha onde o utilizador pode filtrar e ordenar os dados.

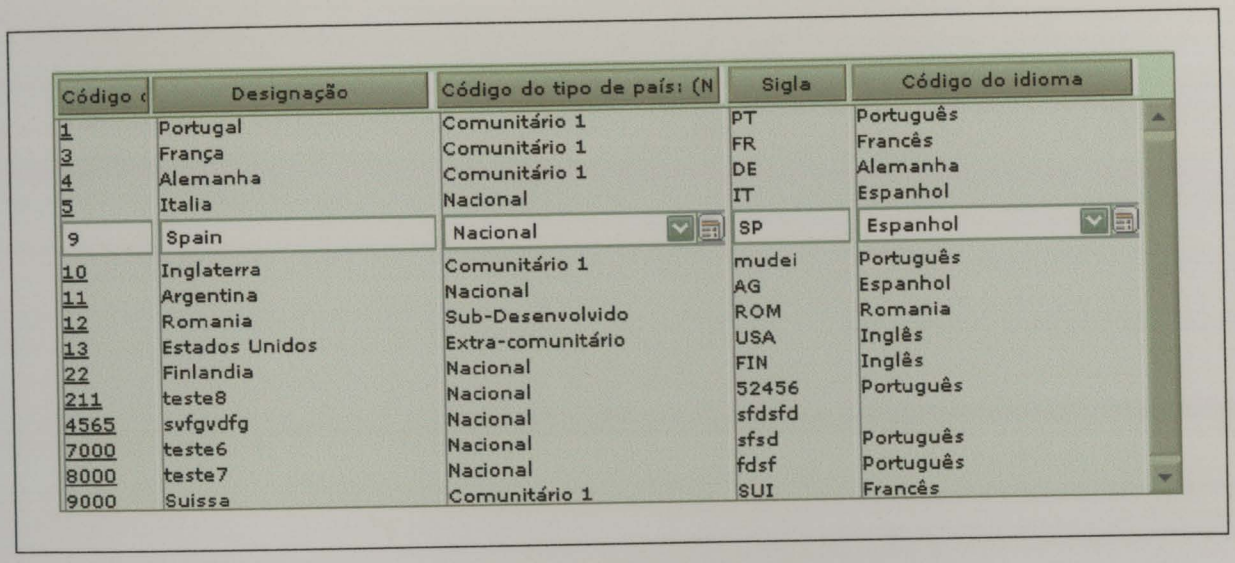


Figura 18 – *Grid* em modo de alteração

- Caso o modo seja de inserção, adiciona-se inicialmente um novo registo (com os campos sequenciais já determinados) no final da vista descrita atrás e fixa-se o valor da propriedade *EditItemIndex* da *DataGrid* para o índice da linha inserida. Assim, ao preencher a *DataGrid* com esta vista, aparece editada a linha para inserir os dados. A seguir actualiza-se a linha editada exactamente como para o modo de alteração, modificando se necessário o tipo do controlo presente em cada célula da linha editada e/ou adicionando botões complementares.

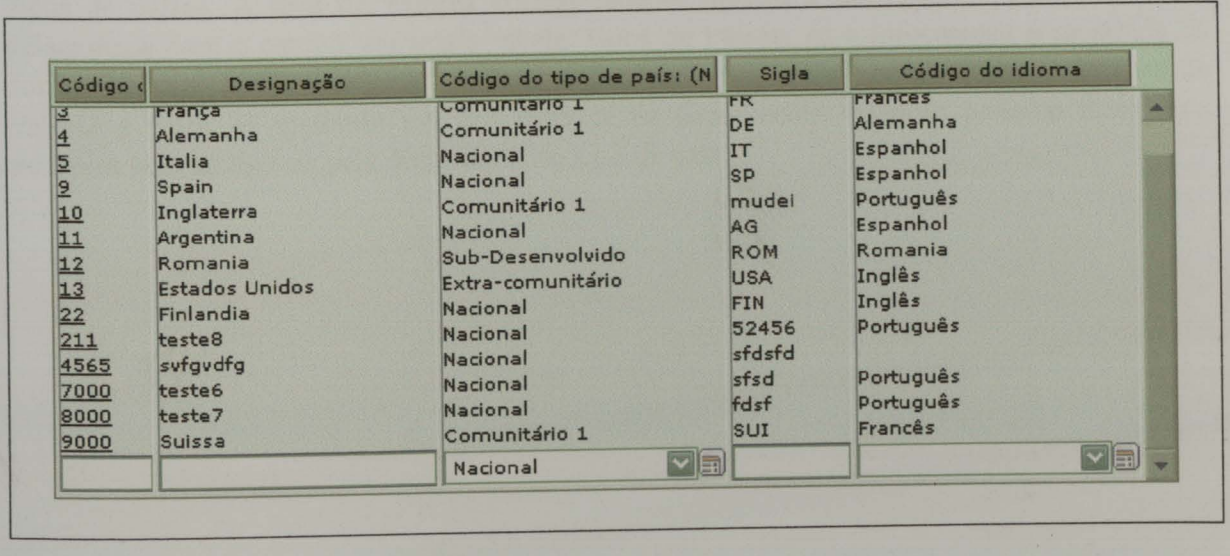


Figura 19 – *Grid* em modo de inserção

- Caso o modo seja de procura, cria-se primeiro uma nova vista com um único registo vazio e fixa-se o valor da propriedade *EditItemIndex* da *DataGrid* para o índice da linha correspondente a este registo, isto é o índice 0. Assim, ao preencher a *DataGrid* com esta vista, aparece editada a linha para procurar dados. A seguir actualiza-se a linha editada, adicionando um botão de LOV às células associadas a campos com o papel de chave estrangeira.

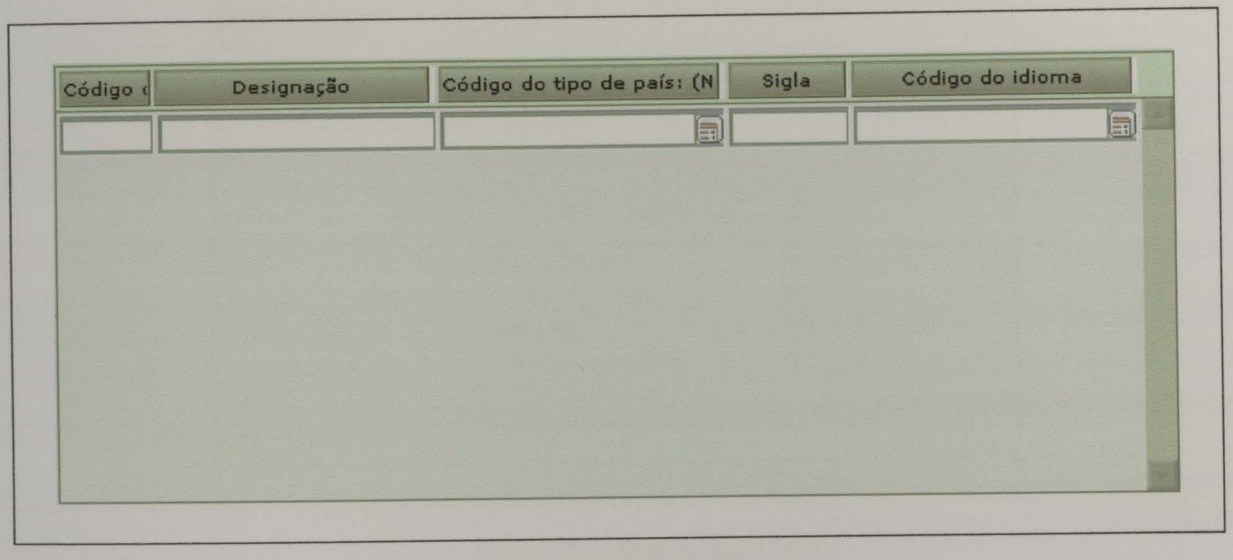


Figura 20 – *Grid* em modo de procura

O botão de LOV, que aparece sempre associado aos campos com papel de chave estrangeira quando a *Grid* está em modo de edição (alteração, inserção ou procura), abre uma nova janela com uma lista de valores correspondente a informação do campo. Ao seleccionar um registo da lista de valores, a janela é automaticamente fechada e o item escolhido na lista é usado para actualizar a *TextBox* ou a *ComboBox* associada ao campo.

Na figura seguinte, visualizamos a *ComboBox* usada na Grid associada a tabela Países (em modo alteração/inserção) e referente ao campo com o papel de chave estrangeira relativo aos tipos de países. A lista de valores correspondente mostra a informação relativa a tabela relacionada com o campo, ou seja a tabela Tipos de Países. Aí a informação é mostrada na forma de uma *Grid* e é possível ordenar a lista de valores e procurar um item da lista de valores a partir de qualquer um dos campos da *Grid*; neste exemplo pode-se fazer uma pesquisa pelo código ou pela designação do tipo de país.

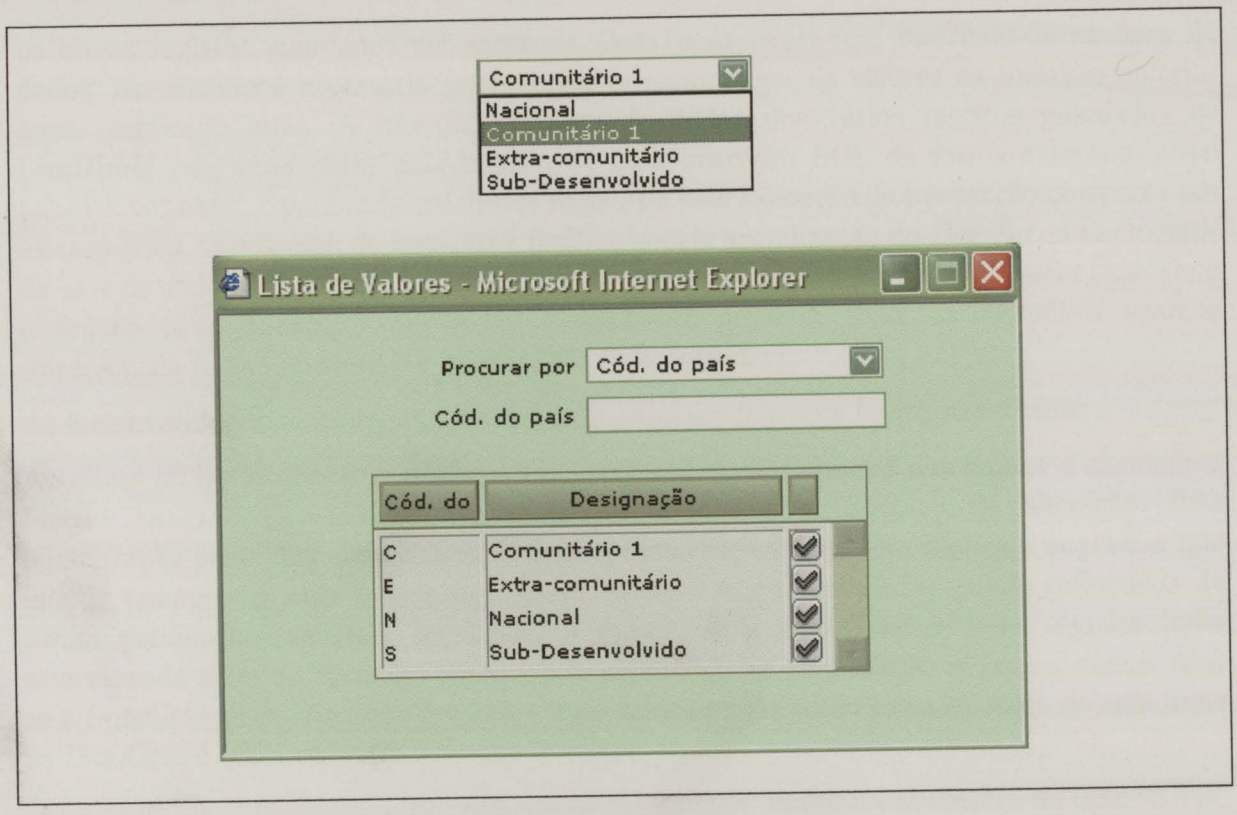


Figura 21 – *ComboBox* / LOV

Vou agora explicar as operações de inserção, alteração, remoção e procura associadas ao formulário do tipo *Master* quando a tabela principal é representada por uma *Grid*.

Ao clicar no botão inserir, que se encontra na barra de ferramentas das tarefas, é chamada uma função designada `inserir_dados` que trata de inserir um novo registo na *DataGrid*.

Salienta-se que é possível efectuar múltiplas inserções de uma só vez, isto é pode-se fazer vários pedidos de inserção clicando sucessivamente no botão inserir e só no fim finalizar as inserções efectuadas clicando no botão confirmar.

A função `inserir_dados` determina inicialmente se já houve ou não um pedido de inserção.

Se não houve ainda pedidos de inserção então é chamada uma função auxiliar `inicializa_inserir` que trata de inicializar o processo de inserção (esta inicialização passa nomeadamente pela criação de uma nova *DataTable* chamada `registos` que guarda temporariamente os novos registos inseridos até estes serem guardados de forma permanente na base de dados); a seguir é adicionada uma nova linha à *DataGrid*, mudando o modo corrente de selecção para inserção e chamando a função `bind_datagrid` descrita atrás.

Se já foi pedido para inserir um registo, temos que guardar os valores da inserção anterior (já validados) antes de permitir uma nova inserção. Os valores inseridos são guardados num novo registo adicionado à *DataTable* `registos`; um novo registo com esses valores é adicionado também ao *DataSet* `dados` para o utilizador poder ver os novos dados na *DataGrid* como se estes já estivessem guardados na base de dados. Uma vez guardados os valores da inserção anterior, pode-se então adicionar uma nova linha na *DataGrid*, atribuindo o modo de inserção ao modo corrente e chamando a função `bind_datagrid`.

Para finalizar as inserções clica-se no botão confirmar da barra de ferramentas das tarefas, que chama então a função `insere_dados_bd`. Esta função trata de guardar de forma permanente

os novos registos guardados até agora na *DataTable* `registos`, inserindo-os na base de dados. No entanto é necessário guardar em primeiro lugar os valores da inserção anterior como explicado atrás. A inserção na base de dados dos vários registos guardados na *DataTable* `registos` passa pela criação de uma instrução SQL da forma `"INSERT INTO tabela VALUES ..."` para cada um desses registos e pela execução da transacção composta por estas *queries*. O processo de inserção é finalizado pela actualização do *DataSet* `dados` a partir da base de dados e pela actualização da *DataGrid* ao preenchê-la com este *DataSet* chamando a função `bind_datagrid` em modo selecção (a primeira linha da *DataGrid* aparece seleccionada fixando o valor 0 à propriedade *SelectedIndex*).

As funcionalidades de alteração, remoção e procura seguem uma lógica semelhante.

Ao clicar no botão alterar que se encontra na barra de ferramentas das tarefas é chamada a função `inicializa_alterar` que trata de inicializar o processo de alteração. Esta inicialização passa inicialmente pela criação de uma nova *DataTable* chamada `registos` que guarda temporariamente os registos alterados até estas actualizações serem reflectidas de forma permanente na base de dados; a seguir, se a *DataGrid* já tinha alguma linha seleccionada antes de clicar no botão alterar, então chama-se a função `alterar_dados`. Mas se a *DataGrid* ainda não tinha nenhuma linha seleccionada então só na selecção de uma linha na *DataGrid* é que é chamada a função `alterar_dados`.

Salienta-se que como para a inserção é possível efectuar múltiplas alterações de uma só vez, isto é podem-se fazer vários pedidos de alteração seleccionando sucessivamente na *DataGrid* as linhas correspondentes aos registos a alterar, e só no fim finalizar as alterações efectuadas clicando no botão confirmar.

A função `alterar_dados` determina num primeiro tempo se já houve ou não um pedido de alteração.

Se não houve ainda pedidos de alteração é editada a linha seleccionada na *DataGrid*, mudando o modo corrente de selecção para alteração, fixando o índice da linha editada como o índice da linha seleccionada e chamando a função `bind_datagrid`.

Se já foi pedido para alterar um registo, temos que guardar os valores da alteração anterior (já validados) antes de permitir uma nova alteração. Os valores alterados são guardados num novo registo adicionado a *DataTable* `registos`; o registo do *DataSet* `dados` correspondente ao registo alterado é actualizado com os novos valores, para o utilizador poder ver as actualizações feitas na *DataGrid* como se estas já estivessem guardadas na base de dados. Uma vez guardados os valores da alteração anterior, pode-se então editar a linha seleccionada para alterar na *DataGrid*.

Para finalizar as alterações clica-se no botão confirmar da barra de ferramentas das tarefas, que chama então a função `altera_dados_bd`. Esta função trata de reflectir de forma permanente na base de dados as alterações feitas até agora e registadas de forma temporária na *DataTable* `registos`. No entanto, é necessário guardar em primeiro lugar os valores da alteração anterior como explicado atrás. A alteração na base de dados dos vários registos guardados na *DataTable* `registos` passa pela criação de uma instrução SQL da forma `" UPDATE tabela SET ... WHERE ... "` para cada um desses registos e pela execução da transacção composta por estas *queries*. O processo de alteração é finalizado pela actualização do *DataSet* `dados` a partir da base de dados e pela actualização da *DataGrid* ao preenchê-la com este *DataSet* chamando a função `bind_datagrid` em modo selecção.

Ao clicar no botão remover que se encontra na barra de ferramentas das tarefas é chamada a função `inicializa_remover` que trata de inicializar o processo de remoção. Esta inicialização passa inicialmente pela criação de uma nova *DataTable* chamada `registos` que guarda temporariamente os registos a remover, até estes serem efectivamente removidos de forma permanente da base de dados; a seguir, se a *DataGrid* já tinha alguma linha seleccionada antes de clicar no botão remover então é chamada a função `remover_dados`. Mas, se a *DataGrid* ainda não tinha nenhuma linha seleccionada então só na selecção de uma linha na *DataGrid* é que é chamada a função `remover_dados`.

Salienta-se que, tal como acontece para a inserção e a alteração, é possível efectuar múltiplas remoções de uma só vez, isto é, podem-se fazer vários pedidos de remoção seleccionando sucessivamente na *DataGrid* as linhas correspondentes aos registos a remover, e só no fim finalizar os pedidos de remoção clicando no botão confirmar.

A função `remover_dados` determina num primeiro tempo se já houve ou não um pedido de remoção.

Se não houve ainda pedidos de remoção, verifica-se primeiro se o registo correspondente a linha seleccionada na *DataGrid* pode ser removido da base de dados. Esta verificação passa por um teste à base de dados: se for possível executar com sucesso a transacção constituída pela instrução SQL da forma " `DELETE FROM tabela WHERE ...` ", usada para a remoção do registo em questão, então a remoção do registo será considerada como viável. Qualquer que seja o resultado da execução da *query* é efectuado um ROLLBACK a transacção para que a remoção não toma ainda efeitos na base de dados. Caso o registo possa ser removido da base de dados, este é adicionado a *DataTable* `registos`.

Se já foi pedido para remover pelo menos um registo, temos que determinar primeiro se o registo seleccionado não faz parte já dos pedidos de remoção anteriores (o que significaria que o utilizador já não pretende remover o registo). Caso o registo já tinha sido seleccionado para remover, anula-se o pedido de remoção para este registo apagando-o da *DataTable* `registos`, senão é um novo pedido de remoção e segue-se então os passos descritos atrás: verifica-se se o registo correspondente a linha seleccionada na *DataGrid* pode ser removido da base de dados e caso seja possível, este é adicionado a *DataTable* `registos`.

Para finalizar os pedidos de remoção clica-se no botão confirmar da barra de ferramentas das tarefas, que chama então a função `remova_dados_bd`. Esta função trata de executar de forma permanente na base de dados os pedidos de remoção registados de forma temporária na *DataTable* `registos`. A remoção da base de dados dos vários registos guardados na *DataTable* `registos` passa pela criação de uma instrução SQL da forma " `DELETE FROM tabela WHERE ...` " para cada um desses registos e pela execução da transacção composta por estas *queries*. O processo de remoção é finalizado pela actualização do *DataSet* `dados` a partir da base de dados e pela actualização da *DataGrid* ao preenchê-la com este *DataSet* chamando a função `bind_datagrid` em modo selecção.

Nestas três funções é preservada a integridade dos dados da base de dados, finalizando a transacção por um COMMIT no caso de esta ser executada sem problemas ou por um ROLLBACK caso contrário. Assim é garantida a propriedade transaccional em que ou todos os registos são removidos da base de dados com sucesso ou então nenhum deles é removido em caso de falha.

Ao clicar no botão **procurar** que se encontra na barra de ferramentas das tarefas é chamada a função `procurar_dados`. Esta limita-se apenas a limpar os registos da *DataGrid* e a inserir lá

uma nova linha em branco para permitir a inserção pelo utilizador dos valores da procura, fixando o modo corrente para o modo procura e chamando a função `bind_datagrid`.

Depois de introduzir os valores da procura, a procura dos registos determinados por estes valores é efectuada clicando no botão confirmar da barra de ferramentas das tarefas, que chama neste contexto a função `procura_dados_bd`. Esta função actualiza o *DataSet* `dados` a partir da base de dados, filtrando os dados a partir dos valores inseridos para a procura. Para terminar, a *DataGrid* é preenchida com este *DataSet* ao chamar a função `bind_datagrid` em modo selecção.

