Instituto Politécnico de Castelo Branco
Escola Superior de Tecnologia

Desenvolvimento de Aplicação Informática para Controlo e Análise de Custos de Obra

David Manuel Dias Justo de Morais Caldas

Dissertação apresentada ao Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Mestrado em Desenvolvimento de Software e Sistemas Interactivos, realizada sob a orientação científica do Doutor José Carlos Metrôlho, Professor adjunto da Unidade Técnico-Científica de Informática da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Ano 2012

Dedicatória

Carla, Geninha, Cris, Kitty, Gil, O Bando, família e amigos.

Agradecimentos

Agradeço ao meu orientador pela ajuda e incentivo prestado ao longo da realização do projeto, à Eng.ª Patrícia Santos e ao Eng.º Marco Simões por me terem dado a oportunidade de desenvolver o projeto no contexto da sua empresa e, juntamente com a Sara Macedo, me terem apresentado e explicado toda a lógica do negócio.

Agradeço a todos aqueles que direta ou indiretamente me ajudaram na realização do projeto, em especial ao Nuno pelo apoio sempre demonstrado.

À Carla quero agradecer toda a paciência, apoio e incentivo.

Índice

Índice de figuras

Índice de tabelas

Palavras-chave

Obras, Controlo e Análise Custos, Gestão de Obras, *Framework*.

Sumário

Este trabalho, realizado no âmbito do Mestrado em Desenvolvimento de *Software* e Sistemas Interactivos da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco, foca o desenvolvimento de aplicação informática para controlo e análise de custos de obra.

Esta aplicação permite registar todos os custos referentes à realização de uma obra, tornando possível a realização de uma análise criteriosa dos vários custos, potenciando assim a redução de gastos excessivos e uma orçamentação de obras mais realista.

Para uniformizar e agilizar o desenvolvimento deste e de próximos projetos a serem realizados foi desenvolvida uma *framework*, que inclui *templates*, controlos e aplicações.

Keywords

Construction, Cost Control and Analysis, Construction Management, *Framework*.

Abstract

This work, performed in the context of the Master's Degree in Software Development and Interactive Systems at the Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco, focuses the development of a computer application for constructions' cost control and analysis.

This application allows the registry of all costs associated to a construction's tasks, making it possible to perform a criterial analysis of various costs, therefore enabling a reduction in excessive costs and a more realistic construction budgeting.

In order to have a more uniform and agile development of both this and future projects a *framework* was developed, which includes *templates*, controls and applications.

Capítulo 1 - **Introdução**

Controlar os custos para maximizar os lucros é uma das máximas de qualquer empresa. O uso de *software* no controlo de custos é cada vez mais utilizado pelas empresas, mas muitas utilizam ferramentas generalistas (folhas de cálculo, *software* de gestão genérico, etc.) que, por vezes, não dão uma resposta adequada às suas necessidades. Na construção civil são já muitas as empresas que utilizam *software* desenhado especificamente para o ramo, mas outras há que não o fazem, seja porque estes são dispendiosos, porque são complexos, ou porque não se adequam ao modo como estas operam. Este projeto surge como resposta às necessidades da empresa Famirev[1], que não tinha encontrado até à data um *software* à medida das suas necessidades.

A Famirev é uma empresa de construção civil vocacionada para a construção, reabilitação e conservação de edifícios, sediada em Vila Nova de Famalicão.

No mundo atual, as mudanças tecnológicas acontecem a uma velocidade vertiginosa e cada vez mais o desenvolvimento de uma aplicação deve ser não apenas rápido mas também fiável. Para se mantar competitivo um programador deve atualizar-se constantemente e munir-se das ferramentas de desenvolvimento mais adequadas. A definição de estratégias e padrões no desenvolvimento de aplicações é uma mais-valia de um programador. É com base neste pressuposto que surgiu o desafio de desenvolver uma *framework* para agilizar não só para o desenvolvimento deste projeto como também para próximos que possam surgir.

1.1. Resumo dos Capítulos Abordados

Esta dissertação está dividida em oito capítulos nos quais são descritos todos os processos efetuados para a realização deste projeto.

No presente capítulo é feito um enquadramento do tema escolhido e dos objetivos propostos para o projeto.

No capítulo "Estado da Arte" são apresentadas algumas aplicações informáticas existentes no mercado que poderiam dar resposta ao problema apresentado.

O funcionamento da empresa, bem como a modelação do problema em entidades, modelo de dados e dicionário de dados, é descrito no capítulo "Análise do Sistema".

No capítulo "*Aplicação*" são apresentados os objetivos a cumprir pela aplicação desenvolvida, as suas funcionalidades e estrutura.

Para a realização deste projeto optou-se pela criação de uma *framework* para uniformizar e facilitar o desenvolvimento deste e de próximos projetos. No capítulo "*Framework*" são descritos os componentes que constituem a *framework* e o seu modo de funcionamento.

[1] http://www.famirev.pt acedido a 16 de outubro de 2011

No capítulo “Desenvolvimento” é apresentado o modelo adotado para o desenvolvimento da aplicação, as ferramentas utilizadas para o seu desenvolvimento e descritos os detalhes do desenvolvimento da aplicação.

Nos últimos capítulos, “Conclusão” e “Trabalho Futuro”, são apresentadas, respetivamente, as conclusões desta dissertação e algumas considerações sobre possível trabalho futuro.

Capítulo 2 - **Estado da Arte**

Cada empresa é única na sua identidade, organização e modo de funcionamento. Quando se desenvolve um *software* à medida, desenvolve-se uma solução que deve ser capaz de responder às necessidades de uma empresa ou contexto específico. Regra geral, um *software* feito à medida tem menos funcionalidades que um *software* genérico que tenha sido concebido para o mesmo contexto mas para dar resposta a várias realidades (exemplo: para diferentes empresas). Um dos problemas que quase sempre se coloca quando se adquire um *software* genérico já existente é a necessidade da empresa se ajustar a este e não o contrário. O desenvolvimento de um *software* à medida cria também a oportunidade de reavaliar os processos de negócio, no sentido de os otimizar.

No mercado atual existe *software* destinado ao controlo e análise de custos de obra. Numa primeira análise poderíamos dizer que estes dão resposta às necessidades da empresa mas, como iremos ver neste capítulo, tal não se verifica.

No presente caso de estudo, a empresa tinha necessidade de um *software* simples, que permitisse controlar as compras, pagamentos e adiantamentos a fornecedores, distribuição pelas várias obras do material comprado, registo de todos os gastos existentes nas obras, orçamentos, controlar faturas e recebimentos das obras. Uma vez que a empresa já possuía um *software* de faturação não era requisito do sistema gerir a mesma. De seguida apresento algumas aplicações relacionadas com a que foi desenvolvida neste projeto:

Primavera Construction (PRIMAVERA BSS, 2011): *software* desenvolvido pela Primavera BSS[2] destinado a empresas de construção civil e obras públicas e tem como principais funcionalidades:

- Orçamentos;
- Planeamentos;
- Subempreitadas;
- Autos de medição;
- Contratos adicionais.

LiveMarket Obras (LiveSolutions, 2011): desenvolvido pela Livesolutions[3] este *software* destina-se a empresa que realizam obras públicas ou privadas e tem como principais funcionalidades:

- Orçamentos;
- Planeamentos;

[2] http://www.primaverabss.com acedido a 25 de novembro de 2011

[3] http://www.livesolutions.pt acedido a 25 de novembro de 2011

- Mão-de-obra / Máquinas e Viaturas;
- Despesas e Custos Diversos;
- Gestão de Obra;
- Gestão de Contratos;
- Gestão Materiais;
- Distribuição de Custos Indiretos;
- Faturação a Clientes.

OranGest Obras (Magnisoft, 2011): desenvolvido pela Magnisoft[4] este *software* está direcionado para as empresas de construção civil e áreas de negócio que envolvam empreitadas como: Eletricidade, Pinturas, Canalizações, Carpintarias, Projetos, entre outras. Tem como principais funcionalidades a Gestão de:

- Obras;
- Materiais;
- Pessoal;
- Equipamentos;
- Orçamentos;
- Autos de Medição.

Gestão de Obras (Improxy, 2011): desenvolvido pela Improxy[5] este *software* destina-se a empresa que realizam obras públicas ou privadas e tem como principais funcionalidades:

- Controlo e gestão orçamental de cada obra;
- Tratamento comercial e financeiro das vendas;
- Gestão bancária e contas correntes de fornecedores e clientes;
- Agenda e gestão de tarefas pendentes e realizadas;
- Gestão de recursos;
- Orçamentação detalhada e autos de mediação.

Todas as soluções apresentadas anteriormente se integram com os ERPs desenvolvidos pelas respetivas empresas, sendo assim soluções mais abrangentes. No entanto, o seu elevado preço de aquisição, instalação e manutenção leva a que muitas empresas não possam adquirir estas soluções.

[4] http://www.magnisoft.pt acedido a 25 de novembro de 2011

[5] http://www.improxy.pt acedido a 26 de novembro de 2011

Depois de analisadas as soluções existentes no mercado e ponderada a aquisição ou não de uma destas, optou-se pelo desenvolvimento de um *software* à medida. Tal ficou a dever-se não apenas aos custos associados à aquisição e utilização dessas soluções, mas também ao facto de, segundo a experiência dos responsáveis da empresa na utilização dessas mesmas soluções, nenhuma se enquadrar na perfeição às necessidades e lógica de funcionamento da empresa. Foi também expressa pela empresa a necessidade de ter uma solução perfeitamente adaptada às suas necessidades e características, quer no que respeita a funcionalidades quer no que respeita à disposição do interface. Exemplo disso são as funcionalidades que permitem a integração entre compras de materiais e sua distribuição por diferentes obras. O desenvolvimento de um *software* à medida traria ainda a vantagem adicional de permitir a evolução da aplicação consoante as necessidades da empresa assim o exigissem.

Capítulo 3 - **Análise do Sistema**

Neste capítulo são apresentados os objetivos da aplicação, a análise dos requisitos e a modelação dos dados.

3.1. Descrição do Sistema

Na Figura 3.1 está representado um diagrama de Caso de Uso. Este permite ter uma panorâmica do modelo do negócio, que será detalhado de seguida.

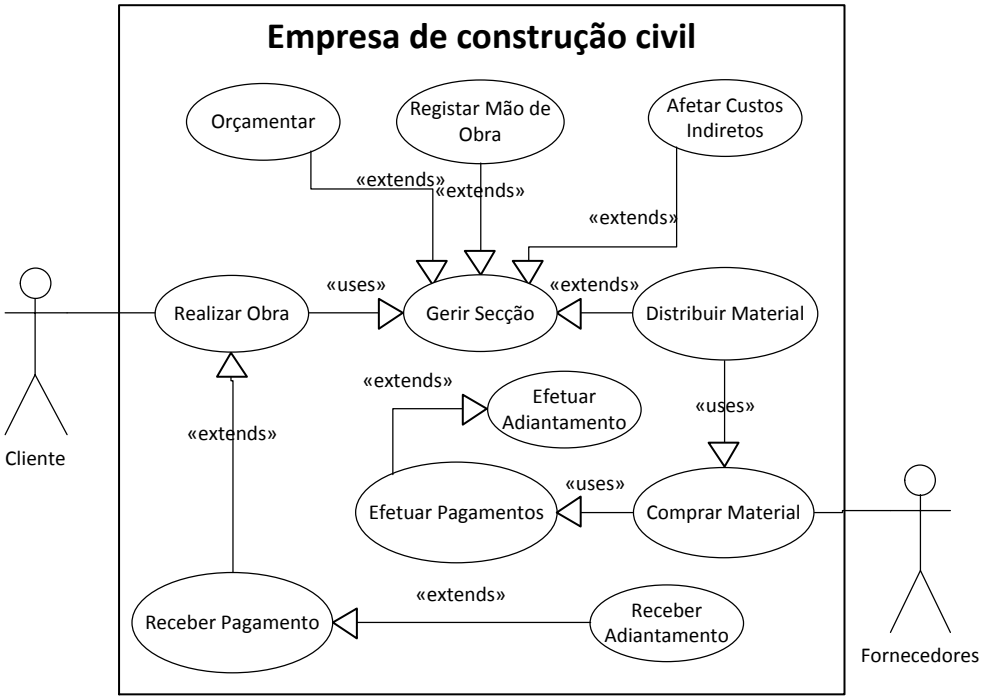


Figura 3.1: Diagrama de Caso de Uso – Modelo de Negócio

Quando um cliente requisita os serviços da empresa FamiRev, para realização de uma empreitada, é necessário abrir a ficha de obra. Nessa ficha é necessário especificar as várias secções (carpintaria, pichelaria, etc.) que serão intervencionadas pela empresa. Para cada uma destas secções é possível registar os orçamentos, a mão-de-obra (horas/custo) e os custos indiretos (telemóvel, seguros, etc.).

Para realizar uma obra é necessário comprar material. A compra de material a um fornecedor pode não destinar-se apenas a uma obra mas a várias obras. É por isso necessário controlar, em cada compra, qual a obra a que se destina determinado artigo e a respetiva quantidade. É também preciso ter em conta que os artigos têm várias unidades de compra. Por vezes durante a realização da obra o material não é todo utilizado e este pode ser devolvido ao fornecedor, sendo necessário controlar a devolução de material aos fornecedores.

As obras e as compras podem ser pagas em várias *tranches*, sendo gerada uma fatura sempre que é necessário que o cliente efetue o pagamento de uma *tranche*. Ao criar uma fatura é gerada uma dívida por parte do cliente para com a empresa. Ao longo da obra um cliente pode

ter uma ou mais faturas a liquidar, sendo necessário controlar o que já foi pago e o que falta ainda pagar. As faturas podem ser liquidadas na totalidade ou parcialmente. Sempre que um cliente paga uma ou mais faturas, é gerado um recibo que liquida o débito do cliente para com a empresa. Existe ainda a possibilidade de um cliente fazer adiantamentos para a realização da obra. Quando é criado um recibo para o cliente, pode ser especificado um adiantamento e o valor a utilizar, sendo esse valor abatido no saldo disponível desse adiantamento. Também existe adiantamentos entre a empresa e os fornecedores, que permite à empresa adiantar verbas para o pagamento dos materiais e sempre que é efetuado um pagamento a um fornecedor, pode ser especificado um adiantamento e o valor a utilizar, sendo esse valor abatido no saldo disponível desse adiantamento. O mesmo acontece com o pagamento a fornecedores, em que é também possível especificar qual o adiantamento e o valor a utilizar. As obras, as compras e a faturas ficam fechadas quando tiverem sido liquidadas na totalidade. Caso não exista essa possibilidade, por falta de boa cobrança ou motivos excecionais, estas podem ainda ser fechadas manualmente.

Relativamente à mão-de-obra, um funcionário pode ter vários tipos de hora (Normal, Sábado, Domingo/Feriados, etc.) o que faz variar o custo/hora. Ao efetuar o registo das horas dos funcionários nas obras é necessário especificar que tipo de hora se está a aplicar.