Se o utilizador decidir em qualquer momento abortar o processo de inserção, alteração, remoção ou procura, basta clicar no botão cancelar da barra das tarefas, que chama a função `cancelar_dados`. Esta função faz com que não sejam reproduzidos na base de dados as modificações aos registos guardados até agora na *DataTable* `registos`, limitando-se apenas a actualizar o *DataSet* `dados` a partir da base de dados (apesar da não confirmação das modificações efectuadas pelo utilizador, outros utilizadores poderão ter modificados em paralelo os registos da base de dados pelo que é necessário manter actualizada a informação mostrada na *DataGrid*) e a preencher a *DataGrid* com este *DataSet*.

b) Usando um *Formulário*

Para representar a tabela principal na forma de um *Formulário*, recorreu-se apenas a um tipo de controlo ASP: o controlo *Table*. Como para a implementação do tipo de objecto Grid, o controlo *Table* associado ao tipo de objecto Formulário é construído a partir do *DataSet* com os metadados e é preenchido a partir do *DataSet* com os dados.

Relativamente à construção das células do controlo *Table*, a cada registo do *DataSet* `metadados` correspondem duas novas células sucessivas da *Table*, em que a primeira contém um controlo *Label* com a designação do campo associado a este registo e a segunda contém um controlo *TextBox* que irá conter os dados relativos ao campo.

No fim obtém-se um formulário com uma *Label* e uma *TextBox* para cada um dos campos da tabela principal a representar:

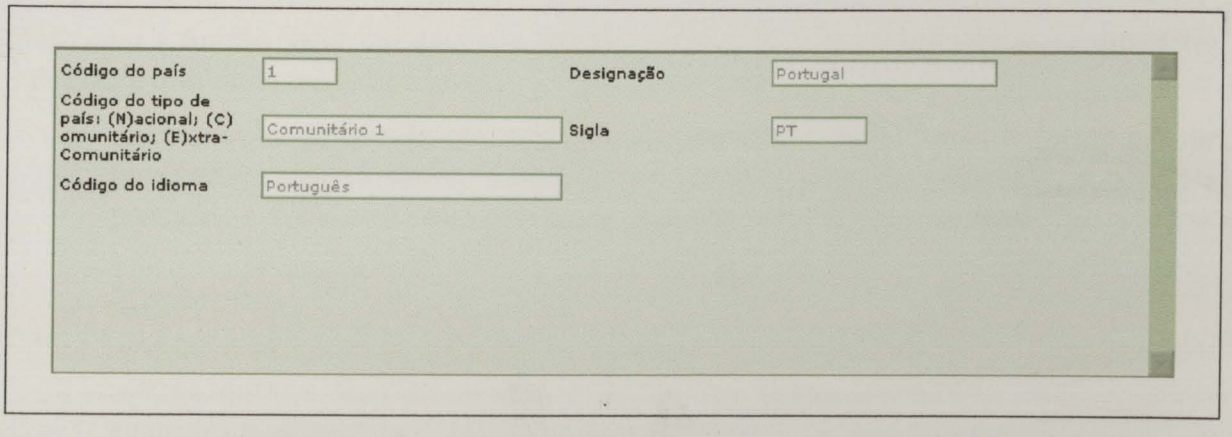


Figura 22 – Controlo *Table* associado ao tipo de objecto *Formulário*

Como o leitor poderá já ter antecipado, o *Formulário* só permite mostrar um único registo do *DataSet* `dados` de cada vez, ao contrário da Grid que permite visualizar todos os registos de uma só vez. De facto o *Formulário* mostra apenas o registo corrente seleccionado no *DataSet*.

Para seleccionar outro registo como registo corrente foram criados quatro botões auxiliares representados na figura seguinte.

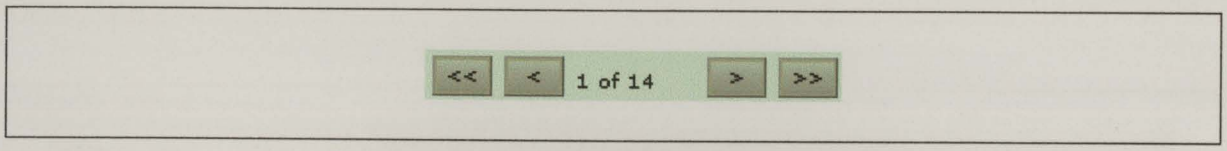


Figura 23 – Botões auxiliares ao Formulário

Estes permitem a movimentação para trás ou para frente dentro do *DataSet* `dados`. O primeiro botão permite aceder ao registo inicial do *DataSet*, o segundo ao registo imediatamente antes do actual, o terceiro ao registo imediatamente a seguir e o quarto ao último registo do *DataSet*. Salienta-se que existe também, juntamente com os botões auxiliares, informação numa *Label* sobre a posição do registo corrente em relação ao número total de registos do *DataSet*. Por exemplo, na figura, podemos observar que o registo corrente visualizado no formulário é o primeiro dos 14 registos do *DataSet*.

Enquanto que na *Grid* a selecção de registos consistia em seleccionar uma linha da *DataGrid* clicando num dos *links* presentes na linha (para cada chave primaria), aqui no *Formulário* a selecção de registos está associada ao clique em qualquer um dos botões auxiliares descritos atrás.

Os dados são carregados na *Table* através de uma função complexa designada `encheforms` que vou passar a descrever sucintamente e que é organizada de maneira muito semelhante à função `bind_datagrid` usada para o controlo *DataGrid*.

A função `encheforms` recebe como argumentos a *Table* a preencher, o *DataSet* com os dados da tabela principal, o *DataSet* com os metadados da tabela, o campo de ordenação e respectiva ordem de ordenação, as chaves primárias e estrangeiras da tabela, o *owner* da tabela, o modo operacional actual (inserção, alteração, procura, etc.) e o índice do registo corrente a mostrar no *Formulário*.

Tal como a função `bind_datagrid`, a função `encheforms` tem comportamentos diferentes consoante o valor do modo operacional:

- Assim, em modo de selecção, a função limita-se a criar uma *DataView* (vista) a partir do *DataSet* com os dados, ordenada pelo campo/ordem de ordenação, e depois a preencher a Table com esta vista usando a função auxiliar enchevalores.

```
DataView myView = new DataView();
myView = dados.Tables[0].DefaultView;
myView.Sort = SortExpression + " " + SortMode;
enchevalores(myView, ref tab, i);
```

A função `enchevalores` consiste em preencher as *TextBoxes* relativas a cada campo com os valores do registo desta vista correspondente ao registo corrente (determinado pelo índice passado como parâmetro).

```
for (int j = 0; j < view.Table.Columns.Count; j++) {
    TextBox t = (TextBox)tab.FindControl("TEXT" + j.ToString());
    t.Text = view[i][j].ToString();
}
```

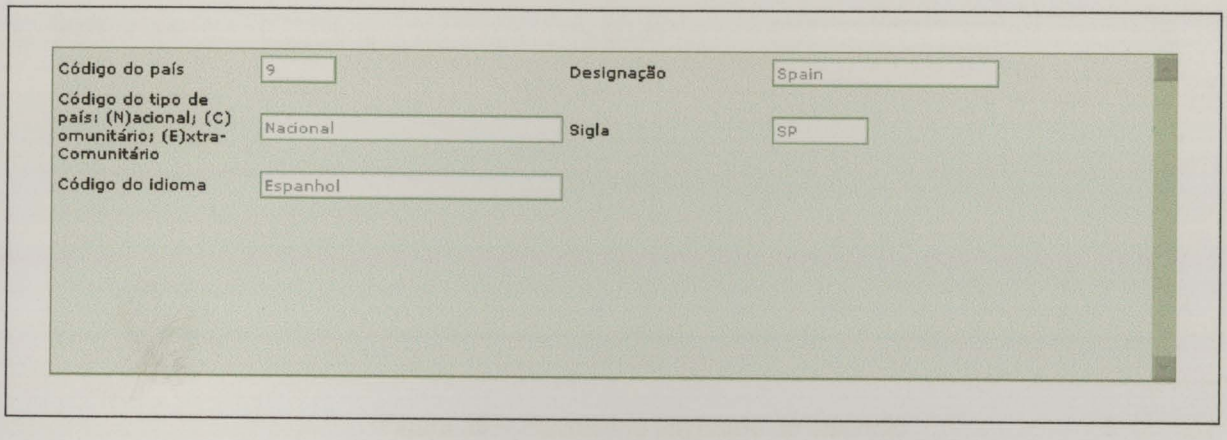


Figura 24 – *Formulário* em modo de selecção

- Caso o modo seja de alteração, a *Table* é preenchida com a vista descrita anteriormente, tendo em atenção que o índice do registo corrente passado como parâmetro corresponde ao índice do registo a editar. A seguir actualiza-se as células da *Table* com as *TextBoxes*, modificando se necessário as *TextBoxes* para outro tipo de controlo mais adequado. A certas células adiciona-se também botões que trazem funcionalidades complementares (botões que abrem uma janela com um calendário para as datas ou com uma lista de valores para as chaves estrangeiras, etc.). Assim por exemplo, para as células correspondentes a campos com o papel de chave estrangeira, a *TextBox* é substituída por uma *ComboBox* preenchida dinamicamente a partir dos dados da tabela relacionada com o campo e é adicionado à célula um botão de LOV (*List Of Values*) que permite escolher o item numa grelha onde o utilizador pode filtrar e ordenar os dados.

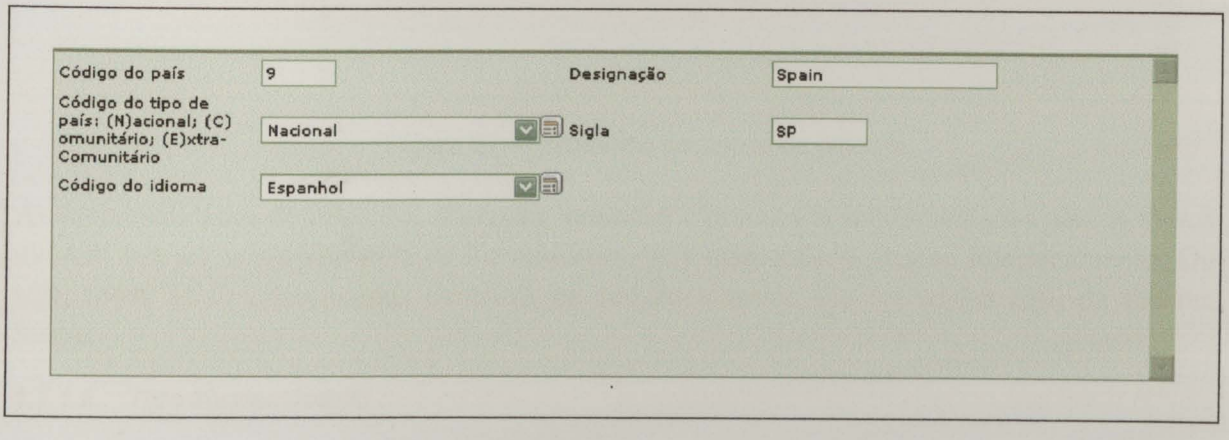


Figura 25 – *Formulário* em modo de alteração

- Caso o modo seja de inserção, adiciona-se inicialmente um novo registo ao fim da vista descrita atrás e modifica-se o índice do registo corrente para o índice deste novo registo. Depois preenche-se a *Table* com esta vista, tendo em atenção que o índice do registo corrente corresponde agora ao índice do registo inserido. A seguir actualiza-se

as células da *Table* com as *TextBoxes*, modificando se necessário as *TextBoxes* para outro tipo de controlo mais adequado e/ou adicionando botões para LOV ou calendário.

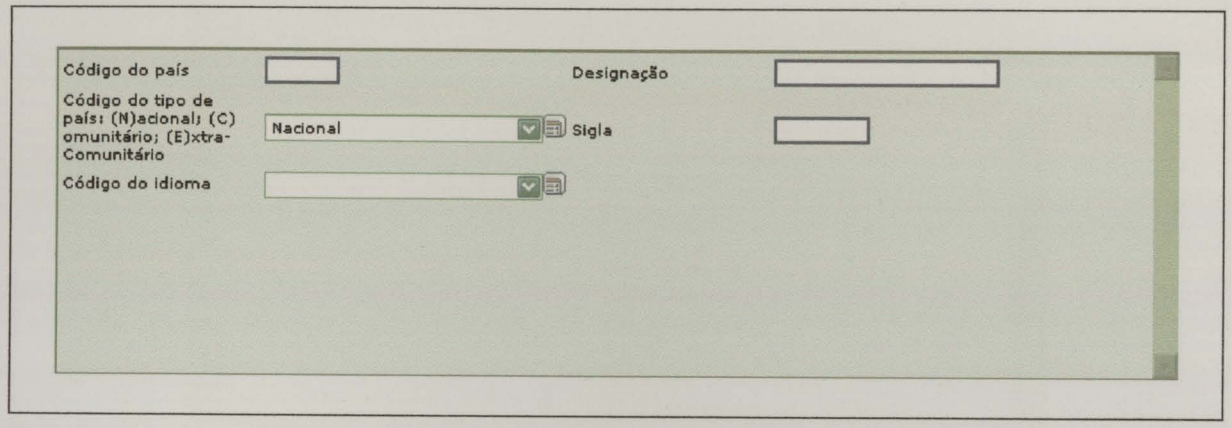


Figura 26 – *Formulário* em modo de inserção

- Caso o modo seja de procura, cria-se primeiro uma nova vista com um único registo sem valores e modifica-se o índice do registo corrente para o índice deste registo vazio, isto é o índice 0. Depois preenche-se a *Table* com esta vista e actualiza-se as células da *Table* associadas aos campos com o papel de chave estrangeira, adicionando-lhes um botão de LOV para facilitar a introdução de valores de procura.

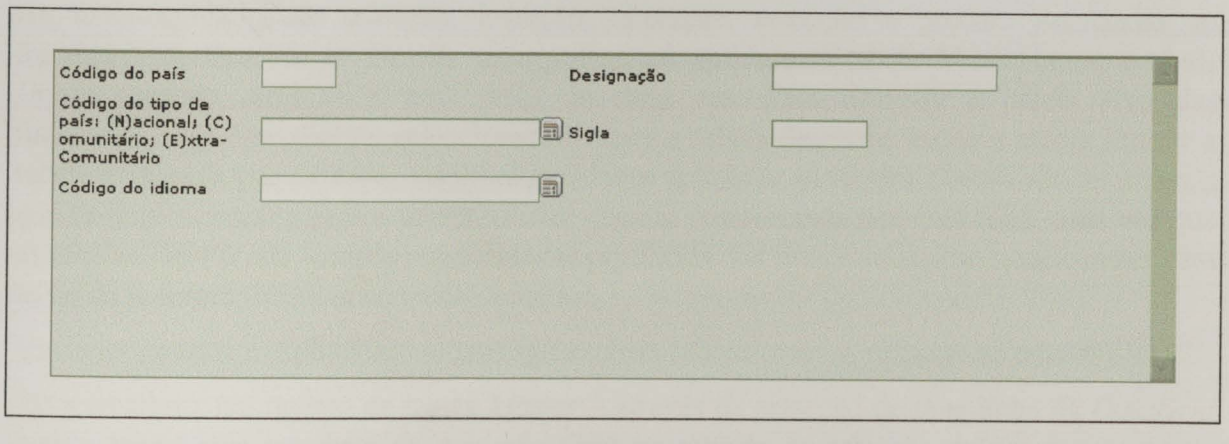


Figura 27 – *Formulário* em modo de procura

As funcionalidades de inserção, alteração, remoção e procura já implementadas para as *Grids* tiveram que ser adaptadas para os *Formulários*, mas mantendo o mesmo funcionamento. Ou seja, todas as funcionalidades descritas na secção anterior são realizadas aqui da mesma forma.

2.7.1.2 Tipo Master/Details

Nesta secção descrevo os passos seguidos na construção do formulário do tipo *Master/Details* em que a tabela principal é referenciada em outras tabelas aplicacionais.

O formulário *Master/Details* representa a tabela *Master* na forma de uma *Grid* ou de um *Formulário* consoante o tipo de objecto escolhido para a sua representação e representa as tabelas *Details* na forma de Grids situadas por baixo da tabela *Master*. Só sendo mostrada no

ecrã uma tabela *Detail* de cada vez, existe no topo das tabelas *Details* um painel de separadores que permite mudar de tabela *Detail* se necessário.

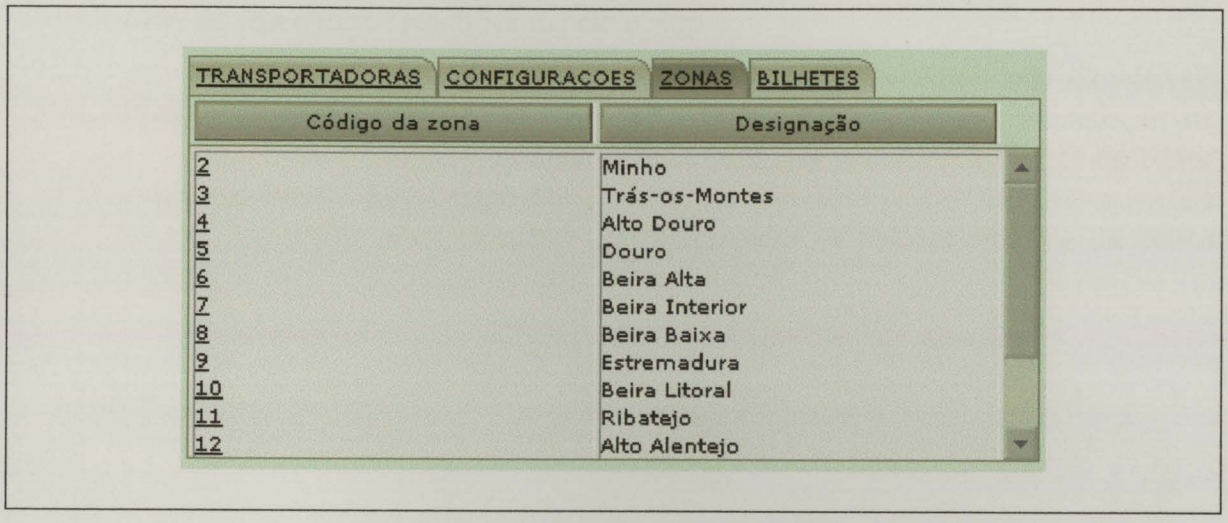


Figura 28 – Painel de separadores e tabelas *Details*

A *DataGrid* associada a tabela *Detail* é construída e preenchida da mesma forma que a *DataGrid* associada a tabela *Master*. No entanto, os *DataSets* com os metadados e os dados a considerar dizem aqui respeito a tabela *Detail* (i.e. `metadados_detail` e `dados_detail`), sendo o *owner* da tabela *Detail* o mesmo que o da tabela *Master*.

As funcionalidades de selecção, inserção, alteração, remoção e procura associadas ao formulário do tipo *Master/Details* tanto podem ser aplicadas a tabela *Master* como a tabela *Detail* corrente, mas só a uma delas de cada vez. Para designar a tabela alvo das funcionalidades é necessário seleccionar primeiro a tabela desejada, sendo a tabela *Master* a tabela designada por omissão. As funcionalidades aplicadas as tabelas *Details* são as mesmas que as implementadas para a tabela *Master* quando representada por uma Grid, uma vez que as tabelas *Details* são sempre representadas por Grids. As funcionalidades mencionadas para as Grids já foram descritas na secção anterior.

Vou agora passar a explicar como interage a tabela *Master* com as respectivas tabelas *Details*.

Ao seleccionar um registo da tabela *Master* – através da selecção de uma linha da *DataGrid* caso a tabela seja representada por uma Grid ou através da selecção de um registo como corrente na *Table* caso a tabela seja representada por um Formulário – é necessário actualizar a informação mostrada na tabela *Detail* actual. De facto, ao seleccionar um registo da tabela *Master* devem aparecer na *DataGrid* representando a tabela *Detail* apenas os registos relacionados com o registo seleccionado.

Esta ligação é fácil de implementar uma vez que a tabela *Master* e a tabela *Detail* estão relacionadas através de campos relacionais (campos com o papel de chave estrangeira) existentes na tabela *Detail*, cujo valor corresponde sempre a um dos valores das chaves primárias existentes na tabela *Master*. Mais concretamente, ao seleccionar um registo diferente na tabela *Master* (identificado pelo valor único das suas chaves primarias) é actualizado o *DataSet* `dados_detail` usado para preencher a *DataGrid* representando a tabela *Detail* corrente. Os registos do *DataSet* determinados a partir da base de dados são filtrados pelos campos relacionais, cujo valor deve corresponder ao valor das chaves primárias identificando o registo seleccionado na *Master*.

Por exemplo, na figura seguinte, a tabela *Master* designada **BANCOS** encontra-se associada à tabela *Detail* designada **BALCOES**. Ao seleccionar o primeiro registo da tabela *Master* são filtrados os registos da tabela *Detail*, fazendo aparecer apenas os balcões relativos ao banco seleccionado, ou seja o banco identificado pelo código 1.

Nota-se que os campos da tabela *Detail* relacionados com a tabela *Master* não são visíveis para o utilizador de maneira a não haver repetição de informação. Assim, por exemplo, existe um campo como chave estrangeira na tabela BALCOES que identifica o código do banco correspondente ao balcão. Este campo não é visível ao utilizador porque sabemos à partida que os balcões mostrados neste momento são unicamente os balcões relativos ao banco seleccionado na tabela BANCOS (neste caso, o banco BCP).