Perante os objetivos propostos, e a informação que a empresa usa, a aplicação deverá permitir executar as seguintes operações:

- **Gestão de Clientes**
    - Gerir dados pessoais dos clientes da empresa
    - Controlar pagamentos em falta
- **Gestão de Fornecedores**
    - Gerir dados pessoais dos fornecedores da empresa
    - Controlar pagamentos em falta
- **Gestão de Funcionários**
    - Gerir dados pessoais dos funcionários da empresa
- **Gestão de Artigos**
    - Gerir dados dos artigos
- **Gestão de Custos Indiretos**
    - Gerir custos indiretos que uma obra possa ter
- **Gestão de Secções**
    - Gerir secções que uma obra possa ter
- **Gestão de Compras**

- Registar compras efetuadas a fornecedores e a distribuição das várias mercadorias pelas obras correspondentes
- Controlar pagamentos e adiantamentos

- **Gestão de Obras**
  - Gerir dados referentes a cada obra
  - Controlar compras, custos indiretos e mão-de-obra
  - Controlar pagamentos e adiantamentos

3.2. Modelação do Problema

Após a fase de análise do problema passou-se à fase de modelação do problema.

3.2.1. Entidades

Entidade é uma abstração para a descrição de objetos, ou conceitos, que possuem um conjunto de características comuns, ou seja, pode ser um objeto com existência física (exemplos: uma pessoa, uma casa ou um funcionário) ou pode ter apenas existência conceptual (exemplos: uma empresa, um emprego, um curso universitário).

Com base na descrição realizada na análise do problema foram identificadas as seguintes entidades:

- **Entidade Artigos**
  - Regista dados referentes a artigos.
- **Entidade Clientes**
  - Regista dados referentes a clientes.
- **Entidade Compras**
  - Regista dados referentes à compra de materiais a fornecedores.
- **Entidade Compras_Linhas**
  - Regista dados referentes à ligação existente entre a compra de material e os artigos, nomeadamente a quantidade adquirida e a que preço.
- **Entidade Custos_Indirectos**
  - Regista dados referentes a custos indiretos.
- **Entidade Facturas**
  - Regista dados referentes às faturas a clientes.
- **Entidade Fornecedores**
  - Regista dados referentes a fornecedores.

- **Entidade Fornecedores_Adiantamentos**
    - Regista dados referentes a adiantamentos a fornecedores.
- **Entidade Funcionarios**
    - Regista dados referentes aos funcionários.
- **Entidade Funcionarios_Custo_Hora**
    - Regista valores de custo de hora por funcionário para cada tipo de hora em que este trabalha.
- **Entidade Obras**
    - Regista dados referentes a obras.
- **Entidade Obras_Adiantamentos**
    - Regista dados referentes a adiantamentos feitos pelo cliente.
- **Entidade Obras_Seccoes**
    - Regista dados referentes a secções de obra.
- **Entidade Obras_Seccoes_Compras_Linhas**
    - Regista dados referentes a compras/artigos de cada secção.
- **Entidade Obras_Seccoes_Custos_Indirectos**
    - Regista dados referentes a custos indiretos de cada secção.
- **Entidade Obras_Seccoes_Funcionarios**
    - Regista dados referentes à mão-de-obra de cada secção.
- **Entidade Obras_Seccoes_Orcamentos**
    - Regista dados referentes a orçamentos de cada secção.
- **Entidade Pagamentos**
    - Regista dados referentes a pagamentos a fornecedores.
- **Entidade Pagamentos_Compras**
    - Regista que compras foram pagas por um determinado pagamento.
- **Entidade Recibos**
    - Regista dados referentes a recibos de clientes.
- **Entidade Recibos_Facturas**
    - Regista faturas liquidadas por recibo.
- **Entidade Seccoes**
    - Regista dados referentes às secções.

- **Entidade Tipos_Hora**
    - Regista dados referentes a tipos de hora de funcionários.
- **Entidade Unidades**
    - Regista dados referentes a unidades de artigos.

3.2.2. Modelo de dados

A informação de uma base de dados pode ser organizada através de um Modelo de Dados (Sommerville, 2007). O Modelo de Dados define um conjunto de conceitos para a representação dos dados de uma entidade.

A relação existente entre as diferentes entidades, no âmbito deste projeto, pode ser visualizada através do Modelo de Dados representado nas figuras seguintes. Dado a dimensão do Modelo de Dados, optou-se por dividi-lo em três partes. Na primeira parte (Figura 3.2) estão representados os dados e relações referentes a fornecedores, artigos, compras, pagamentos e adiantamentos a fornecedores.

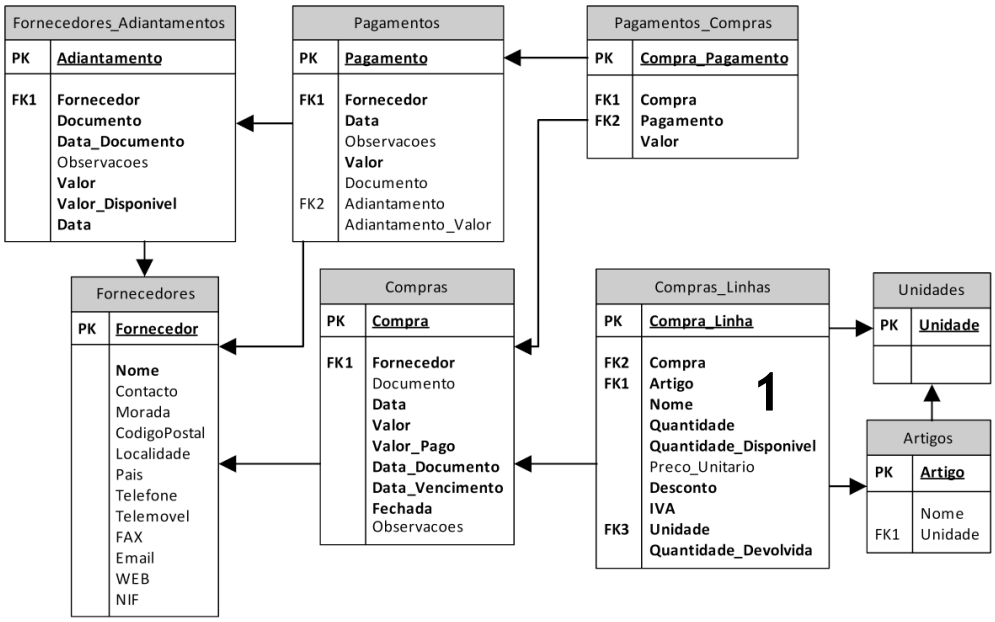


Figura 3.2: Modelo de dados referente às compras de material e pagamento a fornecedores

Na segunda parte (Figura 3.3) estão representados os dados e relações entre compras e obras.