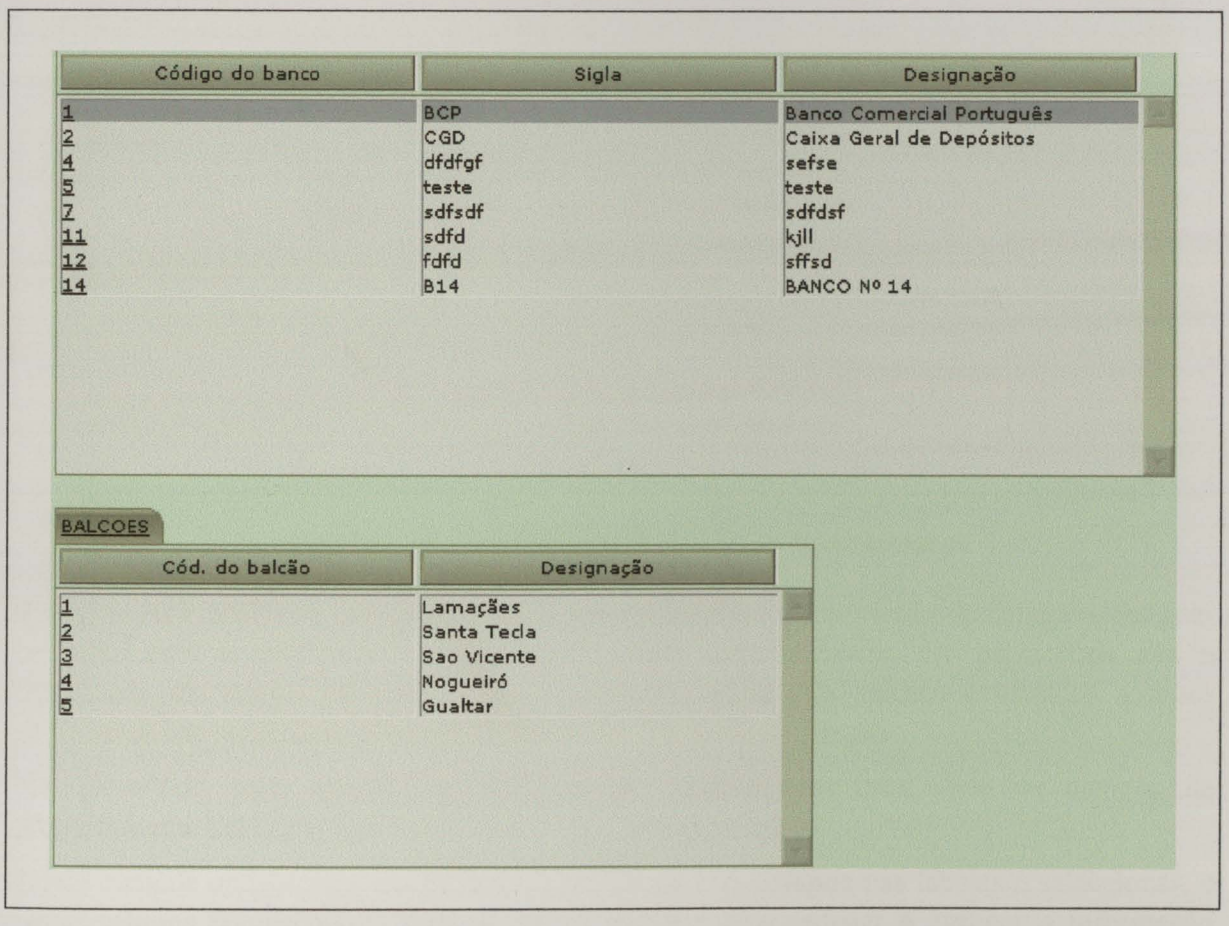


Figura 29 – Interacção entre a tabela *Master* e a tabela *Detail*

2.7.2 *Formulário genérico para criação de listagens*

O formulário genérico para criação de listagens permite criar novas listagens a partir da informação presente no formulário actual (simples) para manutenção de dados.

2.7.2.1 *Criação da listagem*

Ao clicar no botão das listagens presente na barra de tarefas abre-se um formulário para criação de listagens sobre os dados do formulário actual.

O formulário para criação de listagens é constituído por uma lista de controlos que são usados para o utilizador escolher a informação a mostrar na listagem.

O utilizador pode seleccionar qualquer um dos campos de qualquer umas das tabelas que se encontram no formulário actual para manutenção de dados, efectuar operações de agregação sobre os campos seleccionados, e apresentar a informação ordenada, agrupada e filtrada.

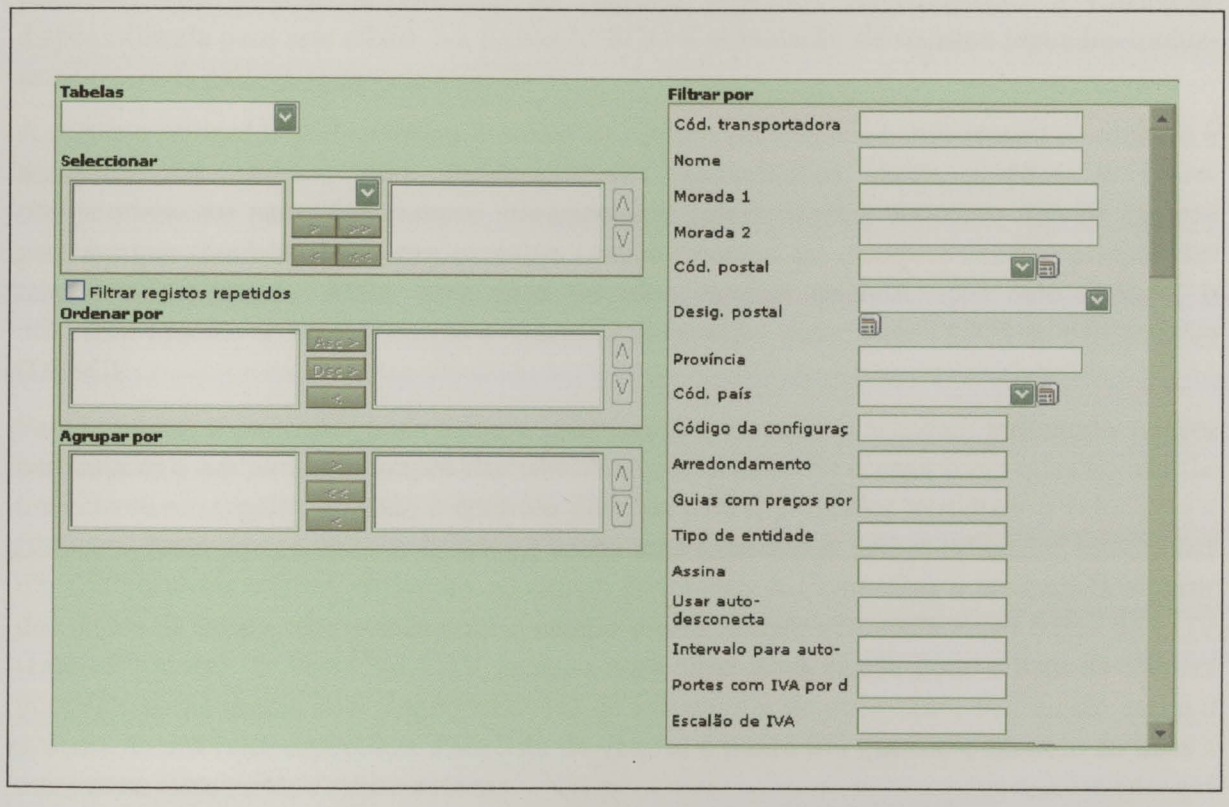


Figura 30 – Formulário para criação de novas listagens

A página ASP.NET que implementa o formulário genérico para criação de listagens designa-se select.aspx e apresenta-se como uma página parametrizada. Os parâmetros são a designação da tabela principal e o *owner* dessa tabela. Os valores associados a esses parâmetros são recebidos pela página através de Variáveis de Sessão.

No formulário para criação de listagens são apresentados uma série de campos de preenchimento/selecção, uns facultativos e outros obrigatórios.

Como campos obrigatórios consideramos as tabelas e os campos das tabelas a seleccionar, e como campos facultativos consideramos os campos para ordenar e agrupar a informação assim como os valores para filtrar os dados.

Relativamente ao campo referente às tabelas, o utilizador escolha dentro de uma *ComboBox* a ou as tabelas das quais ele quer extrair a informação. As tabelas listadas são apenas a tabela principal, caso o formulário actual para manutenção de dados seja do tipo *Master*, ou a tabela *Master* mais as tabelas *Details*, caso o formulário seja do tipo *Master/Details*. Como para a página associada ao formulário genérico para manutenção de dados, a página select.aspx determina as tabelas *Details* relativas a tabela *Master* à partir da informação registada na tabela BDI_TABPAI da base de dados BDINET.

Uma vez escolhidas a ou as tabelas, o utilizador selecciona dentro de uma lista – usando o controlo *ListBox* – os campos das tabelas (anteriormente seleccionadas) a usar como parâmetros para a listagem.

Ainda é possível usar funções de agregação sobre os campos seleccionados. Estas funções são: Sum, Count, Max, Min, Avg e são disponibilizadas ao utilizador mais uma vez através de uma *ComboBox*.

Refere-se que é possível eliminar os registos repetidos seleccionando a *CheckBox* disponibilizada para este efeito. Na instrução SQL, a eliminação de registos repetidos traduz-se pelo uso da palavra reservada *DISTINCT*.

A seguir o utilizador pode também escolher os campos pelos quais a informação é ordenada e agrupada. Os campos para ordenação são apresentados numa *ListBox* e fazem obrigatoriamente parte dos campos seleccionados anteriormente, enquanto que os campos para agrupar (também apresentados numa *ListBox*) podem ser qualquer um dos campos das tabelas seleccionadas. Mais, para cada um dos campos de ordenação seleccionado, o utilizador escolhe ao mesmo tempo a ordem da ordenação: ascendente (ASC) ou descendente (DESC).

Para terminar, o utilizador pode filtrar a informação a mostrar na listagem atribuindo valores particulares a um ou mais campos das tabelas seleccionadas. Se o campo em questão não for uma chave estrangeira, é usado o controlo *TextBox* para o utilizador introduzir o valor para a filtragem deste campo; caso contrário, é usado uma *ComboBox* para o utilizador seleccionar um dos possíveis valores atribuídos ao campo relacional. A *ComboBox* é preenchida a partir dos dados da tabela relacionada com o campo e está sempre associada a um botão de LOV (List of Values). O botão de LOV permite abrir uma nova janela com a lista de valores possíveis numa grelha com funcionalidades de filtragem e de ordenação, facilitando assim a procura de um valor específico. Esta lista de valores é muito útil quando o número de itens a mostrar na *ComboBox* é muito extenso.

2.7.2.2 Geração da listagem

Uma vez introduzidos os dados necessários no formulário para criação de listagens, o utilizador gera a listagem correspondente clicando uma segunda vez no botão das listagens presente na barra de tarefas.

A geração de listagem faz uso de uma *query* determinada a partir da informação introduzida pelo utilizador no formulário para criação de listagens. A instrução SQL associada a esta *query* é construída a partir das escolhas feitas no formulário, e pode ser mais ou menos complexa consoante o tipo do formulário actual para manutenção de dados. De facto, se este for do tipo *Master/Details* o utilizador pode, no extremo, relacionar todas as tabelas entre si, o que resulta na utilização de vários *INNER JOINS* para a junção das tabelas pelos campos relacionais.

A instrução SQL tem a forma:

```
SELECT {1}
FROM {2}
[WHERE {3}]
[GROUP BY {4}]
[ORDER BY {5}]
```

Onde:

```
{1}  ⇔  campos a seleccionar dentro das tabelas seleccionadas
{2}  ⇔  tabelas seleccionadas
{3}  ⇔  restrições subjacentes à filtragem dos campos
```

```
{4}  ⇔  campos para agrupar os dados
{5}  ⇔  campos para ordenar os dados
[ ]  significa que é facultativo
```

Os campos {1}, {2}, {3}, {4} e {5} da instrução SQL são determinados a partir dos cinco campos de preenchimento/selecção apresentados no formulário para criação de listagens (ver figura 35) e designados respectivamente Seleccionar, Tabelas, Filtrar por, Agrupar por, Ordenar por.

Uma vez determinada a instrução SQL relativa à *query* da listagem, esta é usada para construir o *DataSet* com os dados da listagem. Este *DataSet* é obtido usando a função **Querie** que consiste em executar na base de dados um comando com uma instrução SQL:

```
public DataSet Querie(string p_query, string p_owner)
{
        string sqlquery;
        DataSet db = new DataSet();
OleDbConnection l_owner = new OleDbConnection(ConfigurationSettings.
AppSettings.Get(p_owner));
        l_owner.Open();
        OleDbCommand cmd = new OleDbCommand(p_query, l_owner);
        db = UtilsBD.executaQuery(l_owner, cmd, null);
        l_owner.Close();
        return db;
}
```

Daí é construída uma *DataGrid* que é preenchida com o *DataSet* obtido, para o utilizador poder visualizar a listagem resultante na forma de uma grelha.

2.7.2.3 *Exportação da listagem*

A listagem visualizada pode ser imprimida, enviada por e-mail ou exportada para os formatos PDF, RTF e XLS correspondentes aos programas Acrobat, Word e Excel respectivamente, clicando no botão adequado da barra de operações. A exportação da informação da listagem para estes formatos é efectuada usando o *DataSet* com os dados da listagem, que foi previamente determinado na geração da listagem.

2.8 Estado Actual

No seu estado actual, a aplicação permite a criação de um enorme número de formulários simples, desde que sejam previamente definidos na BDINET.

Está criada uma base que permite de uma forma segura partir para o objectivo inicial do projecto, a migração de todo o sistema de informação aplicacional Wine-Cellar IS, não só pelo estado em que ficou o projecto mas também pelo conhecimento adquirido. Desta forma interessa salientar os pontos negativos e positivos da aplicação neste momento.

Globalmente os pontos negativos do projecto constam na degradação da performance da aplicação. Esta é causada em grande parte, de um lado pelos "*refresh*" efectuados à página devido à validação dos dados ser feita do lado do servidor e à actualização constante da *scrollbar* associada às *grids*, e de outro lado pelo uso de variáveis de sessão que guardem muita informação em memória (*DataSets* por exemplo). Estes pontos importantes para a

performance da aplicação serão revistos na altura em que se efectuará optimizações à arquitectura genérica.

Podemos ainda destacar como ponto negativo as poucas medidas de segurança tomadas para a aplicação. De momento só é efectuada a encriptação das *passwords* dos utilizadores, usando o algoritmo "DES" como algoritmo de encriptação. Falta nomeadamente tratar da encriptação das passwords de conexão às bases de dados suportadas pela aplicação. O ficheiro de configuração para as aplicações .NET em ambiente web, designado web.config, permite configurar muitos aspectos gerais de segurança, ajudando assim numa melhor performance e estabilidade das aplicações. Salienta-se que actualmente a informação com as conexões às bases de dados é guardada e acedida através do ficheiro web.config.

Outro ponto negativo deste projecto é a falta de rigidez no tratamento de erros e consequentemente uma difícil detecção da causa de erros que possam surgir.

O tratamento de erros é crucial em qualquer aplicação. De forma a facilitar o *debugging* e a resolução de problemas em ambientes de produção é muito importante que as excepções sejam tratadas devidamente em toda a aplicação. Neste momento, a aplicação não está tão rígida quanto isso relativamente a este aspecto. Por exemplo, as excepções detectadas deveriam ser escritas para *trace*, uma vez que em ambientes de qualidade e produção não está disponível qualquer tipo de *debugging*. Os *traces* (ou *logs*) quando bem aplicados são um recurso que permite poupar horas de trabalho e testes adicionais (que quase sempre têm que ser feitos em cima da hora no ambiente de desenvolvimento). O ficheiro web.config permite a configuração do serviço de *trace* das ASP.NET assim como a definição de páginas de erro standard em aplicações *web*.

Para terminar, a estrutura da solução criada poderia ser melhorada em termos de organização.

A estrutura da solução deveria ser reorganizada de maneira a focar os pontos seguintes:

- Optimizar o desenvolvimento dos programadores
- Uniformizar a arquitectura de projectos
- Simplificar a instalação noutros ambientes (ex.: produção)
- Facilitar futuras integrações com outras aplicações
- Reaproveitamento de módulos do projecto para futuros desenvolvimentos
- Organização do código desenvolvido

O modelo de arquitectura que foi sugerido para melhoria da arquitectura actual é o seguinte:

1. Solution

A solução corresponde a raiz da aplicação e englobe todos os sub-projectos existentes na aplicação.

2. Projecto Serviços

Este sub-projecto é do tipo *Class library* e aqui são agrupadas todas as classes específicas para a aplicação (ex: acesso a dados).

3. Projecto WebServices (opcional) → nesta aplicação é desnecessário

Tal como o nome indica, este sub-projecto é do tipo *WebServices*. Aqui são guardados os WebServices que disponibilizam métodos para serem acedidos via HTTP.

4. Projecto Web/Win (Web ou WinForms) → esta aplicação é do tipo Web

Neste sub-projecto fica tudo o que diz respeito à camada de apresentação da aplicação (ex: páginas, ficheiro de estilo, ficheiros JS, imagens, etc.)

5. Projecto Comuns

Este sub-projecto é do tipo *Class Library* e contém os objectos que são comuns a outros projectos e que podem ser facilmente integrados noutras aplicações (ex: utils, security, classes).

Até agora enfatizamos apenas os pontos negativos do projecto actual mas não podemos por de lado todos os seus pontos positivos.

Desses pontos podem-se destacar os seguintes:

- Os objectivos de cada fase concluída do projecto foram cumpridos com sucesso uma vez que seguem rigorosamente todos os requisitos de negócio definidos à partida.
- Até agora o sistema desenvolvido consegue conciliar a fidelidade ao produto existente em termos de funcionalidades oferecidas e a inovação pela criação de novas funcionalidades.
- A aplicação possui uma interface agradável e a sua utilização é bastante simples.
- Para terminar o facto da ferramenta funcionar num ambiente Web possibilita a sua utilização sem necessidade de instalação de software adicional pelos clientes.

De um modo geral pode-se dizer que, embora não acabado, este projecto foi bem sucedido até agora pois atingiu de forma satisfatória todos os objectivos propostos inicialmente.

2.9 Perspectivas de Trabalho Futuro

Como trabalho futuro está previsto terminar o protótipo deste projecto efectuando primeiro algumas optimizações à arquitectura genérica já desenvolvida e avançando depois com as fases restantes, i.e. a fase de migração da aplicação Gestão Comercial e a fase de migração dos restantes módulos da aplicação WIS (módulos das Vindimas, Lotes/Cubas, ...).

Só depois de uma revisão exaustiva (correcção de *bugs*) do sistema desenvolvido em paralelo com uma série de testes ao produto e a sua validação com os utilizadores, é que se poderá dar como terminado o projecto.

Uma vez concluído o projecto e avaliado o sucesso comercial do produto final obtido efectuar-se-á ou não a migração para a plataforma .NET das outras aplicações da EGAPI (Pedreiras, ...).

3 Projecto Accessu .NET

Da mesma forma que na anterior, nesta secção irei detalhar o âmbito do projecto Accessu .NET. Será feita inicialmente uma descrição da solução anterior, explicando sucintamente as suas funcionalidades e modo de funcionamento. De seguida apresentarei os requisitos que levaram a criação deste mesmo projecto e as suas condicionantes. Nos quatro pontos seguintes farei uma descrição detalhada de todo o trabalho desenvolvido, passando pela arquitectura da nova solução e módulos desenvolvidos. Finalmente, nos dois últimos pontos descrevo o seu estado actual e perspectivas de trabalho futuro.

3.1 Solução Anterior

O "Accessu"™ é um sistema de informação e equipamentos para suporte a processos de bilhética, controle de acessos e de presenças, desenvolvido, como o anterior projecto, em Oracle Forms para o ambiente *Desktop*, sendo, da mesma forma, todos os relatórios e documentação oficial produzidos em Oracle Reports.

Este sistema divide-se em dois grande módulos, Sistema de Controlo de Eventos e Gestão de Sócios e Quotas, que funcionam independentemente ou em conjunto.

O Sistema de Controlo de Eventos é um módulo que contempla a definição de instalações e as suas diversas componentes, definição de eventos e competições, venda de títulos de acesso, que no seu conjunto proporcionam a gestão eficaz do sistema de controlo de acessos.

Em funcionamento "*online*" com o sistema de monitorização e controle de acessos, permite que um título, imediatamente após a venda em bilheteira, seja validado numa tabela para o respectivo evento. É assim possível vender bilhetes em simultâneo para diversos eventos e, logo após a venda, o titular ter acesso imediato à instalação onde decorre o evento, sendo que, o título, devidamente parametrizada a instalação, tem definido a porta, a zona, o sector, a fila e lugar a que dá acesso.

A aplicação permite a venda de bilhetes e títulos de acesso, a venda de agrupamentos de eventos, a troca de lugares vendidos, a definição de diversos tipos de bilhetes com preços parametrizáveis e a definição e utilização de promoções para venda de bilhetes.

Podem também ser definidas várias empresas e vários locais, às quais podem ser associadas instalações e as receitas das vendas das mesmas.

O módulo de gestão de sócios e quotas tem como objectivo facilitar o acesso à informação de sócios. É constituída por um conjunto de tabelas onde nos é permitido introduzir valores que irão ser constantemente utilizados pela aplicação e que irão influenciar o seu funcionamento:

- Parametrização do Sistema de Informação de Sócios:
  - Situações, Profissões, Habilitações
  - Estados Civis, Graus de Parentesco, etc.
- Parametrização do Sistema de Quotização:
  - Tipo / Períodos de Quotas, Jóias
  - Categorias / Classes / Escalões de Quotas

- Tipo / Períodos de Pagamento de Quotas

Cada associado possui um cadastro completo de informação particular e relativa à sua conta corrente com o clube (quotas). Toda esta informação permite articular a parametrização de venda de "produtos" para eventos com essa informação, ou seja um sócio cujo cadastro apresente quotas em atraso não poderá adquirir produtos de "venda" afectos a associados.