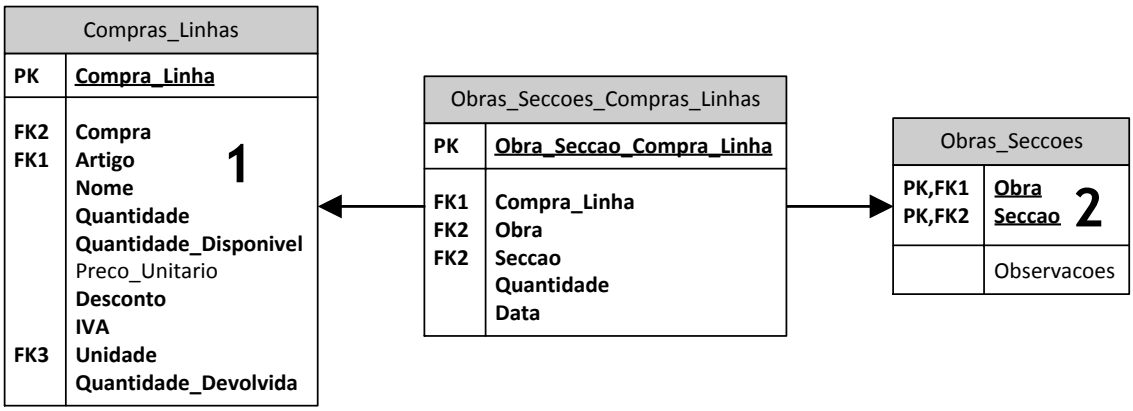


Figura 3.3: Modelo de dados referente à distribuição do material pelas secções de obra

Na terceira parte (Figura 3.4) estão representados os dados e relações entre obras, secções, custos indiretos, orçamentos, mão-de-obra, funcionários, clientes, faturas e recibos.

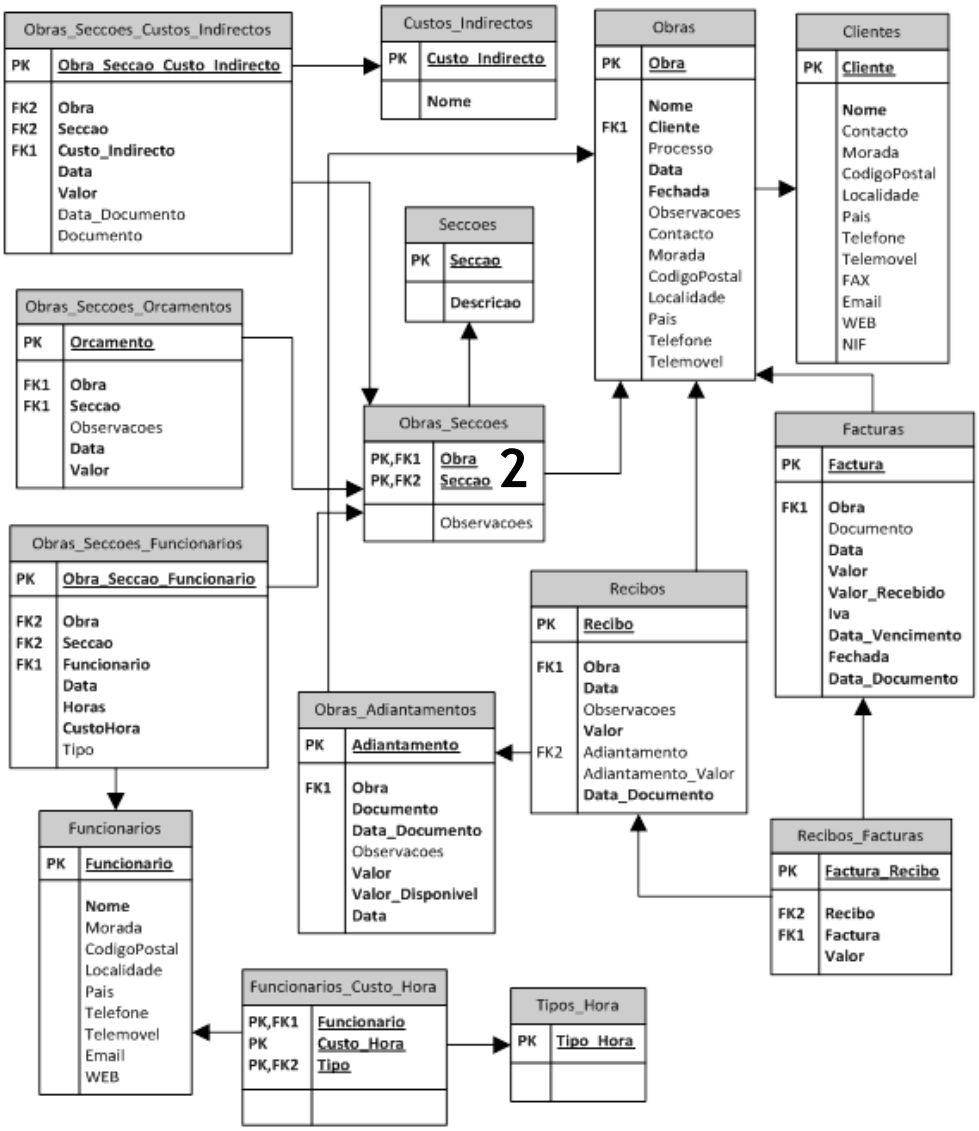


Figura 3.4: Modelo de dados referente às obras e ao pagamento de clientes

As três partes estão ligadas entre si por uma tabela em comum que, nas figuras anteriores, estão identificadas por um 1 na ligação entre a primeira e a segunda parte e por um 2 na ligação entre a segunda e a terceira parte.

3.2.3. Dicionário de Dados

Um dicionário de dados (Sommerville, 2007) possui as definições dos elementos e conceitos pertinentes do sistema de informação. Sem o dicionário de dados, o modelo de dados não pode ser considerado completo. Este deve poder ser entendido por pessoas leigas no assunto. O dicionário de dados possui a definição formal dos elementos que constituem o modelo de dados e nele estão descritas as relações, os tipos de dados, tamanho e restrições de integridade.

Uma restrição de integridade (Silva & Videira, 2001) permite-nos garantir a consistência dos dados num modelo de dados relacional. É uma condição especificada no esquema da base de dados que restringe os dados que podem ser removidos, ou alterados, numa instância da base de dados. Para existir uma restrição de integridade mais coerente é necessário que exista uma restrição chave entre tabelas e dentro da própria tabela. Existem três tipos de chave:

- **Chave Primária:** Permite identificar um registo da tabela em qualquer parte da base de dados.
- **Chave Composta:** Enquanto a chave primária utiliza um único campo para identificar o registo, numa chave composta são utilizados dois ou mais campos para construir a chave (*Ex: Código+Nome*).
- **Chave Estrangeira:** Trata-se de um subconjunto constituído por um ou mais atributos que são chaves primárias de outras tabelas.

De seguida apresenta-se o dicionário de dados para este caso de estudo. Nele são apresentadas as tabelas criadas e os seus atributos. Para cada um dos atributos é apresentado o nome, o tipo e, se necessário, o seu tamanho. É também indicado se o seu preenchimento é obrigatório (**Obrig.**) e se é chave primária (**CP**) ou chave estrangeira (**CE**).

Artigos:

	Campo	Tipo	Descrição	Obri.
CP	Artigo	Texto(15)	Código do artigo.	*
	Nome	Texto (50)		*
	Unidade	Texto (5)		*

Tabela 3.1: Tabela de dados relativos a artigos

Clientes:

	Campo	Tipo	Descrição	Obri.
CP	Cliente	Texto (15)	Código do Cliente.	*
	Nome	Texto (50)		*
	Contacto	Texto (50)		
	Morada	Texto (100)		
	CodigoPostal	Texto (15)		
	Localidade	Texto (30)		

	Pais	Texto (25)		
	Telefone	Texto (20)		
	Telemovel	Texto (20)		
	Fax	Texto (20)		
	Email	Texto (75)		
	Web	Texto (75)		
	NIF	Texto (9)		

Tabela 3.2: Tabela de dados relativos a clientes

Compras:

	Campo	Tipo	Descrição	Obri.
CP	Compra	Inteiro	Código da compra.	*
CE	Fornecedor	Texto (15)	Código do fornecedor.	*
	Documento	Texto (25)	Número do documento do fornecedor.	
	Data	Data		*
	Valor	Decimal(10,2)	Valor total da compra.	*
	Valor_Pago	Decimal(10,2)	Valor já pago ao fornecedor.	*
	Data_Documento	Data		
	Data_Vencimento	Data	Data final para o pagamento ao fornecedor.	
	Fechada	Verd./Falso	Indicação que o documento está fechado.	*
	Observacoes	Texto (4000)		

Tabela 3.3: Tabela de dados relativos a compras

Compras_Linhas:

	Campo	Tipo	Descrição	Obri.
CP	Compra_Linha	Identificador Único	Código da linha.	*
CE	Compra	Inteiro	Código da compra.	*
CE	Artigo	Texto (15)	Código do artigo.	*
	Nome	Texto (50)	Nome do artigo.	*
	Quantidade	Decimal(7,2)		*
	Quantidade_Disponivel	Decimal(7,2)	Quantidade ainda disponível para ser afeta a uma obra.	*
	Preco_unitario	Decimal(10,2)		*
	Desconto	Inteiro	Taxa de desconto aplicada pelo fornecedor.	*
	IVA	Inteiro	Taxa de IVA.	*
	Unidade	Texto (5)		*
	Quantidade_Devolvida	Decimal(7,2)	Quantidade de artigo que foi devolvida ao fornecedor.	*

Tabela 3.4: Tabela de dados relativos às linhas (artigos) de compras

Custos_Indirectos:

	Campo	Tipo	Descrição	Obri.
CP	Custo_Indirecto	Texto(15)	Código do custo indireto.	*
	Nome	Texto (50)		*

Tabela 3.5: Tabela de dados relativos a custos indiretos

Facturas:

	Campo	Tipo	Descrição	Obri.
CP	Factura	Inteiro	Código da fatura.	*
CE	Obra	Inteiro	Código da obra.	*
	Documento	Texto (25)	Número do documento do programa de faturação.	
	Data	Data		*
	Valor	Decimal(10,2)	Valor total da fatura.	*
	Valor_Recebido	Decimal(10,2)	Valor já pago pelo cliente.	*
	IVA	Decimal(10,2)	Valor do IVA do documento.	*
	Data_Vencimento	Data	Data limite para a liquidação do documento por parte do cliente.	*
	Fechada	Verd./Falso	Indicação que o documento está fechado.	
	Data_Documento	Data	Data em que o documento foi criado no *software* de faturação.	*

Tabela 3.6: Tabela de dados relativos a faturas a clientes

Fornecedores:

	Campo	Tipo	Descrição	Obri.
CP	Fornecedor	Texto (15)	Código do Fornecedor.	*
	Nome	Texto (50)		*
	Contacto	Texto (50)		
	Morada	Texto (100)		
	CodigoPostal	Texto (15)		
	Localidade	Texto (30)		
	Pais	Texto (25)		
	Telefone	Texto (20)		
	Telemovel	Texto (20)		
	Fax	Texto (20)		
	Email	Texto (75)		
	Web	Texto (75)		
	NIF	Texto (9)		

Tabela 3.7: Tabela de dados relativo a fornecedores

Fornecedores_Adiantamentos:

	Campo	Tipo	Descrição	Obri.
CP	Adiantamento	Inteiro	Código do adiantamento.	*
CE	Fornecedor	Texto (15)	Código do Fornecedor.	*
	Documento	Texto (100)	Número do documento criado no *software* de faturação.	*
	Data_Documento	Data	Data em que o documento foi criado no *software* de faturação.	*
	Observacoes	Texto (4000)		
	Valor	Decimal(10,2)	Valor total do adiantamento.	*
	Valor_Disponivel	Decimal(10,2)	Valor ainda disponível do adiantamento feito ao fornecedor.	*
	Data	Data (25)	Data em que foi criado o adiantamento no sistema.	*

Tabela 3.8: Tabela de dados relativos a adiantamentos efetuados a fornecedores

Funcionarios:

	Campo	Tipo	Descrição	Obri.
CP	Funcionario	Texto (15)	Código do funcionário.	*
	Nome	Texto (50)		*
	Morada	Texto (100)		
	CodigoPostal	Texto (15)		
	Localidade	Texto (30)		
	Pais	Texto (25)		
	Telefone	Texto (20)		
	Telemovel	Texto (20)		
	Email	Texto (75)		
	Web	Texto (75)		

Tabela 3.9: Tabela de dados relativos a funcionários

Funcionarios_Custo_Hora:

	Campo	Tipo	Descrição	Obri.
CP/CE	Funcionario	Texto (15)	Código do funcionário.	*
CP	Custo_Hora	Decimal(5,2)	Valor por hora de trabalho.	*
CP/CE	Tipo	Texto (25)	Tipo de hora.	*

Tabela 3.10: Tabela de dados relativos a custo de mão-de-obra de cada funcionário segundo o tipo de hora

Obras:

	Campo	Tipo	Descrição	Obri.
CP	Obra	Texto (15)	Código da obra.	*
	Nome	Texto (30)		*

CE	Cliente	Texto (15)		*
	Processo	Texto (30)		
	Data	Data	Criação da obra no sistema.	*
	Fechada	Verd./Falso	Indicação se a obra está ou não fechada.	*
	Observacoes	Texto (4000)		
	Contacto	Texto (50)	Nome da pessoa responsável por parte do cliente perante a obra.	
	Morada	Texto (100)		
	CodigoPostal	Texto (15)		
	Localidade	Texto (30)		
	Pais	Texto (25)		
	Telefone	Texto (20)		
	Telemovel	Texto (20)		

Tabela 3.11: Tabela de dados relativos a obras

Obras_ Adiantamentos:

	Campo	Tipo	Descrição	Obri.
CP	Adiantamento	Inteiro	Código do Adiantamento.	*
CE	Obra	Inteiro	Código da obra.	*
	Documento	Texto (100)	Número do documento criado no *software* de faturação.	*
	Data_Documento	Data	Data em que o documento foi criado no *software* de faturação.	*
	Observacoes	Texto (4000)		
	Valor	Decimal(10,2)	Valor total do adiantamento.	*
	Valor_Disponivel	Decimal(10,2)	Valor ainda disponível do adiantamento feito pelo cliente.	*
	Data	Data	Data em que foi criado o adiantamento no sistema.	*

Tabela 3.12: Tabela de dados relativos a adiantamentos dos clientes para liquidação da obra

Obras_Seccoes:

	Campo	Tipo	Descrição	Obri.
CP/CE	Obra	Texto (15)	Código da obra.	*
CP/CE	Seccao	Texto(5)	Código da secção.	*
	Observacoes	Texto (4000)		

Tabela 3.13: Tabela de dados relativos a secções de obras

Obras_Seccoes_Compras_Linhas:

	Campo	Tipo	Descrição	Obri.
CP	Obra_Seccao_Compra_Linha	Identificador Único	Código da linha.	*

CE	Compra_Linha	Identificador Único	Código da linha do documento de compra.	*
CE	Obra	Inteiro	Código da obra.	*
CE	Seccao	Texto(5)	Código da secção.	*
	Quantidade	Decimal(7,2)	Quantidade de artigo afeta a esta secção da obra.	*
	Data	Data	Data em que os artigos foram afetos à obra.	*

Tabela 3.14: Tabela de artigos de uma compra afetos a secções de obras

Obras_Seccoes_Custos_Indirectos:

	Campo	Tipo	Descrição	Obri.
CP	Obra_Seccao_Custo_Indirecto	Identificador Único	Código da linha.	*
CE	Obra	Inteiro	Código da obra.	*
CE	Seccao	Texto(5)	Código da secção.	*
CE	Custo_Indirecto	Texto(5)	Código do custo indireto.	*
	Data	Data	Data em que os custos foram afetos à obra.	*
	Valor	Decimal(6,2)		*
	Documento	Texto(25)	Número do documento do fornecedor.	
	Data_Documento	Data	Data do documento do fornecedor.	