O sistema de Gestão de Sócios, permite a gestão de múltiplas forma de pagamento: cobradores, transferência bancária, Multibanco, secretaria, créditos, etc. As funcionalidades relativas a pagamentos efectuados a entidades externas ao clube possuem processos de exportação e importação de dados dos pagamentos. Esses pagamentos podem ser de quotas, renumeração, renovação de agrupamentos de época e bilhetes para eventos.

São ainda funcionalidades deste sistema a Renumeração de associados e a parametrização das Assembleias Gerais do clube assim como o registo dos participantes nestas assembleias.

Como sub-módulo da Gestão de Sócios e Eventos existe o sistema de Gestão de Cartões. Este permite o controle do processo administrativo de emissão de cartões de associados. Além dos cartões de associados existem também os cartões emitidos aquando da venda de agrupamentos.

3.2 Requisitos e Condicionantes do projecto

Apesar de, na sua essência, o projecto Accessu .NET se centrar na migração de uma aplicação já existente, todo o projecto foi desenvolvido com o intuito de criar alguns protótipos das funcionalidades mais importantes do sistema já existente, de forma a ser possível avaliar a sua viabilidade.

Sendo assim, foram escolhidas as funcionalidades de inscrição de um novo sócio, o pagamento de quotas e a venda de bilhetes, estando esta última a meu cargo. Os requisitos funcionais foram, desta forma, reduzidos aos inerentes a estas três funcionalidades, sendo que, sempre que possível, se introduziram melhoramentos no seu funcionamento.

Já os requisitos não funcionais foram definidos numa forma muito semelhante ao projecto anterior. Na área deste projecto que me foi atribuída, houve um intuito de modificar o modo de venda de bilhetes, tornando o processo mais visual e intuitivo, ao contrário do existente.

Neste projecto, houve uma maior preocupação em criar uma solução que fosse independente da base de dados, tendo mesmo sido realizados alguns testes com a utilização de duas bases de dados diferentes.

Outros requisitos, já referidos no projecto anterior, como a *performance*, a visibilidade comercial do produto e a redução de custos foram também importantes no desenvolvimento do projecto.

3.3 Arquitectura da Nova Solução

Da mesma forma que o projecto anterior, a arquitectura definida pretendia minimizar a replicação de dados em diversos locais da empresa, mantendo a informação centralizada ao máximo, e minimizar o número de ferramentas necessárias para a execução da aplicação.

Mas, desta vez, disponibilizar o acesso à aplicação através de um browser web teve que ser posto de parte devido à interacção necessária com certos equipamentos, p. ex. a interacção

com uma impressora de bilhetes é realizada através da porta série (RS232) o que torna a sua utilização através de um *browser web* impossível. Optou-se então pela criação de *Smart Clients*.

Um *Smart Client* pode ser configurado de forma a tirar partido da facilidade de instalação e gestão de versões existentes numa aplicação do tipo *browser web*, ao mesmo tempo que permite uma melhor utilização dos recursos locais e criar uma interface mais rica, entre outras características.

No ponto seguinte irei descrever a arquitectura básica da nova solução.

3.3.1 Arquitectura

Neste projecto, coexistem duas arquitecturas, utilizadas em situações diferentes. Foi definida uma arquitectura para a instalação do *Smart Client* na máquina do utilizador e uma outra para a utilização da aplicação.

A arquitectura utilizada para a instalação do *Smart Client* tira partido dos conceitos *ClickOnce Deployment* disponibilizado pelo ambiente de desenvolvimento .NET. Esta funcionalidade permite disponibilizar a partir de um endereço Web a aplicação e todas as suas versões. Sendo assim, ao efectuar uma nova instalação da aplicação, é instalada a ultima versão disponível no sítio *web* e todas as instalações já realizadas verificam constantemente se existe alguma actualização da aplicação, podendo tornar este processo completamente transparente para o utilizador.

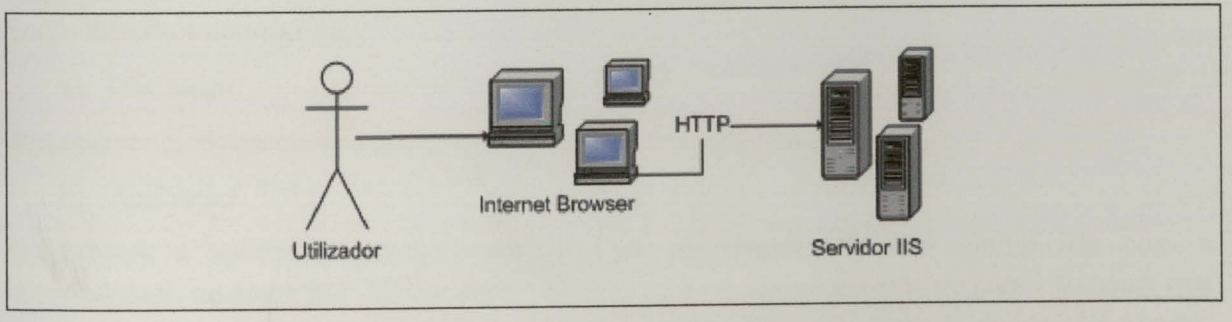


Figura 31 - Arquitectura básica da instalação da aplicação Accessu.NET

Na figura anterior podemos ver três nós que se descrevem de seguida:

a) Utilizador

Representa qualquer pessoa que possa interagir com o sistema.

b) Internet Browser

Tirando partido do *ClickOnce Deployment*, a instalação da aplicação é efectuada através do acesso a uma página Web. Sendo assim, um qualquer browser Web instalado na máquina do cliente permitirá instalar todos os componentes necessários para o correcto funcionamento da aplicação.

c) Servidor IIS

Não necessita de ter qualquer tipo de tecnologia instalada, uma vez que apenas irá disponibilizar o acesso a um pequeno utilitário que ajudará à instalação da aplicação.

Já a arquitectura disponibilizada para a utilização da aplicação retira partido dos conceitos de *Smart Client*. Esta aplicação poderá ser parametrizada de forma a poder ser utilizada *offline* ou sempre *online*. Dependendo desta opção será feita uma distribuição diferente de componentes descritos de seguida.

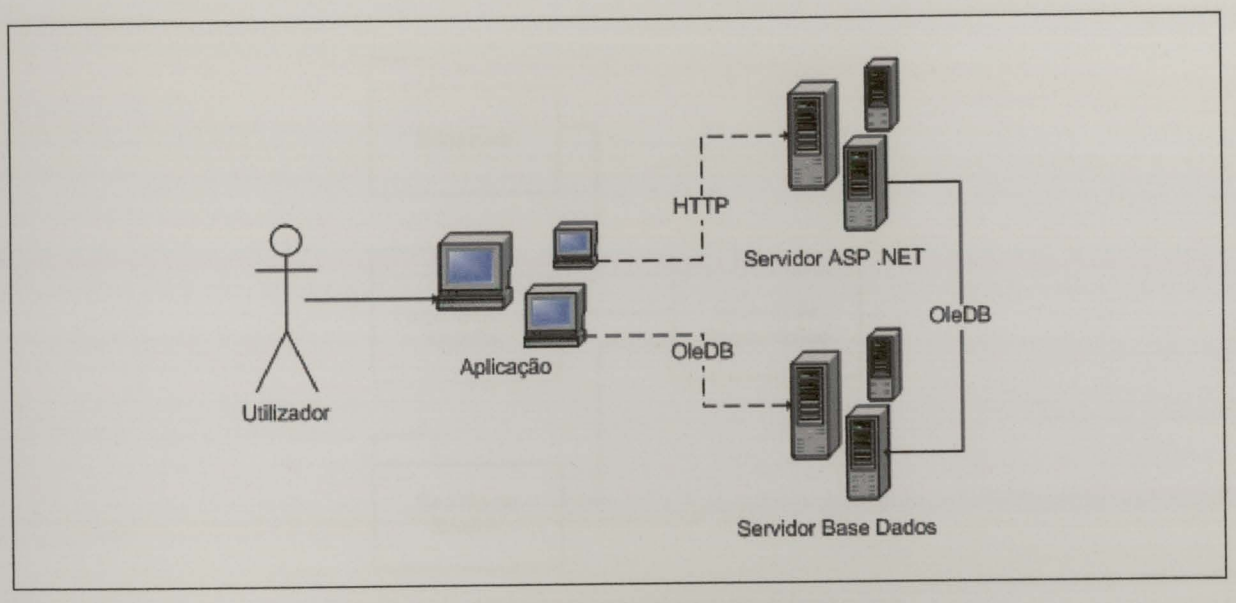


Figura 32 - Arquitectura básica da aplicação Accessu .NET

Na figura anterior estão desenhados os quatro nós que poderão existir nesta arquitectura e que serão descritos de seguida.

a) Utilizador

Representa qualquer pessoa que possa interagir com o sistema.

b) Aplicação

Representa a aplicação instalada na máquina do cliente. Se for configurada com a possibilidade de trabalhar *offline* será instalado com a aplicação um pequeno ficheiro que possa simular uma base de dados, seja ele um ficheiro XML, uma base de dados MS Access, uma base de dados SQL Mobile, entre outras. Nesta configuração também serão instaladas as *dlls* necessárias para realizar as funcionalidades de cada aplicação. A sincronização de dados com o servidor será realizada assim que exista ligação.

c) Servidor ASP.NET

Se for seleccionada uma configuração que funcione sempre *online*, poderá ser criado um servidor com diversos *web services* que representem as funcionalidades da aplicação, desta forma evitando sobrecarregar a aplicação com *dlls*, e garantindo uma maior consistência nas *dlls* utilizadas nos diversos postos onde a aplicação estiver instalada.

d) Servidor de Base de Dados

Neste último servidor estarão todas as bases de dados necessárias para a aplicação. A ligação a este servidor será feita através de *OleDB* de forma a permitir a utilização de diferentes bases de dados.

O número de servidores a utilizar depende, como no projecto anterior, unicamente da qualidade do serviço que se pretende oferecer.

3.3.2 *Módulos e integrações previstas*

Nesta fase está previsto o desenvolvimento de três funcionalidades, como foi referido inicialmente.

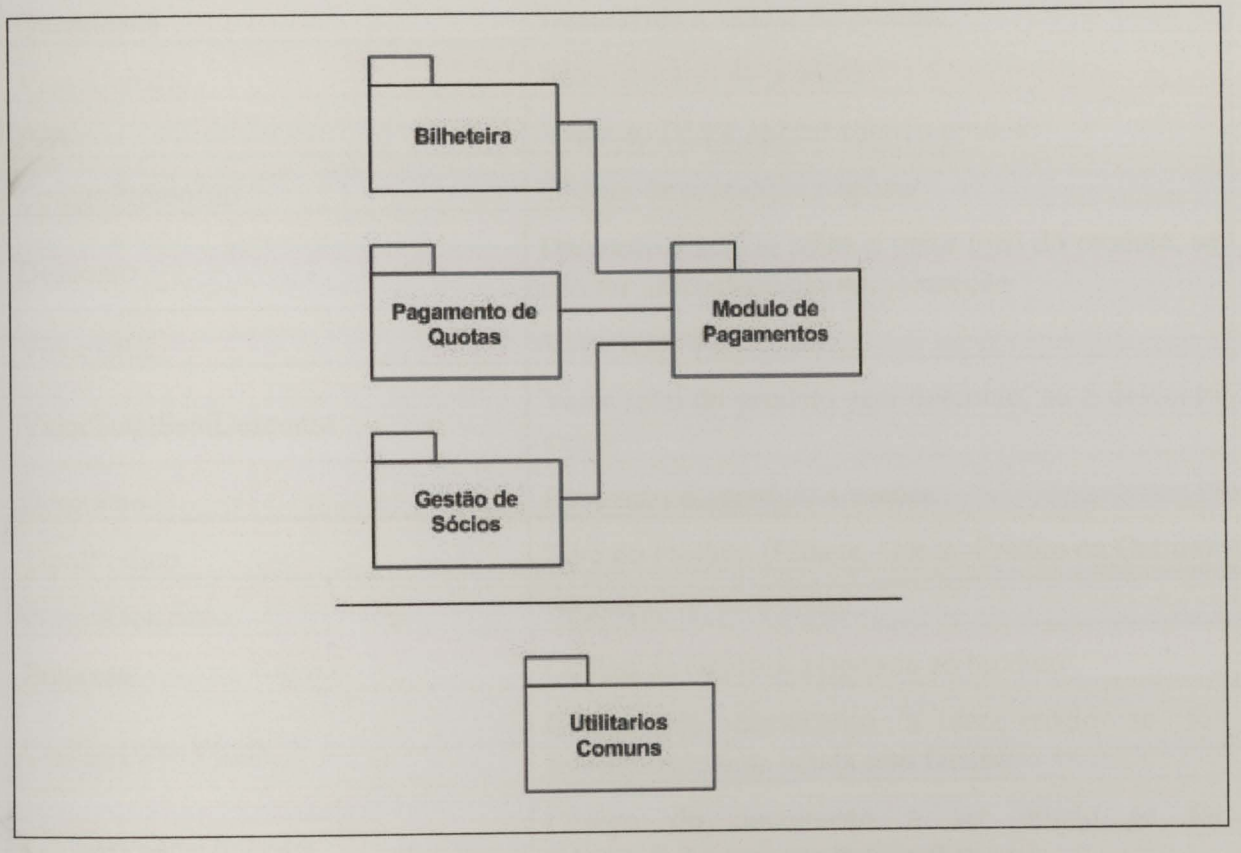


Figura 33 - Módulos e interacções entre eles

Como mostra a figura anterior estas três funcionalidades serão independentes entre si neste momento, apenas interagindo com o módulo de pagamento directamente. Numa outra fase, existiram outras integrações, como por exemplo o pagamento de quotas em atraso de um sócio no momento da compra do bilhete. Os utilitários comuns desenvolvidos serão reutilizados em todos estes módulos.

3.4 Desenvolvimento do módulo de pagamentos

O primeiro módulo a ser criado foi um módulo de pagamentos, módulo este que seria reutilizado em qualquer outro local onde fosse necessário efectuar qualquer tipo de pagamento.

Sendo assim, a primeira coisa a definir foi um objecto Produto que permitisse em qualquer altura ser instanciado e passado a este módulo de forma a ser possível efectuar o pagamento. De seguida apresento este objecto descrevendo cada uma das suas propriedades.

PROPRIEDADE	DESCRIÇÃO
ID	Identifica cada produto unicamente
CodigoProduto	Código numérico do produto na base de dados
Quantidade	Quantidade a vender do produto
ValorUnitario	Valor unitário do produto
IVA	Valor do IVA a aplicar sobre o produto
CodigoPromocao	Código da promoção a aplicar
Desconto	Desconto a aplicar sobre o valor total do produto, se não for utilizado nenhuma promoção
ValorTotal	Valor total do produto
ValorTotalSemDesconto	Valor total do produto sem desconto, se o desconto existir
Descricao	Descrição do produto a vender
TipoProduto	Tipo do Produto (Bilhete, Quota, Crédito ou Outros)
PromoDescricao	Descrição da Promoção
Empresa	Código da empresa associada ao produto
CodigoTalaoVenda	Código do documento a ser criado se for seleccionado uma venda sem factura
CodigoVendaDinheiro	Código do documento a ser criado se for seleccionado uma venda com factura
CodigoDocumento	Código do documento criado para efectuar a venda deste produto (talão de venda ou venda a dinheiro)

Tabela 8 - Propriedades do objecto Produto

De todas estas propriedades, apenas as 7 primeiras são passíveis de ser preenchidas pelo módulo que está a chamar este módulo, sendo que apenas as cinco primeiras são obrigatórias. Todas as outras, se instanciadas no momento da chamada, serão preenchidas com valores retirados da base de dados.

Dentro deste objecto, existe ainda uma estrutura que permite guardar a informação de uma promoção.

PROPRIEDADE	DESCRIÇÃO
Codigo	Código da promoção
Designacao	Designação da promoção
Qtdminima	Quantidade mínima para usufruir da promoção
Desconto	Desconto a efectuar sobre a quantidade mínima de produtos

PROPRIEDADE	DESCRIÇÃO
Qtdadicional	Quantidade adicional de produtos da promoção
Descontoadicional	Desconto a efectuar sobre a quantidade adicional da promoção

Tabela 9 - Propriedades da estrutura Promocao

Foram definidos outros objectos, mas estes apenas para estruturarem e facilitarem a organização dos dados durante a execução do módulo e na altura de inserir os dados na base de dados. Nas quatro tabelas seguintes está descrito cada um destes objectos.

O objecto do tipo Pagamento é utilizado para representar os tipos de pagamento disponíveis e guardar a informação relativa aos tipos de pagamentos utilizados numa determinada venda.

PROPRIEDADE	DESCRIÇÃO
ID	Identificador do pagamento
Designacao	Designação do tipo de pagamento
Defeito	Se este é o pagamento por defeito (para permitir troco é necessário que exista pelo menos um tipo de pagamento por defeito)
Divisa	Divisa em que está a ser utilizado
ValorCambio	Valor do cambio (para Euros)
Imagem	Nome da imagem a apresentar
Ordem	Ordem em que aparece o tipo de pagamento na lista de meios de pagamento
TipoPagamento	Se é de entrada ou saída (troco)
Valor	Valor do meio de pagamento
ValorUtilizado	Valor utilizado em cada momento deste meio de pagamento. É utilizado quando se distribui os pagamentos pelos documentos criados
ValorMinimo	Valor mínimo para se poder utilizar este meio de pagamento
Quotas	Se é possível pagar quotas com este meio de pagamento
NeedsIdentificacao	Se necessita de um documento de identificação para ser aceite
Identificacao	Documento de identificação
NeedsValidade	Se necessita da data de validade do documento de identificação
DataValidade	Data de validade do documento de identificação

Propriedade	Descrição
NumeroCheque	Número do cheque
NeedsTelefone	Se necessita de um numero de telefone de contacto
Telefone	Número de telefone de contacto
NeedsBanco	Se necessita do nome do banco
Banco	Nome do banco
Local	Local onde está a ser efectuado o pagamento (secretaria ou bilheteira p. ex.)

Tabela 10 - Propriedades do objecto Pagamento

O objecto do tipo Documento representa os documentos que são criados ao efectuar um pagamento.

Propriedade	Descrição
Código	Código do documento
Entidade	Entidade associada a este documento
ValorTotal	Somatório do valor de cada linha do documento.
ValorPago	Valor do documento que já está pago
ValorIVA	Valor do IVA
Incidencia	Valor da incidência do IVA
Tipo	Tipo de Documento (Talão de Venda, Venda a Dinheiro, Pagamento Automático, Crédito, Recibo de Cobrador, Devolução)
TipoID	Código do tipo de documento a criar
Print	Se é para ser impresso ou não
Operador	Operador que criou o documento
Caixa	Caixa onde foi criado o documento
DiarioCaixa	Diário de caixa aberto na altura da criação do documento
CodigoEmpresa	Código da empresa associada ao documento
CodigoLocal	Local da empresa onde foi criado o documento
Linhas	Lista com as linhas do documento
Pagamentos	Lista com os pagamentos utilizados para este documento.

Tabela 11 - Propriedades do objecto Documento

Como foi possível verificar na tabela anterior, existe uma lista de objectos do tipo linha de documento, descrito de seguida.

PROPRIEDADE	DESCRIÇÃO
IDLinha	Identificador da linha do documento
CodigoProduto	Código do produto
Valor	Valor da linha
ValorSemDesconto	Valor da linha sem desconto
Quantidade	Quantidade do produto
IVA	Valor do IVA
CodigoPromocao	Código do promoção
Desconto	Desconto aplicado à linha

Tabela 12 - Propriedades do objecto LinhaDocumento

Ainda associado ao documento existe o objecto CabecalhoDocumento que permite agregar a informação necessária para emitir uma factura.

PROPRIEDADE	DESCRIÇÃO
Nome	Nome da entidade
Contribuinte	Número de contribuinte
Morada1	Primeira linha da morada
Morada2	Segunda linha da morada
CodigoPostal	Código postal
CodigoPostalDesignacao	Designação do código postal
IDLocal	Local da empresa

Tabela 13 - Propriedades do objecto CabecalhoDocumento

No Anexo C pode-se encontrar um diagrama destas classes.

Interessa agora descrever a forma de utilização e funcionamento do módulo criado, que será realizada através do exemplo do pagamento de quotas.

No momento do pagamento de quotas, este módulo cria uma lista de objectos do tipo produto, com os dados necessários. Cada aplicação terá a sua forma de apresentar os dados ao utilizador, sendo isto completamente indiferente para o módulo de pagamentos desde que sejam preenchidos as propriedades obrigatórias (código produto, quantidade, valor unitário e IVA).