Tabela 3.15: Tabela de dados relativos a custos indiretos imputados a secções de obras

Obras_Seccoes_Funcionarios:

	Campo	Tipo	Descrição	Obri.
CP	Obra_Seccao_Funcionario	Identificador Único	Código da linha.	*
CE	Obra	Inteiro	Código da obra.	*
CE	Seccao	Texto(5)	Código da secção.	*
CE	Funcionario	Texto(15)	Código do funcionário.	*
	Data	Data	Data de lançamento da hora.	*
	Horas	Decimal(5,1)	Horas a registar.	*
	CustoHora	Decimal(10,2)	Custo por hora de trabalho.	*
CE	Tipo	Texto	Tipo de hora.	*

Tabela 3.16: Tabela de dados relativos a horas e custo de horas de funcionários por secções de obras

Obras_Seccoes_Orcamentos:

	Campo	Tipo	Descrição	Obri.
CP	Orcamento	Identificador Único	Código do orçamento.	*

CE	Obra	Inteiro	Código da obra.	*
CE	Seccao	Texto(5)	Código da secção.	*
	Observacoes	Texto(4000)		
	Data	Data	Data de lançamento da hora.	*
	Valor	Decimal(10,2)	Valor do orçamento.	*

Tabela 3.17: Tabela que contém os dados relativos aos orçamentos das secções das obras

Pagamento:

	Campo	Tipo	Descrição	Obri.
CP	Pagamento	Inteiro	Código do pagamento.	*
CE	Fornecedor	Texto(15)	Código do fornecedor.	*
	Observacoes	Texto(4000)		
	Data	Data	Data de lançamento do documento.	*
	Valor	Decimal(10,2)	Valor do orçamento.	*
	Documento	Texto(25)	Número do documento do *software* de faturação.	
	Adiantamento	Inteiro	Código do adiantamento a ser utilizado no pagamento.	
	Adiantamento_Valor	Decimal(10,2)	Valor utilizado do adiantamento.	

Tabela 3.18: Tabela que contém os dados relativos aos pagamentos efetuados aos fornecedores

Pagamentos_Compras:

	Campo	Tipo	Descrição	Obri.
CP	Compra_Pagamento	Identificador Único	Código da linha.	*
CE	Compra	Inteiro	Código da compra.	*
CE	Pagamento	Inteiro	Código do Pagamento.	*
	Valor	Decimal(10,2)	Valor pago da compra.	*

Tabela 3.19: Tabela de dados relativos a compras pagas por pagamento

Recibos:

	Campo	Tipo	Descrição	Obri.
CP	Recibo	Inteiro	Código do recibo.	*
CE	Obra	Inteiro	Código da Obra.	*
	Observacoes	Texto(4000)		
	Data	Data	Data de lançamento no sistema.	*
	Valor	Decimal(10,2)	Valor do orçamento.	*
	Documento	Texto(25)	Número do documento do *software* de faturação.	

CE	Adiantamento	Inteiro	Código do adiantamento a ser utilizado.	
	Adiantamento_Valor	Decimal(10,2)	Valor utilizado do adiantamento.	
	Data_Documento	Data	Data valor do lançamento.	

Tabela 3.20: Tabela de dados relativos a recibos de pagamentos efetuados por clientes

Recibos_Facturas:

	Campo	Tipo	Descrição	Obri.
CP	Factura_Recibo	Identificador Único	Código da linha.	*
CE	Recibo	Inteiro	Código do recibo.	*
CE	Factura	Inteiro	Código da fatura.	*
	Valor	Decimal(10,2)	Valor pago da fatura.	*

Tabela 3.21: Tabela de dados relativos às faturas pagas por recibo

Seccoes:

	Campo	Tipo	Descrição	Obri.
CP	Seccao	Texto(5)	Código da secção.	*
	Descricao	Texto (50)		*

Tabela 3.22: Tabela de dados relativos às secções

Tipos_Hora:

	Campo	Tipo	Descrição	Obri.
CP	Tipo_Hora	Texto(25)	Código do tipo de hora.	*

Tabela 3.23: Tabela de dados relativos a tipos de hora

Unidades:

	Campo	Tipo	Descrição	Obri.
CP	Unidade	Texto(5)	Código da unidade.	*

Tabela 3.24: Tabela de dados relativos a unidades de artigos

3.3. Conclusões

Neste capítulo foi apresentado a análise e descrição do sistema e a modelação do problema. Foram descritas as entidades presentes no sistema, o modelo de dados adotado e o dicionário de dados para este caso de estudo. No capítulo seguinte são apresentadas as funcionalidades desenvolvidas que constituem o *Software* e a sua estrutura.

Capítulo 4 - **Aplicação**

4.1. Objetivos

O objetivo primordial da aplicação é controlar os custos inerentes a uma obra e às várias secções em que esta se divide, nomeadamente controlar compras e pagamentos a fornecedores, e de clientes.

4.2. Funcionalidades da aplicação

- **Gestão de Obras**
    - Ficha da obra
    - Secções
        - Orçamentos
        - Custos Indiretos
        - Mão-de-obra
    - Faturas
    - Adiantamentos
    - Recibos
- **Gestão de Compras**
    - Compra Artigos
        - Distribuição de artigos pelas Obras
    - Pagamentos
    - Adiantamentos
- **Gestão de Entidades:**
    - Clientes
    - Fornecedores
    - Funcionários
- **Gestão de Artigos**
- **Gestão de Custos Indiretos**

Na Figura 4.1 estão representadas as várias ações que um utilizador poderá realizar na aplicação.

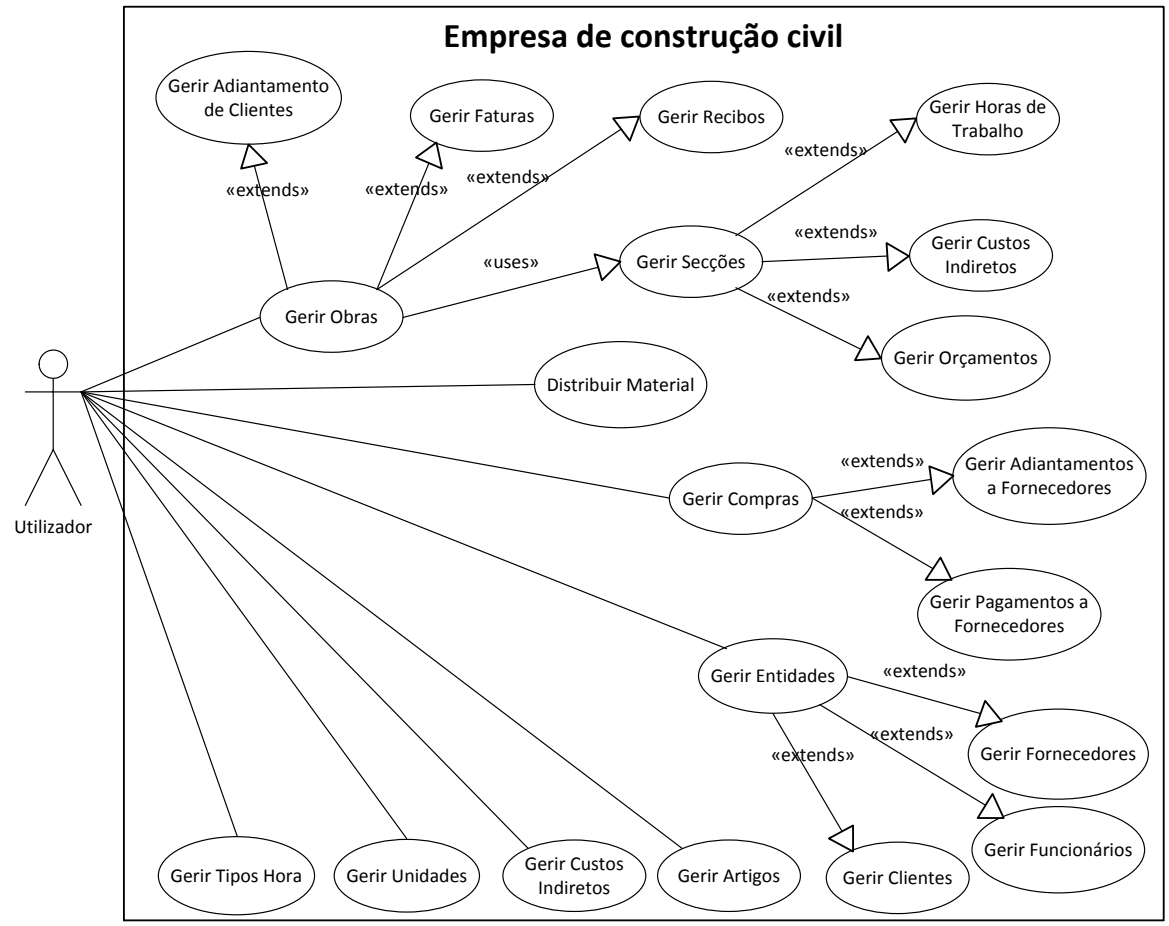


Figura 4.1: Diagrama de Caso de Uso – Funcionalidades da aplicação

4.3. Organização da Aplicação

Para o desenvolvimento desta aplicação optou-se por utilizar uma lógica composta pelas seguintes 4 camadas (N-Tier, 2012):

- **Camada de Objetos (DTO - *Data Transfer Objects*):**

  É nesta camada que estão definidos os objetos que permitem a passagem de informação entre as várias camadas.

- **Camada de Acesso aos Dados (DAL - *Data Access Layer*):**

  É responsável pela interação com a base de dados.

- **Camada de Lógica de Negócio (BLL - *Business Logic Layer*):**

  É responsável por validar os dados segundo a lógica de negócio.

- **Camada de Apresentação (PL – *Presentation Layer*):**

  É responsável pela interação com o utilizador.

Foi ainda criada uma quinta camada, Controlo de Listas (LC – ***List Control***), que permite controlar quais os dados que serão apresentados no formulário de listagem de dados.

O uso de camadas permite maior produtividade, melhor desempenho, e facilita a manutenção. Potencia-se o reaproveitamento de código e uma maior escalabilidade do sistema. O uso de camadas permite ainda ter uma aplicação em plataforma Windows e outra em plataforma web, com apenas a criação de mais uma camada e fazendo uso de todas as outras.

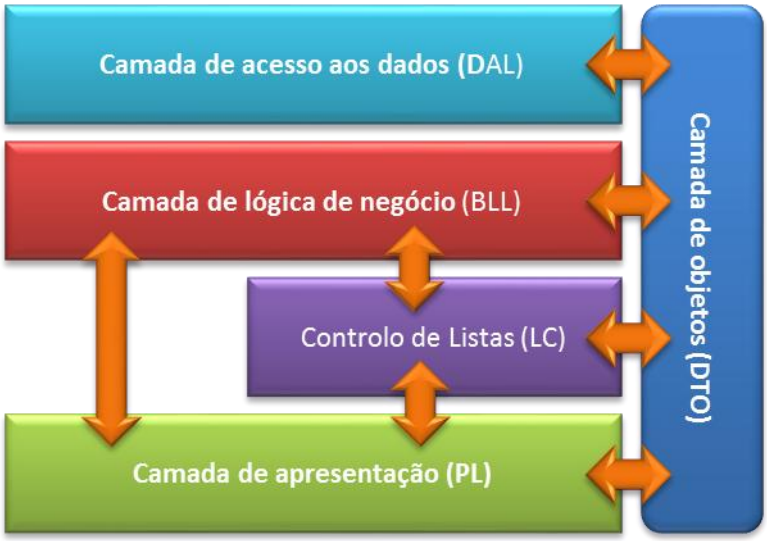


Figura 4.2: Esquema das camadas da aplicação

Na Figura 4.2 é possível ver as várias camadas e a relação entre elas. A Camada de objetos é utilizada por todas as camadas da aplicação. A Camada de acesso aos dados apenas é utilizada pela Camada de lógica de negócio, não sendo por isso possível à Camada de apresentação enviar ou receber informação sem que esta tenha sido validada pela Camada de negócio. A subcamada Controlo de listas faz a ligação entre o formulário de listagem, que se encontra na Camada de aplicação, e a Camada de lógica de negócio.

Na Figura 4.3 está representado o diagrama de classes que descreve a relação entre as diferentes entidades.