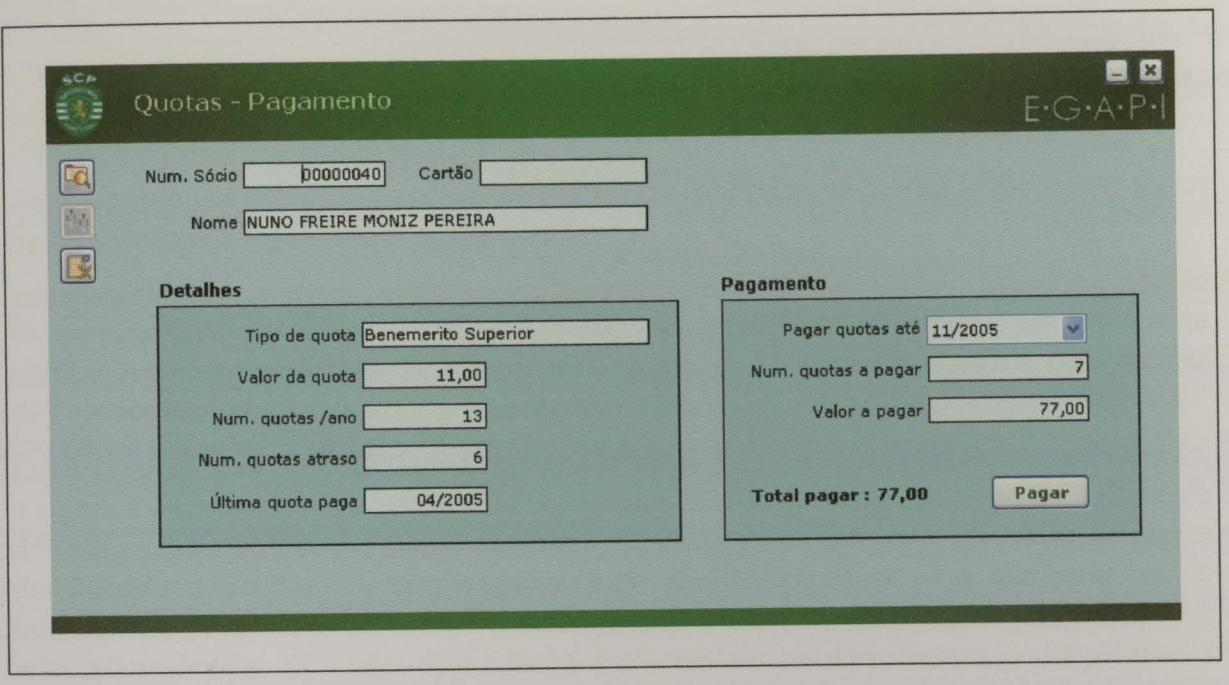


Figura 34 - Formulário de pagamento de quotas antes de chamar o módulo de pagamento

Com este objecto é instanciada a classe *USC_Start*, disponibilizada pela *dll* criada por este módulo (é necessário instanciar um *User Control* que abre um formulário com o módulo de pagamento uma vez que a partir de uma *dll* não é possível instanciar um formulário directamente), com os parâmetros descritos de seguida.

```
public USC_Start(
            string entidade,
            List<GatewayPagamentoWS.Produto> produtos,
            string caixa,
            string diariocaixa,
            string local,
            string operador,
            bool usaTrocoAbatimento,
            bool usaTrocoCredito,
            double valorMaximo)
```

Permite inicializar o módulo de pagamentos.

- entidade: permite associar desde logo o pagamento a uma entidade, apesar de não ser obrigatório;
- produtos: lista de objectos do tipo Produto que identificam os produtos a pagar;
- caixa: caixa onde está a ser realizado o pagamento;
- diariocaixa: diário de caixa aberto no momento do pagamento;
- local: local da empresa onde está a ser efectuado o pagamento;
- operador: operador que está a realizar o pagamento;
- usaTrocoAbatimento: permite definir um valor máximo para o pagamento e utilizar a diferença entre este valor e o valor total da venda para abatimento de outras despesas que a entidade poderá ter;
- usaTrocoCredito: permite utilizar o troco para a criação de crédito a utilizar pela entidade noutra situação;
- valorMaximo: valor máximo se for possível usar o troco para abatimento;

Com estas informações é possível efectuar um pagamento em qualquer situação, seja pagamentos de quotas, venda de bilhetes, inscrição de um novo sócio, entre outras.

Ao arrancar o módulo de pagamento são efectuadas validações aos dados e carregados outros dados necessários para efectuar o pagamento.

Inicialmente são carregadas algumas configurações genéricas da empresa. São elas o código de produto para a criação de créditos (se não estiver definido não poderá ser um crédito utilizando o troco) e a taxa de IVA por defeito (apesar de ser preferível passar o valor da taxa de IVA aos instanciar o módulo, se esta não for definida será utilizada esta taxa por defeito).

Logo de seguida são carregadas informações adicionais ao produtos a pagar, a sua descrição, a empresa associada a eles, o código do documento de talão de venda, o código do documento de venda a dinheiro e as definições da promoção, de forma a preencher todas as propriedades do objecto Produto. No caso de um produto estar abrangido por uma promoção poderá haver uma necessidade de criar um nova linha de produto, como exemplifico de seguida. Se existir uma promoção que ao vender dois bilhetes é oferecido o terceiro, em vez de haver apenas uma linha do produto com quantidade três passará a existir duas, a primeira com quantidade dois e desconto de 0% e uma outra com quantidade 1 e desconto de 100%.

De seguida são carregados os tipo de pagamento disponíveis para os produtos a vender. Existem tipos de pagamento que não são aceites para pagamentos de quotas que por isso não interessa que apareçam nos pagamentos disponíveis. Se apenas se estiver a pagar quotas, também se poderá utilizar o crédito disponível para efectuar o pagamento.

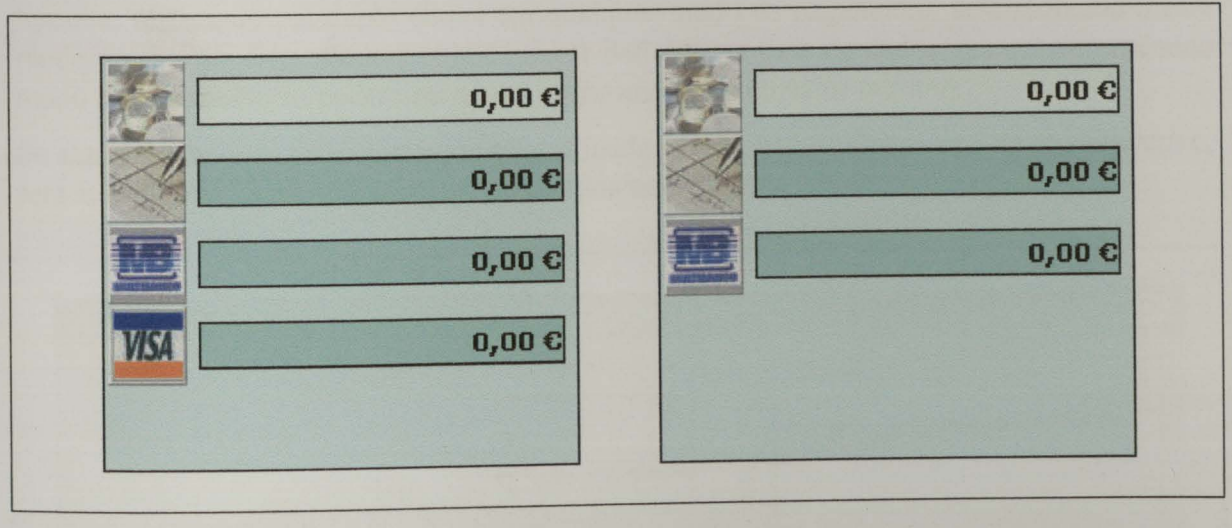


Figura 35 - Métodos de pagamento disponíveis para venda de bilhetes (à esquerda) e para pagamento de quotas (à direita)

Após o carregamento de todos estes dados é apresentado ao utilizador o formulário para pagamento. Aqui, ele terá toda a informação relativamente aos produtos que está a pagar, bem como os modos de pagamento disponíveis.

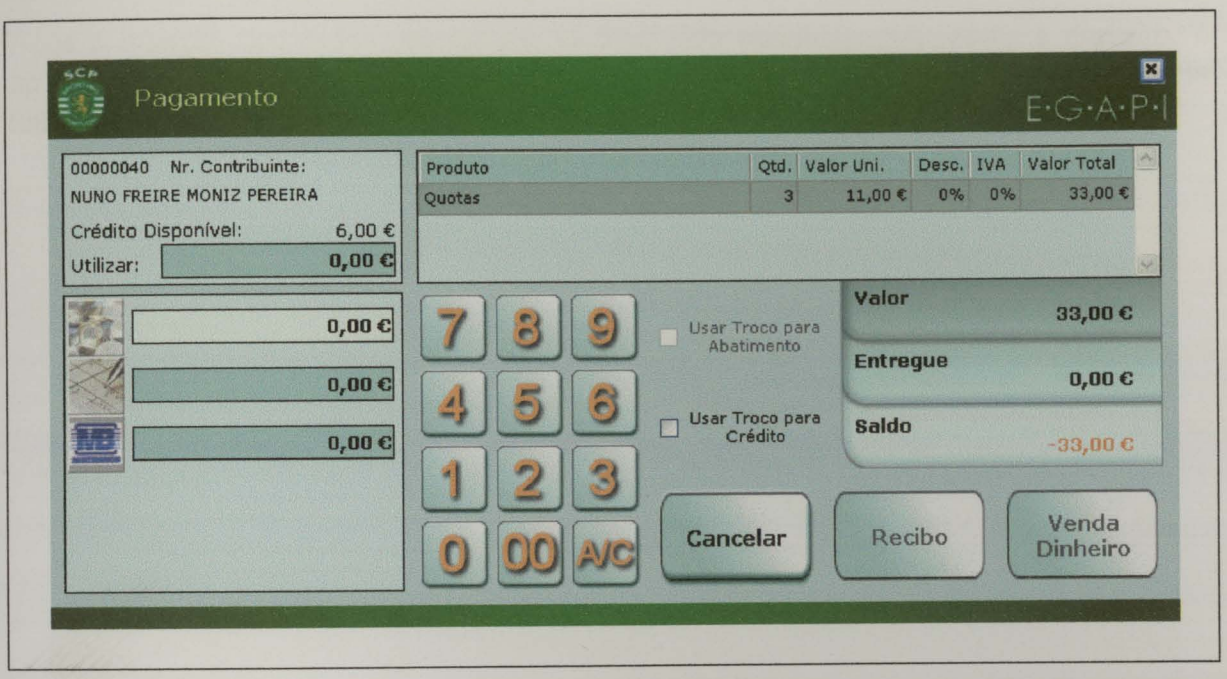


Figura 36 - Módulo de pagamento

Neste momento o utilizador pode utilizar tanto o teclado como os botões disponíveis no formulário para inserir as quantias desejadas em cada modo de pagamento. Como foi referido antes, apenas no modo de pagamento por defeito o valor a pagar poderá ser superior ao valor total, uma vez que é apenas com esse modo que se poderá dar troco ao cliente. Todos os outros modos estão limitados em cada momento pelo valor que falta pagar. Em qualquer instante, realizando um duplo clique em qualquer modo de pagamento, será atribuído a esse modo o valor em falta. Poderá também estar definido na base de dados que um determinado modo de pagamento só poderá ser aceite se for utilizado um valor mínimo.

Se algum dos modos de pagamentos seleccionados necessitar de algum tipo de comprovativo, será apresentada um formulário com os campos necessários.

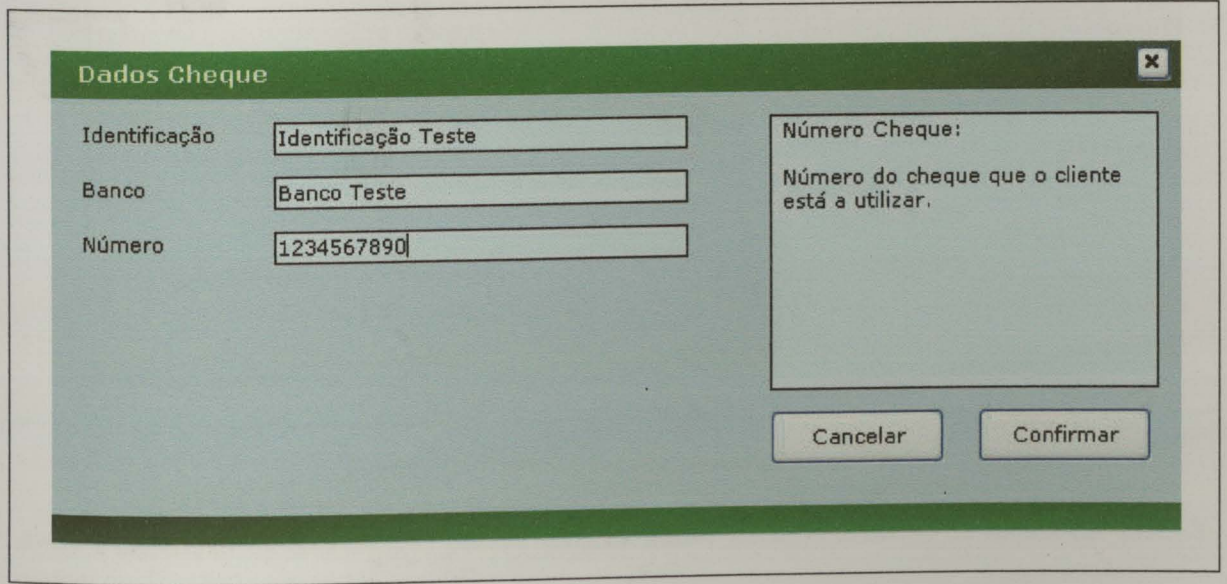


Figura 37 - Formulário para o preenchimento de comprovativos

Após a inserção dos dados necessários, se tiver sido escolhido uma venda a dinheiro, é apresentado o seguinte formulário, que permite inserir os dados necessários para a emissão de uma factura.

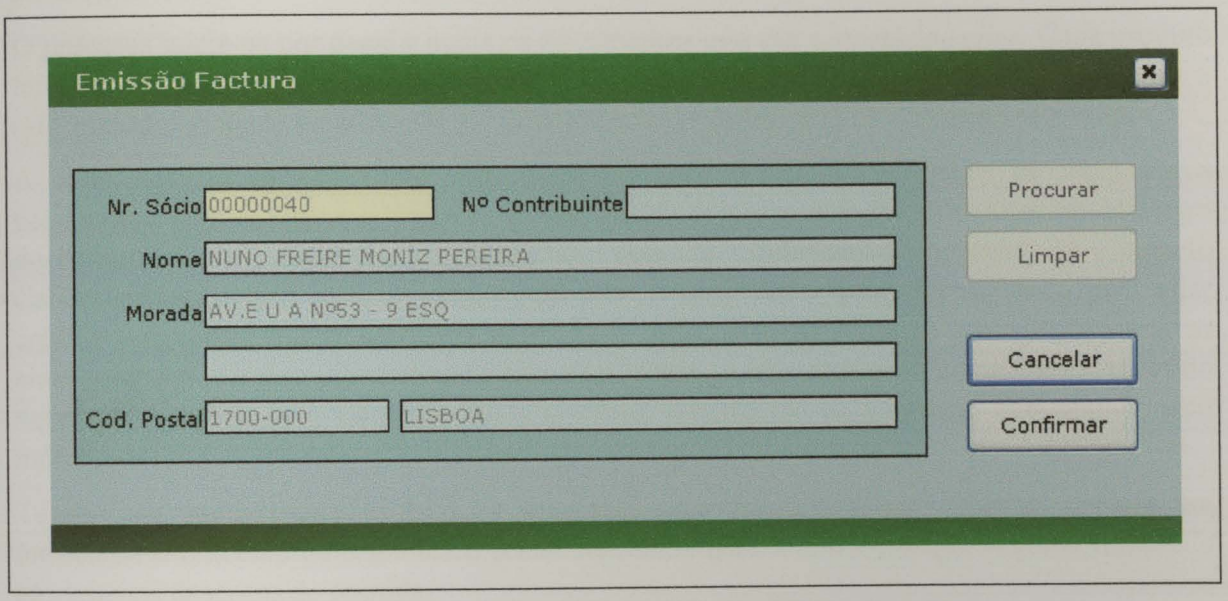


Figura 38 - Formulário para emissão de factura

No caso do pagamento de quotas não é possível modificar os dados da factura, uma vez que esta terá que ser emitida em nome do sócio a quem se está a pagar as quotas. Podendo alterar estes dados, seleccionando o botão Procurar será apresentado o seguinte formulário que permite permitem efectuar uma pesquisa sobre as entidades já existentes no sistema.

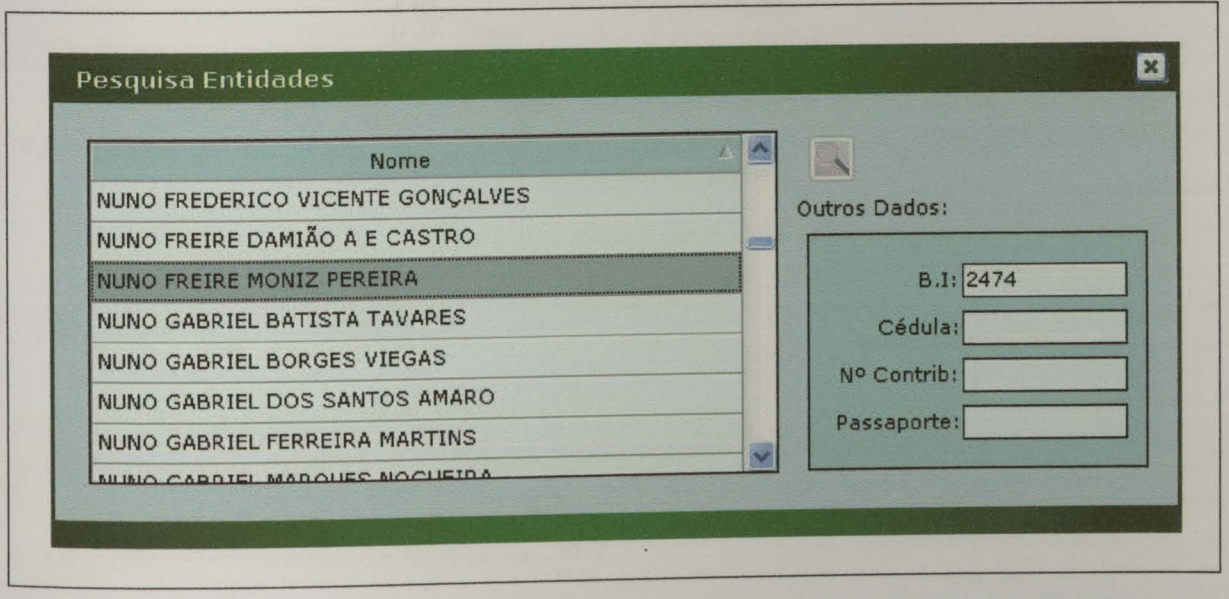


Figura 39 - Formulário para pesquisa de entidades

Após a execução de todos estes passos está na altura de inserir todos os dados na base de dados.

Ao iniciar o procedimento `efectuaPagamento()` são instanciados outros objectos, referidos no início deste ponto, nomeadamente o CabecalhoDocumento, que apenas é criado se for

seleccionado uma venda a dinheiro e um novo Pagamento do tipo saída se existir troco. A partir deste momento são efectuados diversos acessos à base de dados, todos dentro de uma mesma transacção para garantir que se ocorrer algum erro em qualquer um dos passos seja possível anular todas as inserções ou alterações feitas na base de dados.

O processo inicia-se por decidir quais os documentos que são necessários criar. Cada produto tem uma empresa associada que poderá ter documentos diferentes de outra, logo com a informação carregada no arranque do módulo pode-se definir que os documentos a criar.

A cada documento são então associados os seus produtos, repartido os pagamentos necessários para efectuar o pagamento (se existir troco este é inserido logo no primeiro documento) e inserido um cabeçalho com a informação contida no objecto CabecalhoDocumento se for a criação de uma factura. Se o troco for utilizado para criar crédito é também criado um documento de crédito, com as mesmas informações que os anteriores, ou se o troco for utilizado para abatimento, o primeiro produto será encarecido do valor do troco, sendo o valor da dívida em falta actualizado na aplicação que chamou este módulo.

Tendo todas estas inserções na base de dados sido concluídas com sucesso, é dado por terminado o processo de pagamento sendo retornado para a aplicação que chamou o módulo as variáveis descritas de seguida.

VARIÁVEL	DESCRIÇÃO
VAR_PagamentoEfectuado	Se o pagamento foi efectuado com sucesso ou não
VAR_ValorEntregue	Valor entregue no total
VAR_Troco	Valor do troco
VAR_Resto	Valor utilizado para abatimento
VAR_Erro	Se tiver ocorrido algum erro durante a execução do pagamento será retornada uma string com uma descrição sumária do erro
VAR_Produtos	É retornado à aplicação a mesma lista de produtos que foi passada inicialmente, com a propriedade CodigoDocumento preenchida, que representa o código do documento que foi criado para esse produto

Tabela 14 - Variáveis retornadas pelo modulo de pagamento

3.5 Desenvolvimento da *Bilheteira .NET*

Este protótipo foi desenvolvido, como foi referido no início, com o intuito de tornar esta funcionalidade mais intuitiva e visual. Sendo assim, foi elaborado uma solução que concretizasse este objectivo de um modo o mais eficaz possível.

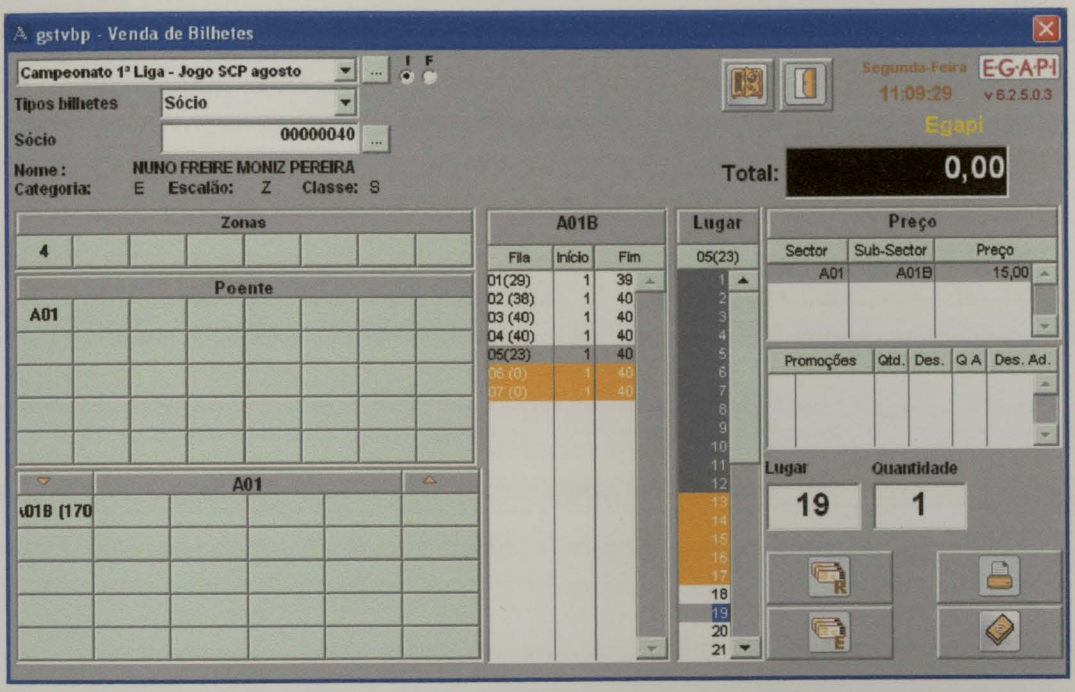


Figura 40 - Formulário antigo de venda de bilhetes (em cima) e o protótipo desenvolvido (em baixo)

Foi criado então um protótipo que permitisse a selecção dos lugares para venda, bem como dos subsectores do estádio, através de uma imagem que nos permite ter uma visão global da

área onde estes locais se encontram. A imagem utilizada é um *User Control* criado para esta aplicação, mas que pode ser reutilizado em qualquer outra.

O *User Control ImageMap* é uma extensão do objecto *PictureBox*, já existente na biblioteca .NET. Mantendo todas as propriedades do objecto *PictureBox* foram então adicionadas algumas funcionalidades que permitem criar um mapa sobre a imagem, podendo atribuir a cada região uma acção específica.

Este *User Control* permite guardar uma estrutura associado a cada região com a *tooltip* e uma qualquer informação genérica que permita a identificação correcta da região e, consequentemente, realizar a função mais adequada. De seguida descrevo os métodos criados para a utilização correcta deste *User Control*.

int AddElipse(string tooltip, object info, Point Center, int Radius)
Cria uma região com o formato de uma elipse na imagem com o centro Center e raio igual a Radius, com uma tooltip com o texto da variável tooltip e com a informação contida no objecto info. Retorna o índice da região criada.

int AddElipse(string tooltip, object info, int x, int y, int Radius)
Cria uma região com o formato de uma elipse na imagem com o centro com as coordenadas (x, y) e raio igual a Radius, com uma tooltip com o texto da variável tooltip e com a informação contida no objecto info. Retorna o índice da região criada.

int AddRectangle(string tooltip, object info, Rectangle rectangle)
Cria uma região com o formato de um rectângulo na imagem com as definições do objecto rectangle, com uma tooltip com o texto da variável tooltip e com a informação contida no objecto info. Retorna o índice da região criada.

int AddRectangle(string tooltip, object info, int x1, int y1, int x2, int y2)
Cria uma região com o formato de um rectângulo na imagem com as coordenadas (x1, y1) para o canto superior esquerdo e com as (x2, y2) para o canto inferior direito, com uma tooltip com o texto da variável tooltip e com a informação contida no objecto info. Retorna o índice da região criada.

int AddPolygon(string tooltip, object info, Points[] points)
Cria uma região com o formato definido pelos pontos da variável points, com uma tooltip com o texto da variável tooltip e com a informação contida no objecto info. Retorna o índice da região criada.

object GetInfo(int index)
Retorna o objecto guardado como informação de uma área com o índice índex.

void ChangeInfo(int index, string tooltip, object info)
Modifica a informação guardada de uma área com o índice índex.

int FindRegion(object info)
Retorna o índice de uma região com a informação igual a info. Retorna -1 se não for encontrada.

Definido então este *User Control*, peça central do protótipo, passo a descrever todas as funcionalidades implementadas com a ajuda de um exemplo de utilização.

Ao iniciar a aplicação, o programa utiliza a função `getEventos` para ir buscar à base de dados um *DataSet* com todos os eventos disponíveis para venda na instalação seleccionada. Neste momento, estes eventos são apenas referentes à instalação Estádio José de Alvalade uma vez que apenas se dispõe de dados referentes à mesma, mas podendo facilmente ser adaptado para outras mudando apenas uma variável global da aplicação.

Com esta informação é preenchida uma *ComboBox* que nos permite seleccionar o evento para o qual queremos vender bilhetes e assim carregar a informação necessária sobre o evento. É isso que acontece ao premir o botão localizado ao lado da *ComboBox* dos eventos.

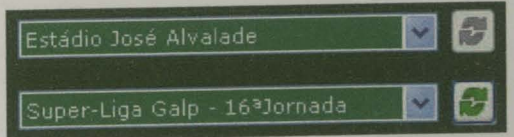


Figura 41 - *ComboBox* para selecção dos eventos disponíveis para venda

Esta acção inicia um processo repartido por quatro etapas:

- Com a função `getDescricaoEvento` é preenchido um *DataSet* com a descrição do evento.
- De seguida é executada a função `getDadosEvento` que também preenche um *DataSet* contendo toda a informação sobre os subsectores disponíveis para venda, bem como uma tabela, sem dados neste momento, com a informação sobre a ocupação da

instalação. Esta tabela vai sendo preenchida quando é necessário desenhar a ocupação de um qualquer subsector.

- Posteriormente é realizada a função `getTipoBilhetes` que, como as duas anteriores, também preenche um *DataSet* com todos os tipos de bilhetes disponíveis para o evento.
- Finalmente, através da função `detectaAreasInvisíveis`, é feita uma verificação de quais os subsectores que não estão definidos na base de dados de acordo com a imagem do estádio utilizada.

Com esta informação é agora possível, de uma forma simples, seleccionar um bilhete para venda, quer para sócio quer para público.

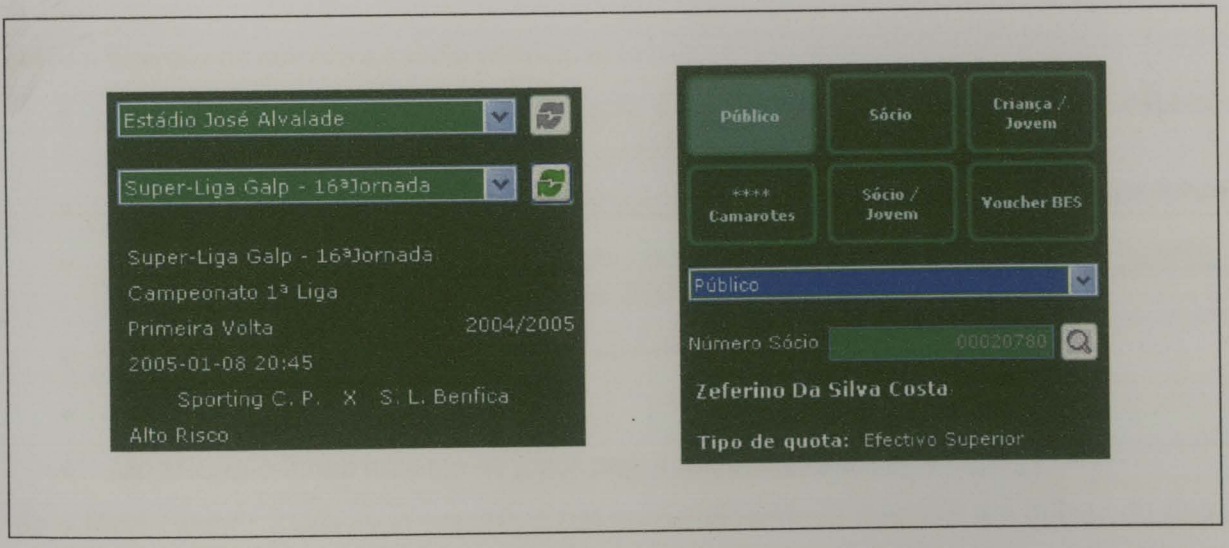


Figura 42 - Menu apresentado à esquerda do formulário apresentando todas as informações relativas ao evento e outro apresentado à direita com os tipo de bilhetes disponíveis

O processo de venda de um bilhete para público é semelhante ao da venda de um bilhete de sócio, apenas não é necessário realizar diversas validações. Sendo assim, de seguida apresento a forma de venda de um bilhete de sócio, referindo que pontos são comuns à venda de um bilhete de público.

Ao seleccionar um bilhete para sócio, se a *TextBox* Nr Sócio apresentada na imagem anterior estiver vazia, surgirá uma janela a pedir o mesmo número de sócio. É impossível em qualquer circunstância vender um bilhete para sócio sem antes este ter sido validado.

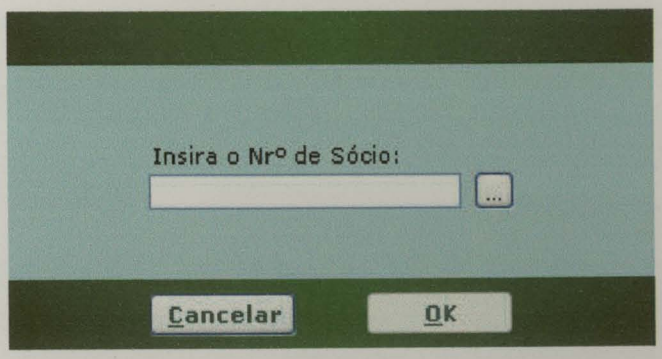


Figura 43 - Formulário para inserção do número de sócio

Após a inserção do número de sócio verifica-se se:

- o número de sócio existe (esta validação é também efectuada ao vender um bilhete para público se no momento da venda o sócio se quiser identificar);
- o sócio está activo;
- ainda não adquiriu qualquer bilhete do tipo sócio para o evento (cada sócio apenas pode adquirir um bilhete deste tipo para cada evento);
- não adquiriu um agrupamento que dê acesso ao evento;
- tem as quotas em dia;
- não atingiu o limite máximo de jogos para a sua categoria.

Se o sócio cumprir todos estes requisitos tem a possibilidade de comprar um bilhete do tipo sócio.

Após estas validações é verificado também se o sócio tem alguma reserva para o jogo ou se tem algum cartão destinado a outros eventos, podendo optar por comprar o bilhete para o mesmo lugar, caso esteja disponível.

Após estas validações, o sócio apenas terá que seleccionar o lugar que deseja adquirir. A partir deste ponto, a venda de bilhetes para público processa-se da mesma forma, surgindo na imagem os subsectores disponíveis para venda do tipo de bilhete seleccionado.

Após a escolha do subsector, é criada uma imagem com a ocupação do mesmo. Como foi referido anteriormente, se este subsector nunca tiver sido seleccionado, será carregada a tabela com a sua ocupação, efectuando a partir deste momento apenas actualizações aos dados carregados.

A imagem criada é totalmente customizável, tanto as cores como os tamanhos utilizados, tirando partido do *User Control ImageMap* para representar de uma forma simples e prática a disposição dos lugares no subsector e o seu estado em cada momento.

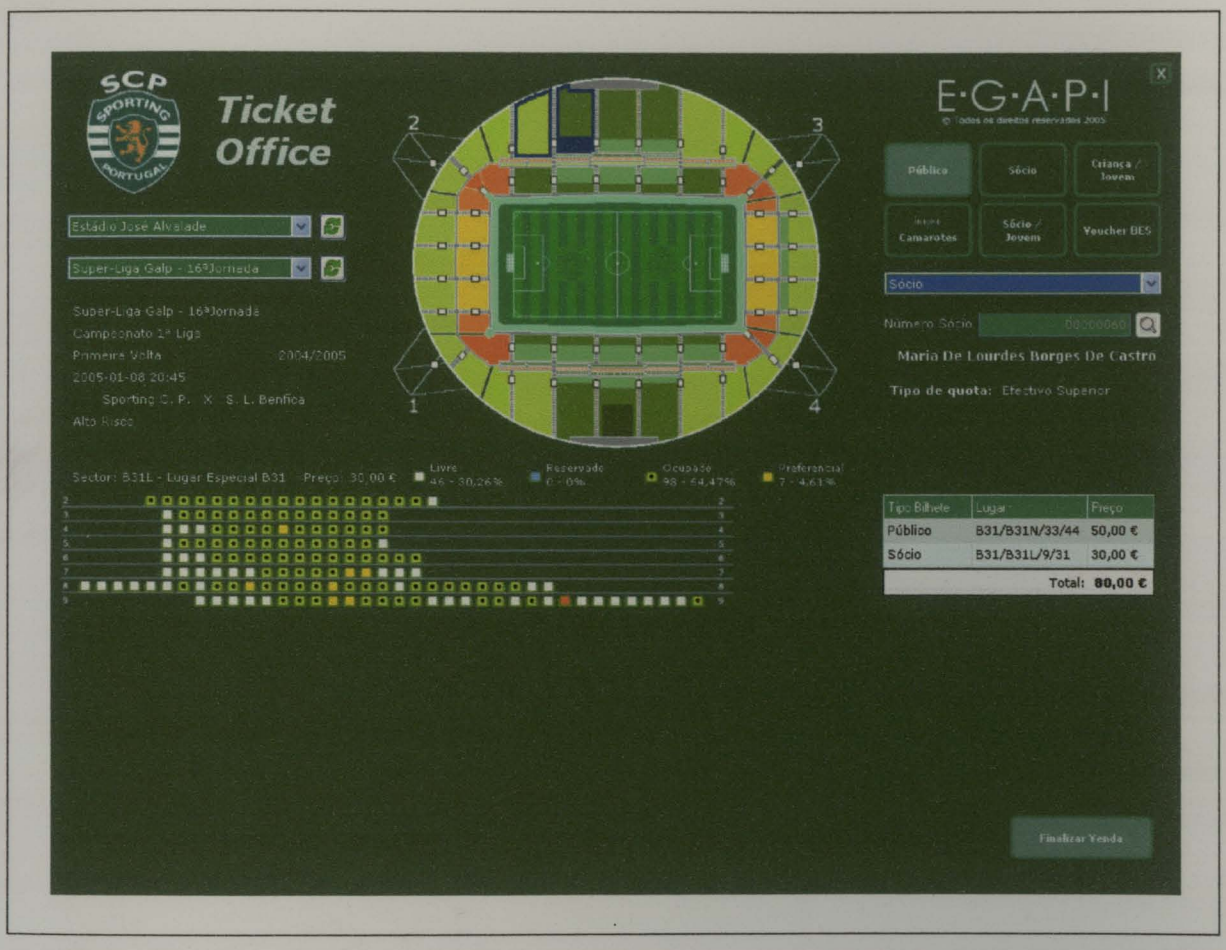


Figura 44 - Protótipo Bilheteira .NET com os subsectores disponíveis para venda delineados em cima e os lugares disponíveis em baixo

Cala lugar pode estar em um de quatro estados: Livre, Ocupado, Reservado ou Preferencial. Os lugares reservados apenas podem ser seleccionados no momento da verificação de reservas, e os preferenciais estão também reservados, mas para sócios que tenham adquirido um agrupamento que tenha acesso ao evento. Sendo assim, apenas estão disponíveis neste momento os lugares livres.

Um sócio apenas pode seleccionar um bilhete de tipo sócio, mas pode sempre seleccionar os bilhetes que quiser do tipo público. Por cada bilhete seleccionado é criada uma linha numa tabela no canto inferior direito com a informação relativa ao mesmo (sector, subsector e preço, estas três são visíveis, e IVA, código produto, lugar, fila, tipo de bilhete e índice relativo ao *ImageMap*, que não são apresentadas mas necessárias para o correcto funcionamento do protótipo).

Estando todos os bilhetes seleccionados pode-se passar ao pagamento dos mesmos. Para isso é criado um objecto do tipo Produto (referido no módulo de pagamentos), agrupando os bilhetes por código de produto, preço e IVA.

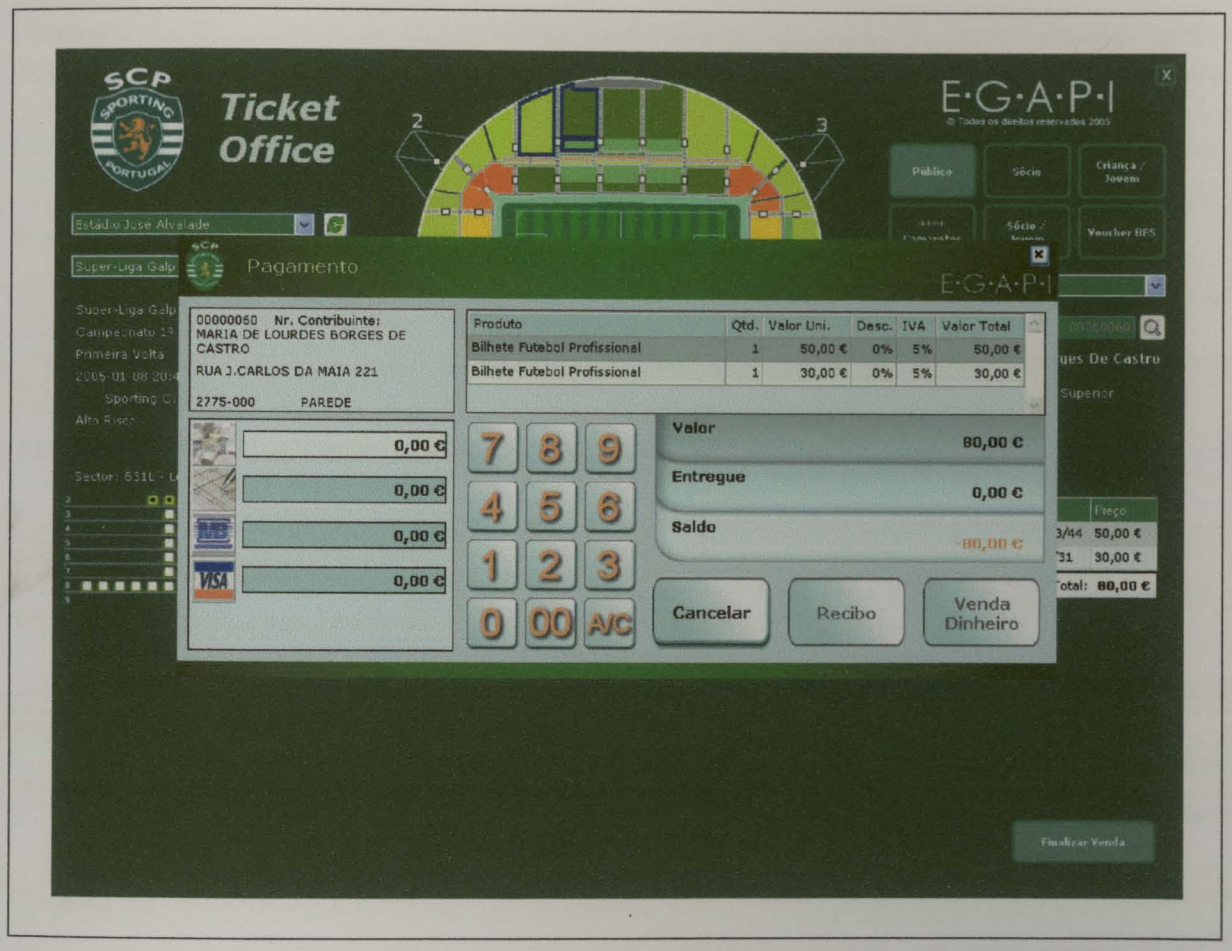


Figura 45 - Módulo de pagamentos chamado através da bilheteira

O funcionamento deste módulo é igual ao descrito na secção 3.4.

Estando o pagamento efectuado, são emitidos os bilhetes vendidos. Para isso é gerado um código de barras único, que permitirá o acesso à instalação por parte do comprador. Em simultâneo são actualizadas as bases de dados, central e o *DataSet* local, com a nova informação. A partir deste momento mais ninguém terá acesso ao mesmo lugar.

No Anexo D pode-se encontrar uma parte do esquema da base de dados GSTSOC6 (em funcionamento em clientes) apenas com as principais tabelas utilizadas no desenvolvimento deste protótipo, uma vez que esta base de dados tem na sua totalidade cerca de 200 tabelas.

3.6 Desenvolvimento de utilitários comuns

Com o intuito de automatizar algumas funções bastante comuns e evitar a repetição de código foi criada uma *dll* com algumas funcionalidades genéricas. Neste momento estão disponíveis funcionalidades de acesso à base de dados e de registo de erros, podendo mais tarde vir a ser completada com outras mais.

3.6.1 Acesso à base dados

O acesso à base de dados é uma das tarefas mais comuns numa aplicação empresarial, ou qualquer outra que lide com dados, principalmente em grande escala. Desta forma criou-se uma classe que permite simplificar este processo e ao mesmo tempo garantir que não fica

nenhuma ligação à base de dados aberta sem motivo. Os métodos implementados estão descritos de seguida.

OleDbTransaction getTransacao(string connString)
Retorna uma transação referente a uma ligação à base de dados com a ligação definida por connString.

bool commitTransacao(OleDbTransaction trans)
Faz o commit de uma transação aberta. Retorna verdadeiro se tudo correr bem.

bool rollbackTransacao(OleDbTransaction trans)
Faz o roolback de uma transação aberta. Retorna verdadeiro se tudo correr bem.

DataSet getDados(OleDbTransaction trans, string SQL)
Retorna um DataSet com os dados referentes à query SQL efectuada na transação trans.

DataSet getDados(string connString, string SQL)
Retorna um DataSet com os dados referentes à query SQL efectuada na ligação definida por connString.

DataSet getDados(OleDbTransaction trans, string[] SQLs, string[] TableNames)
Retorna um DataSet, com diversas tabelas com os nomes definidos em TableNames, com os dados referentes às querys SQL efectuadas na transacção trans.

DataSet getDados(string connString, string[] SQLs, string[] TableNames)
Retorna um DataSet, com diversas tabelas com os nomes definidos em TableNames, com os dados referentes às querys SQL efectuadas na ligação definida por connString.

DataSet getDados(OleDbTransaction trans, OleDbCommand comm)
Retorna um DataSet com os dados referentes ao comando OleDB comm executado na transacção trans.

DataSet getDados(string connString, OleDbCommand comm)
Retorna um DataSet com os dados referentes ao comando OleDB comm executado na ligação definida por connString.

OleDbCommand executaProcedimento(OleDbTransaction trans, OleDbCommand comm)
Retorna o comando OleDb comm com os parâmetros preenchidos de acordo com a execução do mesmo na transacção trans.

OleDbCommand executaProcedimento(string connString, OleDbCommand comm)
Retorna o comando OleDb comm com os parâmetros preenchidos de acordo com a execução do mesmo na ligação definida por connString.

bool executaQuery(OleDbTransaction trans, string SQL)
Executa uma query SQL (Update, Insert ou Delete) na transacção trans e retorna verdadeiro se tudo correr bem.

bool executaQuery(string connString, string SQL)
Executa uma query SQL na ligação definida por connString e retorna verdadeiro se tudo correr bem.

Todos estes métodos são baseados em *OleDb* para permitir uma maior facilidade de acesso a base de dados de diferentes fabricantes. Até este ponto foi testada com sucesso com bases de dados Oracle e SQL Server.

3.6.2 Registo de erros

O registo de erros no projecto Wine-Cellar IS .NET foi dos pontos a que menos atenção foi dada, o que tornou algumas tarefas de *debugging* algo complicadas. Desta forma optou-se por criar uma classe que permita de uma forma simples registar em *logs* estes erros.

De seguida apresento os métodos disponíveis por esta classe.

bool RegistaErro(Exception Excepcao)
Regista a excepção Excepcao nos logs. Retorna verdadeiro se tudo correr bem.

bool RegistaMensagem(string Mensagem)
Regista a mensagem Mensagem nos logs. Retorna verdadeiro se tudo correr bem.

bool RegistaErroMensagem(Exception Excepcao, string Mensagem)
Regista a excepcção Excepcao juntamente com a mensagem Mensagem nos logs. Retorna verdadeiro se tudo correr bem.

Esta classe é configurável através de um ficheiro XML com as propriedades descritas na tabela seguinte.

PROPRIEDADE	DESCRIÇÃO
LogEnable	Se é para realizar o log.
LogEventViewer	Estando a verdadeira, esta propriedade redirecciona o registo de erros e mensagens para o visualizador de eventos do Windows. A falso escreve num ficheiro de texto.
PathLog	Directório onde serão colocados os logs em ficheiro de texto.

Tabela 15 - Propriedades configuráveis da classe ErrorLog

De notar que também é criada automaticamente uma pasta para cada aplicação que execute estes métodos e são criados ficheiros de *log* diários, de forma a garantir que a informação não fique misturada e em ficheiros demasiado extensos.

3.7 Estado Actual

No seu estado actual, qualquer um dos módulos/projectos desenvolvidos permitem uma percepção real das suas potencialidades. Partindo destes protótipos será simples criar uma aplicação simples e inovadora.

Todos estes projectos já foram realizados tendo em conta os pontos negativos do projecto Wine-Cellar IS, sendo que a estrutura de toda a solução foi criada com a arquitectura referida no ponto 2.8, permitindo assim uma melhoria na organização do código desenvolvido e uma integração com outros projectos mais simples.

O tratamento de erros também foi objecto de uma melhoria significativa. Com a criação dos utilitários comuns, obteve-se uma base que permite um registo e controlo, e subsequente tratamento de erros muito mais eficaz.

Em termos de segurança não foi feito qualquer tipo de desenvolvimento. Uma vez que se tratava de protótipos, não houve a preocupação de criar mecanismos que nos permitissem a encriptação dos dados ou de verificar as permissões de um utilizador.

Mas, de um modo geral, como no projecto anterior, o projecto foi bem sucedido uma vez que todos os objectivos propostos inicialmente foram atingidos de uma forma satisfatória.

3.8 Perspectivas de trabalho Futuro

Em termos de trabalho de futuro, a criação de mecanismos de segurança e validação de utilizadores será um passo importante, uma vez que este ponto é e será sempre crítico em todas as aplicações empresariais, de forma a garantir uma aplicação fiável e em que os utilizadores possam confiar.

A criação de novas funcionalidades e a integração de todas elas, de forma a ser criado um único protótipo que seja capaz de provar todo o potencial destes projectos a novos e actuais

clientes, será sem dúvida um ponto chave para a continuação deste projecto além da sua fase de prototipagem.

4 Conclusões

Este estágio foi muito importante do ponto de vista da valorização profissional, uma vez que pude trabalhar com uma equipa jovem, profissional e dinâmica. Tomei conhecimento da metodologia de trabalho de uma empresa de vulto no mercado nacional.

No decorrer do meu estágio pude por em prática conhecimentos adquiridos em variadas disciplinas do curso, como os conceitos de base de dados (das disciplinas de SIBD, TBD e AD), de engenharia e arquitectura de software (disciplinas de ES e ASS), de criação de interfaces (disciplina de IPC), bem como a organização de um trabalho em equipa (disciplina de LGP bem como de todos os trabalhos realizados em grupo).

Graças a este estágio trabalhei pela primeira vez num projecto real. A diferença de todos os projectos realizados no seio da universidade durante o curso e deste prende-se com o facto deste ter finalidades comerciais (a finalidade dos projectos do curso foi sempre a nota final), é limitado por custos financeiros (custos das licenças limitam as possibilidades de escolha no uso de software) e mais, neste projecto tive a possibilidade de lidar com verdadeiros clientes (nos projectos do curso tínhamos o professor como cliente fictício).

Por fim, este trabalho permitiu confirmar que o meu percurso académico foi de extrema importância e os objectivos da LEIC foram cumpridos, isto é, formaram mais um licenciado com um conhecimento sólido e extenso nos domínios da arquitectura de sistemas informáticos e das tecnologias de informação e comunicação.

Bibliografia

[01] *C#.NET Web Developer's Guide*
Saurabh Nandu, Adrian Turtschi, Jason Werry, Greg Hack, Joseph Ahbahari
Syngress, 2002

[02] *C# Complete*
Matt Tagliaferri
Sybex, 2003

[03] *Sams Teach Yourself ASP.NET in 21 Days (2nd edition)*
Chris Payne
Sams Publishing, 2003

[04] *C# Curso Completo*
Paulo Marques, Hernâni Pedroso
FCA - Editora de Informática, 2002

[05] *Programação com ASP.NET (Volume I)*
João Vieira
FCA - Editora de Informática, 2002

[06] *Professional ASP.NET*
Richard Anderson
Whox Press, 2001

[07] *Developing Microsoft ASP.NET server controls and components*
Nikhil Kothari
Microsoft Press, 2003

[08] *ORACLE 8i Curso Completo (2ª Edição)*
Luís M. Campos
FCA - Editora de Informática, 2001

[09] *Oracle Documentation Library*
Release 8.1.7

Referências Web

[01] www.google.com

Página de um motor de busca que se torna um bom ponto de partida para encontrar a mais variada informação que necessitarmos.

[02] http://msdn.microsoft.com/

Página da Microsoft dedicada ao suporte dos seus produtos, disponibilizando muitas e preciosas informações sobre a plataforma .NET.

[03] www.asp.net

Este pode ser considerado o *website* "oficial" das ASP.NET, é suportado e foi criado pela própria Microsoft, tem um fórum bastante completo, onde a equipa que gere e criou a *Framework* .NET, por vezes aparece para dar algumas respostas.

[04] www.codeproject.com

Página onde pude encontrar informação muito útil relativamente à ASP.NET.

[05] http://aspnet.pt4free.net

Este é o primeiro *website* dedicado às ASP.NET em português. Pude encontrar aqui vários tipos de recursos, *links*, ou participar no fórum sobre as ASP.NET, tudo em português. Sem sombra de dúvida o melhor recurso em português sobre as ASP.NET.

[06] www.411asp.net

Este *website* funciona como um directório onde pude encontrar os mais variados apontadores para todo o tipo de necessidades das ASP.NET.

[09] www.acejs.com

Página com muitas scripts Javascript bastante úteis.

[10] www.programmersheaven.com

Página com bastantes informações respeitantes à Javascript e à ASP.NET.

[11] www.unix.org.ua/orelly/oracle/index.htm

Livros online sobre tecnologias Oracle, para uso pessoal, com permissão de O'Reilly & Associates, Inc.

ANEXO A: Historial do Estágio

SEMANA	INÍCIO	FIM
1	1 de Março	4 de Março

TRABALHO REALIZADO

1. Instalação do Microsoft Visual Studio 2005 beta (whidbey) e Oracle Runtime Libraries.
2. Primeiro contacto com as aplicações da empresa, principalmente GESCOM 6.2 e WIS – GESCOM, aplicação que irá ser migrada de Oracle Forms para ASP .NET.
3. Criação de uma página ASP com um datagrid, elemento essencial das páginas para a apresentação dos dados da BD em forma tabular, de forma a testar as suas possibilidades e verificar a sua utilidade para a aplicação.
4. Elaboração de uma datagrid com a tabela de vendedores e a criação dos métodos de inserção, alteração, eliminação e selecção.
5. Reunião com os outros dois estagiários, Miguel e Cátia, e com a Eng. Mónica Santos e Eng. Dulce, de forma a organizar a repartição do trabalho pelos 3 estagiários.
    - Fiquei responsável pela criação de uma classe que juntasse os métodos já criados pelos outros dois estagiários, que estavam a desenvolver uma datagrid genérica, de forma a ser possível juntar o código dos dois e criar um código mais limpo e reutilizável.

SEMANA	INÍCIO	FIM
2	7 de Março	11 de Março

TRABALHO REALIZADO

1. Continuação da elaboração da datagrid com a tabela de vendedores, adicionando possibilidade de ordenação.
2. Criação das seguintes classes com o código produzido até à altura pelos outros dois estagiários:
    - AcessoBD.cs
        - Inclui todos os métodos de acesso à base de dados.
    - CRUD.cs
        - Inclui todos os métodos usados para efectuar a inserção, remoção e alteração de registos na datagrid.
    - FormatDataGrid.cs
        - Inclui todos os métodos para a formatação gráfica da datagrid.
    - Utils.cs
        - Inclui outros métodos que não se enquadravam totalmente em nenhuma das outras categorias.

SEMANA	INÍCIO	FIM
3	14 de Março	18 de Março

TRABALHO REALIZADO

1. Criação do layout da página:
   - Definição da estrutura da página;
   - Criação de um esquema de cores e respectivos ficheiros css (folhas de estilo para páginas HTML);
2. Continuação da criação das classes da semana anterior (AcesssoBD.cs, CRUD.cs, FormatDataGrid.cs, Utils.cs)

SEMANA	INÍCIO	FIM
4	21 de Março	24 de Março

TRABALHO REALIZADO

1. Criação do formulário de manutenção de dados:
   - Criação de uma *Datagrid* com os metadados disponíveis na base de dados BDINET;
   - Formatação da Datagrid;
2. Redesenho de icons presentes na toolbar:
   - Tratamento dos icons da aplicação anterior de forma a se enquadrarem neste aplicação;

SEMANA	INÍCIO	FIM
5	28 de Março	1 de Abril

TRABALHO REALIZADO

1. Desenvolvimento das funcionalidades de inserção, alteração e remoção no formulário de manutenção de dados.

SEMANA	INÍCIO	FIM
6	4 de Abril	8 de Abril

TRABALHO REALIZADO

1. Criação do formulário de manutenção de dados *Master/Detail*
   - Criação de uma *Datagrid* por baixo da *Master* com os dados referentes às tabelas relacionadas com a mesma;
   - Formatação da Datagrid;
   - Implementação das funcionalidades de inserção, alteração e remoção na tabela *Detail*

SEMANA	INÍCIO	FIM
7	11 de Abril	15 de Abril

TRABALHO REALIZADO

1. Criação do formulário de geração de listagens:
   - Layout e formatação do formulário;
   - Carregamento de todos os campos sobre os quais é possível realizar pesquisas;
   - Criação das queries em SQL;
   - Criação de uma tabela com os resultados das queries;

SEMANA	INÍCIO	FIM
8	18 de Abril	22 de abril

TRABALHO REALIZADO

1. Criação da função de exportação para os seguintes formatos:
   - Acrobat Reader (pdf)
   - Microsoft Word (rtf)
   - Microsoft Excel (xls)
2. A função anterior permite fazer a exportação de:
   - Tabelas simples;
   - Listagem criada pelo utilizador;
3. Para a criação desta função foi necessário fazer uma pesquisa sobre bibliotecas de funções que permitissem estas exportações. Durante cerca de metade da semana foram realizados testes a diversas dll's disponíveis na Internet de forma a se poder decidir quais as eficazes para a função. No fim optou-se por usar a iTextSharp, uma livraria de funções que permitia exportar para pdf e rtf da mesma forma, disponível sob uma licença GNU.

SEMANA	INÍCIO	FIM
9	26 de Abril	29 de Abril

TRABALHO REALIZADO

1. Criação de relatórios:
    - Testes com a ferramenta "Crystal Reports", incluída no Visual Studio 2005;
    - Criação de um relatório teste (Relatório de Inventário);
2. Melhoramentos na função de exportação:
    - Exportação de formulários Master-Detail;
    - Arranjo do layout da exportação para pdf e rtf;
    - Correcção de erros na função de exportação para Excel;
    - Exportação para email;

SEMANA	INÍCIO	FIM
10	2 de Maio	6 de Maio

TRABALHO REALIZADO

1. Criação de um layout para a calculadora em flash da aplicação:
    - Criação dos botões e outros icons visuais;
    - Arranjo gráfico dos anteriores;
2. Melhoramentos na função de exportação:
    - Possibilidade de exportar a partir de qualquer ponto na aplicação a tabela apresentada no ecrã;
    - Criação de um menu de exportação;
    - Correcção de erros na função de exportação para Excel;
    - Exportação para email;

SEMANA	INÍCIO	FIM
11	9 de Maio	13 de Maio

TRABALHO REALIZADO

1. Layout da página:
   - Ajuste automático da dimensão das datagrids, quando se esconde o menu;
2. Integração da form de preferências de utilizador.
3. Correcção de erros da função de listagem.
4. Correcção de erros de navegação de forms:
   - Permitir manter o form e modo seleccionado quando se esconde o menu;
   - Manutenção de uma consistência nos forms que foram abertos permitindo que se possa garantir que todos os forms que tenham sido abertos, sejam fechados;

SEMANA	INÍCIO	FIM
12	16 de Maio	20 de Maio

TRABALHO REALIZADO

1. Layout da página:
   - Correcção de bugs;
2. Criação do formulário de alteração de password.
3. Pesquisa de material sobre:
   - Smart Clients;
   - Web Services;
4. Criação de um Smart Client (Windows Application) e de um Web Service, e ligação entre os dois:
   - Acesso inicial a web services disponíveis na Internet (Google);
   - Criação de um web service de acesso à base de dados BDINET;

SEMANA	INÍCIO	FIM
13	23 de Maio	27 de Maio

TRABALHO REALIZADO

1. Continuação dos testes com Smart Clients e Web ServicesLayout da página:
2. Instalação, configuração e testes com uma impressora utilizada no projecto de Bilhética.

Semana	Início	Fim
14	30 de Maio	3 de junho

Trabalho realizado

1. Criação de uma Gateway de pagamentos para a aplicação Accessu .NET
   - Análise de requisitos;
   - Criação de layouts de teste;
   - Criação de uma aplicação Smart Client, acessível através da rede;
2. Trabalho no layout dos restantes módulos da aplicação.

Semana	Início	Fim
15	6 de Junho	9 de Junho

Trabalho realizado

1. Continuação do trabalho no layout da aplicação.
2. Criação de métodos de criação de documentos (Vendas a dinheiro, talões de venda) na base de dados.
3. Reunião na empresa ARMIS – Sistemas de Informação, Lda, no dia 9 de Junho, com algumas pessoas com experiência na área de desenvolvimento de aplicações .NET, para discutir algumas sugestões no desenvolvimento do projecto Accessu .NET.

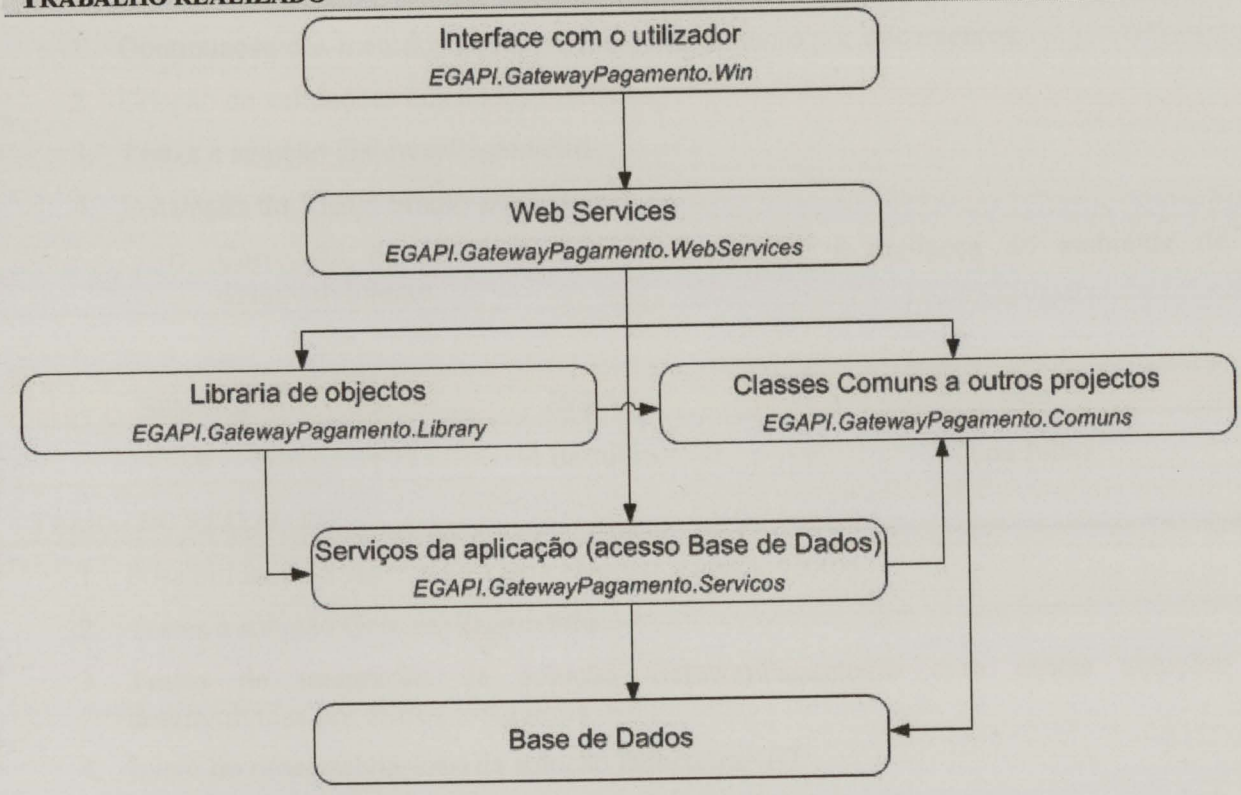
Semana	Início	Fim
16	13 de Junho	17 de Junho

Trabalho realizado

1. Reestruturação da solução criada. Criação de uma nova solução, GatewayPagamento, com os seguintes projectos:
    - EGAPI.GatewayPagamento.Win
        - Contem todos os Windows Forms e User Controls da aplicação;
    - EGAPI.GatewayPagamento.Library
        - Contem todos os objectos utilizados pela solução (Documento, Pagamento, entre outros);
    - EGAPI.GatewayPagamento.Servicos
        - Contem as classes utilizadas por toda a aplicação (acesso à base de dados, por exemplo);
    - EGAPI.GatewayPagamento.Comum
        - Contem classes usadas em outros projectos e soluções da empresa;
    - EGAPI.GatewayPagamento.WebService
        - Contem os métodos essenciais da aplicação, de forma a estarem disponíveis por HTTP;
2. Durante esta semana trabalhei essencialmente sobre o projecto EGAPI.GatewayPagamento.Win e EGAPI.GatewayPagamento.Library, criando todos os objectos necessários para a aplicação, bem como os métodos necessários para a sua manipulação.

SEMANA	INÍCIO	FIM
17	20 de junho	24 de Junho

TRABALHO REALIZADO



1. Continuação da reestruturação da solução anterior:
   - Implementação de uma solução com a seguinte estrutura:
   - Esta estrutura vem no seguimento da reestruturação efectuada a semana passada, mantendo cada um dos os projectos a sua funcionalidade, ficando assim mais especificada a função de cada projecto:
     - O projecto *Library* funciona como uma extensão ao projecto *WebServices*, uma vez que contem os objectos disponibilizados pelo segundo;
     - O projecto *Serviços* disponibiliza, neste momento, essencialmente métodos de acesso à Base de Dados, mas deverá disponibilizar todos os métodos necessários para a aplicação;
     - O projecto *Comuns* contem classes utilizadas em outros projectos, e desta forma pode também aceder à Base de Dados.
2. Criação de métodos para o tratamento das promoções.
3. Criação de métodos para repartir métodos de pagamento por diferentes documentos e atribuição de troco.

SEMANA	INÍCIO	FIM
18	27 de Junho	1 de Julho

TRABALHO REALIZADO

1. Continuação dos métodos de repartição de pagamento por documentos.
2. Criação de validações dos dados inseridos.
3. Testes à solução GatewayPagamento.
4. Instalação do Visual Studio 2005 Beta 2:
    - Correcção de erros de compilação devido à mudança do ambiente de desenvolvimento.

SEMANA	INÍCIO	FIM
19	4 de Julho	8 de Julho

TRABALHO REALIZADO

1. Possibilidade de usar crédito para efectuar os pagamentos.
2. Testes à solução GatewayPagamento.
3. Testes de integração da solução GatewayPagamento com outras soluções desenvolvidas por outros estagiários.
4. Inicio do desenvolvimento da solução BilheteiraNET:
    - Visualização do estádio numa das áreas do ecrã e noutra a zona seleccionada na anterior, com informação detalhada sobre a ocupação dos lugares.
    - Carregamento de alguns dados da base de dados sobre a localização de sectores do estádio na imagem carregada.

SEMANA	INÍCIO	FIM
20	11 de Julho	15 de Julho

TRABALHO REALIZADO

1. Representação, numa matriz, o estado (ocupado, livre, reservado ou preferencial) de cada lugar existente num subsector do estádio.
2. Criação de um User Control que imite a funcionalidade de um Image Map utilizado em ambiente Web:
    - Método para a criação de áreas na imagem, onde seja possível clicar nessas áreas como se fosse um botão;
    - Métodos para a remoção dessas áreas;
3. Organização gráfica da aplicação.
4. Estruturação do problema da criação de novas instalações (no mesmo estádio definir, para alturas distintas, diferentes zonas, sectores, subsectores e lugares).

SEMANA	INÍCIO	FIM
21	18 de Julho	22 de Julho

TRABALHO REALIZADO

1. Avaliação da solução criada.
2. Representação em *comboboxes* dos sectores do estádio que, devido ao seu reduzido tamanho, não é possível representar na imagem.

SEMANA	INÍCIO	FIM
22	25 de Julho	29 de Julho

TRABALHO REALIZADO

1. Criação de uma classe, que possa ser utilizada em todos os projectos, de *Logs*.
2. Reunião durante todo o dia de 26 de Julho, terça-feira, com um representante da Microsoft de forma a expor algumas soluções passíveis de serem utilizadas no projecto Acessu .NET.

SEMANA	INÍCIO	FIM
23	1 de Agosto	5 de Agosto

TRABALHO REALIZADO

1. Testes à aplicação Accessu .NET;
2. Correcção de *bugs*;

SEMANA	INÍCIO	FIM
24	8 de Agosto	12 de Agosto

TRABALHO REALIZADO

(Semana de dispensa)

SEMANA	INÍCIO	FIM
25	16 de Agosto	19 de Agosto

TRABALHO REALIZADO

1. Migração de uma base de dados Oracle para SQL Server 2005:
    - Criação de tabelas e as suas relações e restrições;
    - Migração de Packages Oracle para T-SQL;

SEMANA	INÍCIO	FIM
26	22 de Agosto	26 de Agosto

TRABALHO REALIZADO

1. Reestruturação do *layout* da aplicação:
   - Normalização do esquema de cores com outros protótipos realizados;
   - Reorganização dos objectos no ecrã;
2. Utilização de *Background Workers* (utilização de outras *threads* para executar algum trabalho mais "pesado") para permitir a visualização de barras de progresso;
3. Possibilidade seleccionar diversos lugares pressionando a tecla *Shift* do teclado;

SEMANA	INÍCIO	FIM
27	29 de Agosto	31 de Agosto

TRABALHO REALIZADO

1. Implementação de diversos pormenores relativos à interface com o utilizador;
2. Testes à aplicação;
3. Correcção de *bugs*;

ANEXO B: Modelo da base de dados BDINET

BDI_TIPOS_MENSAGENS

PK,I1	TMSG_CODIGO
	TMSG_DESCRICAO
	TMSG_ICON

BDI_MODULOS_RELATORIOS

PK,FK1,I1	MODRE_MOD_CODIGO
PK,FK2,I1	MODRE_RE_ID
PK,FK2,I1	MODRE_RE_RELEASE
	MODRE_FLAG

BDI_UTILIZADORES_APP_PERFIS

PK,FK2,I1	UAP_UA_UT_ID
PK,FK1,FK2,I1	UAP_UA_APP_ID
PK,FK1,I1	UAP_PRF_ID

BDI_UTILIZADORES_APLICACOES

PK,FK3,I1	UA_UT_ID
PK,FK2,I1	UA_APP_ID
FK2	UA_MOD_CODIGO
FK1	UA_EMPL_EMP_CODIGO
FK1	UA_EMPL_CODIGO

BDI_TABPAI

PK,FK1,FK2,I1	PAI_TA_APP_ID
PK,FK3,I1	PAI_TA_TAB_ID_PAI
PK,FK2,I1	PAI_TA_TAB_ID_FILHO
	PAI_ORDEM

BDI_EMPRESAS_LOCAIS

PK,FK1,I1	EMPL_EMP_CODIGO
PK,FK2,I1	EMPL_EMP_APP_ID
PK,I1	EMPL_CODIGO
	EMPL_DESIGNACAO
	EMPL_STANDALONE
	EMPL_FLAG

BDI_EMPRESAS

PK,FK1,I1	EMP_APP_ID
PK,I1	EMP_CODIGO
	EMP_DESIGNACAO

BDI_LICENCAS

PK,I1	LI_CODIGO
	LI_TIPO
	LI_NUMBER1
	LI_TABELA1
	LI_NUMBER2
	LI_TABELA2
	LI_NUMBER3
	LI_TABELA3

BDI_MENSAGENS

PK,FK1,FK6,FK7,FK8,I1	MSG_APP_ID
PK,I1	MSG_CODIGO
FK10	MSG_TMSG_CODIGO
	MSG_BOTAO1
FK2	MSG_DIC_INDICE1
FK2	MSG_DIC_LNG_CODIGO1
FK7	MSG_MN_CODIGO1
	MSG_BOTAO2
FK3	MSG_DIC_INDICE2
FK3	MSG_DIC_LNG_CODIGO2
FK8	MSG_MN_CODIGO2
	MSG_BOTAO3
FK4	MSG_DIC_INDICE3
FK4	MSG_DIC_LNG_CODIGO3
FK9	MSG_MN_CODIGO3
	MSG_MENSAGEM
FK5	MSG_DIC_INDICE
FK5	MSG_DIC_LNG_CODIGO
	MSG_FLAG

BDI_PERMISSOES_RELATORIOS

PK,FK2,I1	PER_PRF_ID
PK,FK1,I1	PER_MOD_CODIGO
PK,FK1,I1	PER_RE_ID
PK,FK1,I1	PER_RE_RELEASE
PK,FK2,I1	PER_PEL_EMPL_EMP_CODIGO
PK,FK2,I1	PER_PEL_EMPL_CODIGO
	PER_CRIACAO
	PER_DATA_CRIACAO
	PER_ALTERACAO
	PER_DATA_ALTERACAO
	PER_FLAG

BDI_UTILIZADORES

PK,I1	UT_ID
	UT_CODIGO
	UT_DESIGNACAO
FK1	UT_LNG_CODIGO
	UT_MOEDA
	UT_CRIACAO
	UT_DATA_CRIACAO
	UT_ALTERACAO
	UT_DATA_ALTERACAO
	UT_FLAG
	UT_PASSWORD
	UT_FORMATO_DATA

BDI_POSTOS

PK,I1	PO_ID
FK1	PO_EMPL_EMP_CODIGO
FK1	PO_EMPL_EMP_APP_ID
FK1	PO_EMPL_CODIGO
	PO_NOME
	PO_DATA_REGISTO
	PO_DATA_ULTIMO_ACESSO
	PO_ACESSOS
	PO_OBS
	PO_VALOR1
	PO_VALOR2
	PO_VALOR3
	PO_VALOR4
	PO_VALOR5

BDI_MODULOS

PK,FK1,FK4,I1	MOD_APP_ID
PK,I1	MOD_CODIGO
	MOD_DESIGNACAO
FK2	MOD_DIC_INDICE
FK2	MOD_DIC_LNG_CODIGO
	MOD_FLAG
FK3	MOD_LI_CODIGO
	MOD_ORDEM
	MOD_ICON
FK4	MOD_MOD_CODIGO
	MOD_TIPO

BDI_BDIID

PK,I1	BDI_COD_CLIENTE
PK,FK1,I1	BDI_APP_ID
	BDI_NOME
	BDI_NUMERO_SERIE
	BDI_BASE_DIR
	BDI_FORMS_DIR
	BDI_LISTAS_DIR
	BDI_IMAGENS_DIR
	BDI_SQL_DIR
	BDI_TEMP_DIR
	BDI_AJUDA
	BDI_USA_INTERVALO
	BDI_INTERVALO
	BDI_LIN0
	BDI_LIN1
	BDI_LIN2
	BDI_LIN3
	BDI_LIN4
	BDI_LIN5
	BDI_LIN6
	BDI_LIN7
	BDI_LIN8
	BDI_ICONS_DIR

BDI_MENUS

PK,I1	MN_CODIGO
PK,FK2,FK4,I1	MN_MOD_APP_ID
FK3,FK4	MN_MOD_CODIGO
	MN_DESIGNACAO
FK1	MN_DIC_INDICE
FK1	MN_DIC_LNG_CODIGO
	MN_ORDEM
	MN_MN_CODIGO
	MN_FM_CODIGO
	MN_ICON
FK3	MN_RE_ID
FK3	MN_RE_RELEASE

BDI_PERFIS_EMPRESAS_LOCAIS

PK,FK3,I1	PEL_PRF_ID
PK,FK1,FK4,I1	PEL_PRF_APP_ID
PK,FK2,I1	PEL_EMPL_EMP_CODIGO
PK,FK2,I1	PEL_EMPL_CODIGO

BDI_LINGUAS

PK,I1	LNG_CODIGO
	LNG_DESIGNACAO
	LNG_FLAG

BDI_TABELAS_APLICACOES

PK,FK2,I1	TA_TAB_ID
PK,FK1,I1	TA_APP_ID
	TA_ORDEM
	TA_OWNER

BDI_APLICACOES

PK,I1	APP_ID
	APP_CODIGO
	APP_DESIGNACAO
	APP_IMAGE
	APP_COR
FK1	APP_STYLESHEET
	APP_LNG_CODIGO
	APP_ORDEM
	APP_AUTOLOGIN
	APP_FLAG
	APP_COPYRIGHT
	APP_ASPMAIN
	APP_DESCRICAO

BDI_TABELAS

PK,I1	TAB_ID
	TAB_CODIGO
	TAB_TABLE_SPACE
	TAB_OPCOES_TAMANHO
	TAB_TIPO
	TAB_SQL_VIEW
	TAB_VOLUME
	TAB_EMPLOCAL
	TAB_EMPLOCAL_TAB
	TAB_DESIGNACAO
FK1	TAB_DIC_INDICE
FK1	TAB_DIC_LNG_CODIGO

BDI_CONEXOES

PK,FK1,I1	CON_APP_ID
PK,I1	CON_ID
	CON_CONNECTION_STRING
	CON_DBLINK
	CON_FLAG
	CON_BD

BDI_PERFIS_ACESSOS

PK,FK4,I1	PAC_PRF_ID
PK,FK1,FK3,FK5,I1	PAC_APP_ID
PK,FK3,I1	PAC_MOD_CODIGO
PK,FK2,I1	PAC_MN_CODIGO
	PAC_CRIACAO
	PAC_DATA_CRIACAO
	PAC_ALTERACAO
	PAC_DATA_ALTERACAO
	PAC_FLAG

BDI_RELATORIOS

PK,I1	RE_ID
PK,I1	RE_RELEASE
	RE_CODIGO
	RE_DESIGNACAO
FK1	RE_DIC_INDICE
FK1	RE_DIC_LNG_CODIGO
	RE_SQL

BDI_PERFIS

PK,FK1,I1	PRF_APP_ID
PK,I1	PRF_ID
	PRF_DESIGNACAO
FK2	PRF_DIC_INDICE
FK2	PRF_DIC_LNG_CODIGO
	PRF_ROLE_DB
	PRF_CRIACAO
	PRF_DATA_CRIACAO
	PRF_ALTERACAO
	PRF_DATA_ALTERACAO
	PRF_FLAG

BDI_FORMS

PK,I1	FM_CODIGO
PK,FK1,FK3,I1	FM_APP_ID
	FM_DESIGNACAO
FK2	FM_DIC_INDICE
FK2	FM_DIC_LNG_CODIGO
	FM_GERADA
FK4	FM_TA_TAB_ID
	FM_TIPO

BDI_REL_PARAM_RELATORIOS

PK,FK2,I1	PRL_RE_ID
PK,FK2,I1	PRL_RE_RELEASE
PK,FK1,I1	PRL_PR_ID

BDI_PACKETS

PK,I1	PCK_ID
	PCK_CODIGO
	PCK_DESIGNACAO

BDI_BD

PK,I1	BD_NOME_DB
PK,I1	BD_VERSAO
FK1	BD_APP_ID
	BD_OWNER
	BD_PASSWORD
	BD_TAMANHO
	BD_TABLESPACE_DADOS
	BD_TABLESPACE_INDICES
	BD_DBLINK1
	BD_DBLINK2
	BD_TABLESPACE_TEMP

BDI_CAMPOS

PK,FK6,I1	CMP_TAB_ID
PK,I1	CMP_CODIGO
	CMP_NATUREZA
	CMP_TAMANHO
	CMP_DESCRICAO
FK1	CMP_DIC_INDICE
FK1,FK2	CMP_DIC_LNG_CODIGO
	CMP_PAPEL
	CMP_ORDEM
	CMP_OBRIGATORIO
FK5	CMP_FUNCAO_VALIDACAO
	CMP_OBJECTO_ECRAN
	CMP_PARAMETRO_OBJECTO
	CMP_VISIVEL
	CMP_SEQUENCIAL
	CMP_SEQUENCIA
FK4	CMP_FNC_ID
	CMP_LABEL
FK3	CMP_DIC_INDICE_LAB
	CMP_MIGRACAO
	CMP_TAMANHO_PX
	CMP_MASCARA
	CMP_TIPO

BDI_INDICES

PK,FK1,I1	IND_TAB_ID
PK,I1	IND_CODIGO
	IND_TABLESPACE
	IND_CHAVE

BDI_PARAM_REL_ORDENACOES

PK,FK1,I1	ORD_RE_ID
PK,FK1,I1	ORD_RE_RELEASE
PK,FK2,I1	ORD_TAB_ID
PK,I1	ORD_COLUNA
	ORD_ORDEM
	ORD_ORDEM_TIPO

BDI_PARAMETROS_RELATORIOS

PK,I1	PR_ID
	PR_CODIGO
	PR_DESIGNACAO
FK1	PR_DIC_INDICE
FK1	PR_DIC_LNG_CODIGO
	PR_FORM
FK2	PR_TAB_ID
	PR_COLUNA
	PR_CONDICAO
	PR_PARAMETRO1
	PR_PARAMETRO2
	PR_FLAG

BDI_DICIONARIOS

PK,I1	DIC_INDICE
PK,FK1,I1	DIC_LNG_CODIGO
	DIC_TERMO
	DIC_FLAG

BDI_PATCHS

PK,I1	PTC_ID
PK,FK1,I1	PTC_APP_ID
	PTC_DATA
	PTC_TIPO
	PTC_VERSAO
	PTC_VALOR
	PTC_OPCAO1
	PTC_OPCAO2
	PTC_OPCAO3
	PTC_OPCAO4
	PTC_OPCAO5

BDI_OPERACOES_ACESSOS

PK,FK1,I1	OA_OPE_FM_APP_ID
PK,FK2,I1	OA_OPE_FM_CODIGO
PK,I1	OA_OPE_CODIGO
PK,FK3,I1	OA_PEL_PRF_ID
PK,FK3,I1	OA_PEL_EMPL_EMP_CODIGO
PK,FK3,I1	OA_PEL_EMPL_CODIGO

BDI_OPERACOES

PK,I1	OPE_CODIGO
PK,FK2,I1	OPE_FM_CODIGO
PK,FK2,I1	OPE_FM_APP_ID
	OPE_DESIGNACAO
FK1	OPE_DIC_INDICE
FK1	OPE_DIC_LNG_CODIGO
	OPE_TIPO
	OPE_ICON
	OPE_SEPARADOR
FK3	OPE_FNC_ID

BDI_FUNCOES

PK,I1	FNC_ID
	FNC_CODIGO
	FNC_DESCRICAO_FUNCAO
	FNC_TIPO_FUNCAO
	FNC_FICHEIRO
	FNC_SQL_FUNCAO

BDI_CAMPOS_RELACOES

PK,FK1,I1	CR_CMP_TAB_ID
PK,FK1,I1	CR_CMP_CODIGO
PK,FK2,I1	CR_CMP_TAB_ID_REL
PK,FK2,I1	CR_CMP_CODIGO_REL
	CR_RELACAO

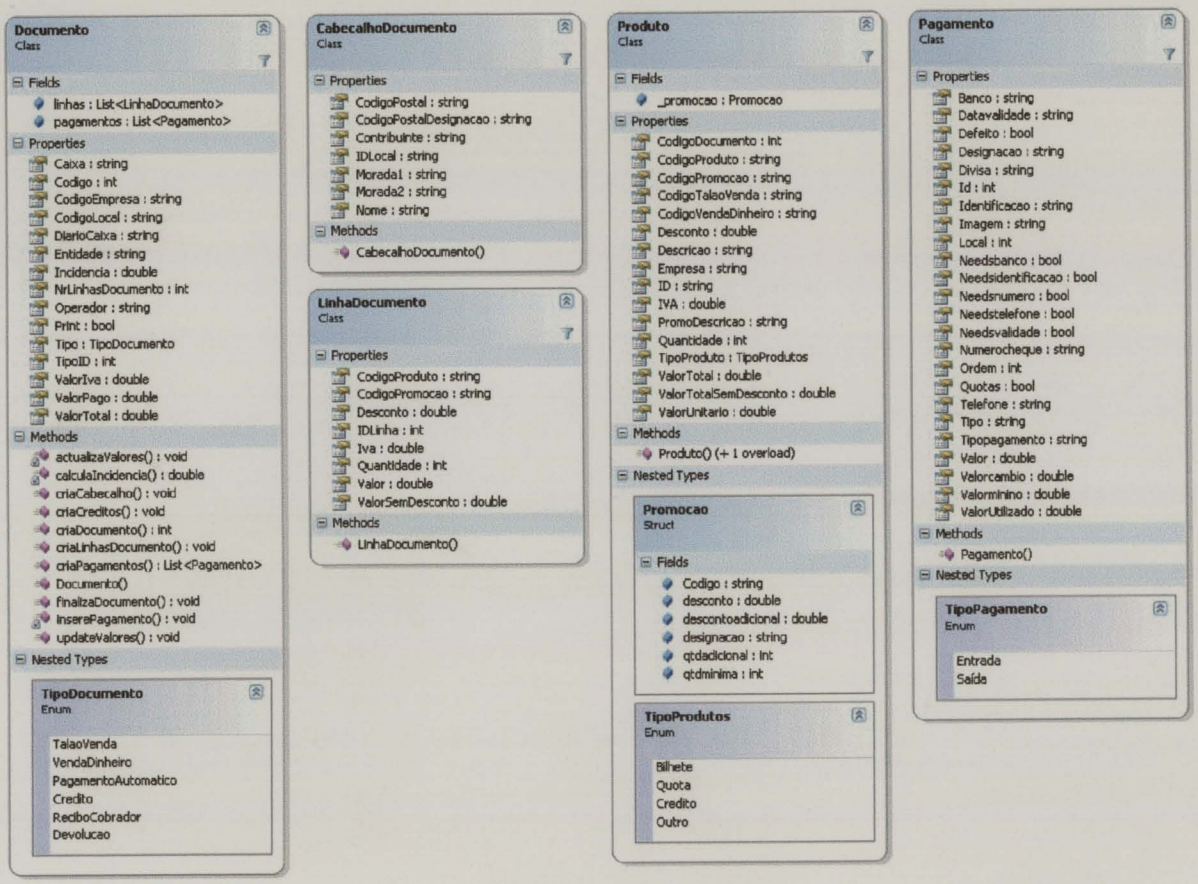
BDI_ACESSOS_CAMPOS

PK,I1	ACC_PRE_ID
PK,FK1,I1	ACC_CMP_TAB_ID
PK,FK1,I1	ACC_CMP_CODIGO

BDI_INDICES_CAMPOS

PK,FK1,FK2,I1	IC_TAB_ID
PK,FK2,I1	IC_IND_CODIGO
PK,FK1,I1	IC_CMP_CODIGO

ANEXO C: Objectos criados no módulo de Pagamentos



ANEXO D: Modelo parcial da base de dados GSTSOC6