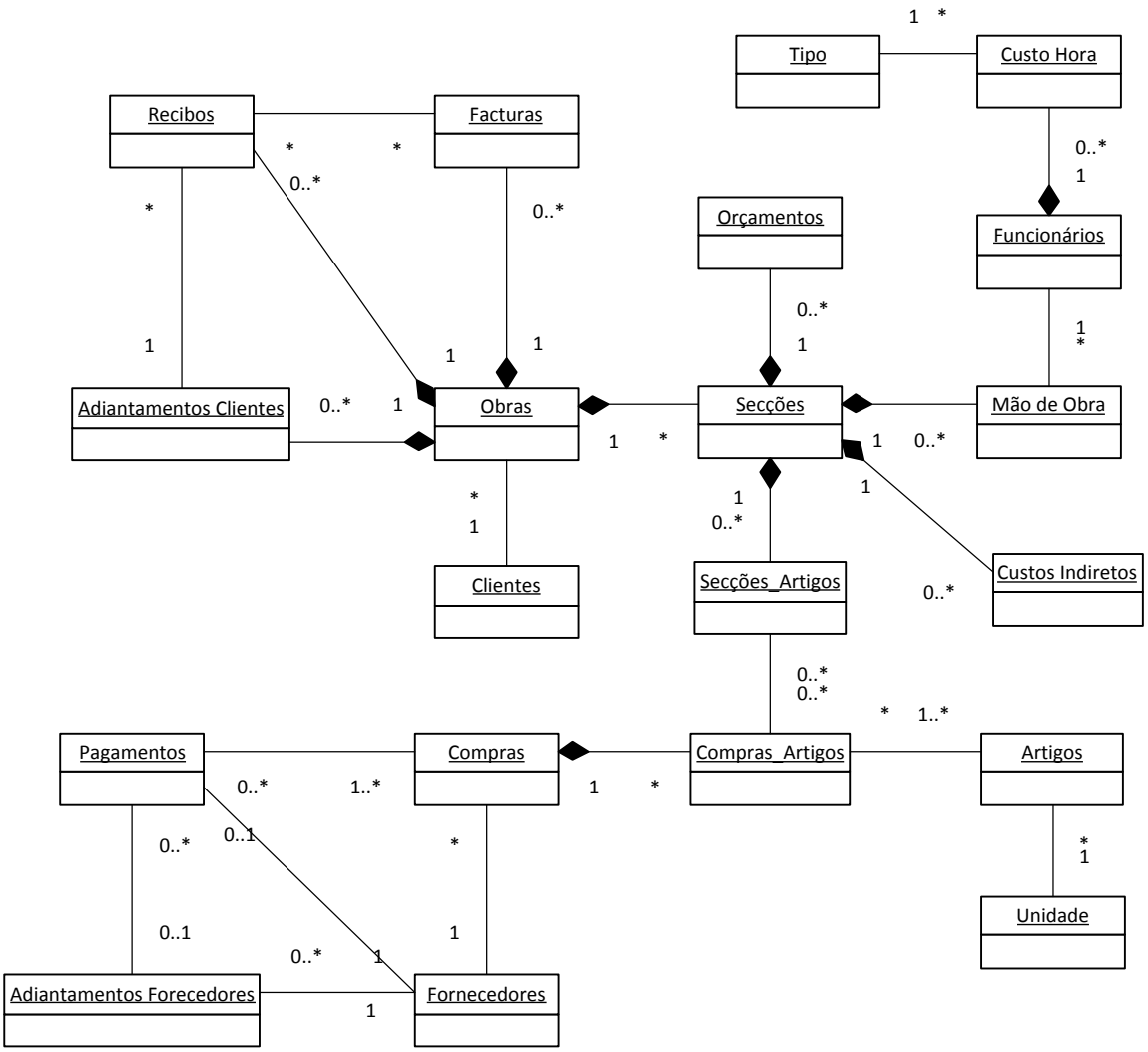


Figura 4.3: Diagrama de Classes

4.4. Ferramentas e tecnologias

Esta aplicação foi idealizada para ser utilizada em plataforma Windows, sendo esta a utilizada pela empresa. Para o desenvolvimento deste projeto foram utilizadas ferramentas de desenvolvimento Microsoft e componentes da Infragistics[6]:

- **Microsoft SQL Server 2008 Express** (Sql Server, 2012): este gestor de base de dados é de uso gratuito mas herda as principais características do versão Standard do SQL Server 2008 com algumas limitações – o tamanho de uma base de dados não pode ser superior a 10GB e só utiliza 1 processador e 1 GB de memória RAM mesmo que o computador possua mais recursos. Dada a dimensão da aplicação e os dados por ela geridos, estas limitações não afetam o bom funcionamento da mesma. Mesmo que no futuro seja necessário mais recursos é sempre possível migrar para uma versão superior sem que seja necessário realizar qualquer alteração à aplicação.
- **.NET *Framework* 4.0** (.NET Framework 4, 2011): a .NET *Framework* foi desenvolvida pela Microsoft para uniformizar a programação de aplicações para vários tipos de dispositivos, plataformas e linguagens de programação. Esta possui dois componentes principais, o ***Common Language Runtime*** (CLR), responsável pela execução do código e a **bibliotecas de classes**. Neste projeto foi utilizado a versão 4.0 por ser a mais recentemente disponibilizada pela Microsoft.
- **Visual Basic 2010 Express** (Express, 2012): este *IDE* (*Integrated Development Environment*) é de uso gratuito e foi desenvolvido pela Microsoft para fornecer uma forma fácil e rápida para a criação de aplicações Windows utilizando a linguagem de programação Visual Basic. Não sendo um requisito específico a linguagem a utilizar optou-se por utilizar o Visual Basic dada a experiência acumulada com a realização de projetos anteriores.
- **SQL Reporting Services** (Service, 2011): plataforma de desenvolvimento de relatórios, desenvolvido pela Microsoft e que é parte integrante do SQL Server. Os relatórios podem ser utilizados em modo servidor, podendo o utilizador interagir com este a partir de uma página Web, ou em modo local, permitindo a integração em aplicações. O Visual Basic 2010 Express não permite a criação destes relatórios sendo necessário recorrer ao Report Builder para a criação dos mesmos.
- **Report Builder** (Builder, 2012): aplicação criada pela Microsoft para criação de relatórios.
- **Windows Forms Controls** (Controls, 2011): pacote de componentes desenvolvidos pela Infragistics. Estes foram desenvolvidos para serem utilizados na plataforma .NET. Deste pacote foram sobretudo utilizados neste projeto o *UltraGrid* (para apresentação de dados em forma de tabela), o *UltraWinToolbars* (para criação do menu), e o *UltraWinTree* (para criação das opções no formulário de gestão das obras).

[6] http://www.infragistics.com acedido a 02 de janeiro de 2012

No mercado existem várias opções para o desenvolvimento e utilização de *software* desde *IDEs*, sistemas de gestão de base de dados, ferramentas de relatórios e linguagens de programação. As ferramentas Microsoft fornecem um ambiente de desenvolvimento rápido (*RAD - Rapid Application Development* (Silva & Videira, 2001)) e abrangem todas as etapas do desenvolvimento e utilização dos *software*s. Sendo estas criadas pela mesma empresa e estarem pensadas para funcionar em conjunto, como um todo, não é necessário ao programador despender muitos recursos para interligar todas as partes. A principal desvantagem das ferramentas Microsoft é só poderem ser utilizadas em sistemas operativos Windows, o que para este projeto não é relevante visto este ser destinado a ser utilizado em computadores com esse sistema operativo. É de realçar que das ferramentas apresentadas anteriormente só os componentes da Infragistics não são de utilização grátis, e estes só foram utilizados porque permitem criar interfaces mais "amigáveis" para o utilizador e assim tornar a utilização da aplicação mais ágil e interativa.

Capítulo 5 - *Framework*

A uniformização de processos de desenvolvimento leva a uma maior eficácia de desenvolvimento e permite uma maior rentabilidade de tempo no desenvolvimento, na manutenção e na correção de erros. Na manipulação de dados numa base de dados relacional existem quatro operações básicas designadas por CRUD (*Create*, *Read*, *Update* e *Delete*) (Silva & Videira, 2001). Estas operações, independentemente do contexto, são quase sempre realizadas da mesma forma, o que possibilita a uniformização destes processos. Partindo deste prossuposto definiu-se a estrutura base da aplicação que consiste na criação, para cada tabela, do seguinte:

- Uma classe na Camada de Objectos, de modo a permitir o envio de dados entre as várias camadas;
- Uma classe na Camada de Acesso aos Dados para manipular os dados entre a aplicação e a base de dados, em que são implementados 5 métodos: um para inserir, outro para alterar, outro para eliminar, um para carregar um elemento da tabela com base na chave e outro para carregar todos os registos da tabela;
- Uma classe na Camada de Lógica de Negócio que implementa os cinco métodos referidos no ponto anterior, mais o método que valida os dados segundo as regras de negócio;
- Formulário para permitir a manipulação dos dados;
- Formulário para listar os dados da tabela.

Depois de definida a estrutura da aplicação optou-se por criar uma *framework* própria que servisse de suporte não só a este caso de estudo mas a todos os projetos semelhantes a realizar no futuro. Assim, foram criados:

- *Templates* de projetos, um por cada camada da aplicação;
- Um *template* da janela de manipulação de dados;
- Dois controlos (a *textbox* MaskBox e a *grid* DJC_Grid);
- Uma aplicação que, com base no modelo relacional, permite criar os vários objetos e respectivo código para cada uma das camadas, incluindo os *script*s para a criação de *Stored Procedures* que permitem as operações de CRUD.

De seguida, são apresentados os vários elementos que constituem a *framework*, qual o papel de cada um e como estes se interligam entre si. No fim é apresentado um caso prático tendo por exemplo a tabela de Artigos do caso de uso deste projeto.

5.1. Templates

5.1.1. Projetos

Cada *template* de projecto gera uma camada da aplicação. Na Figura 5.1 está representada a ligação entre os *templates* e os projetos.

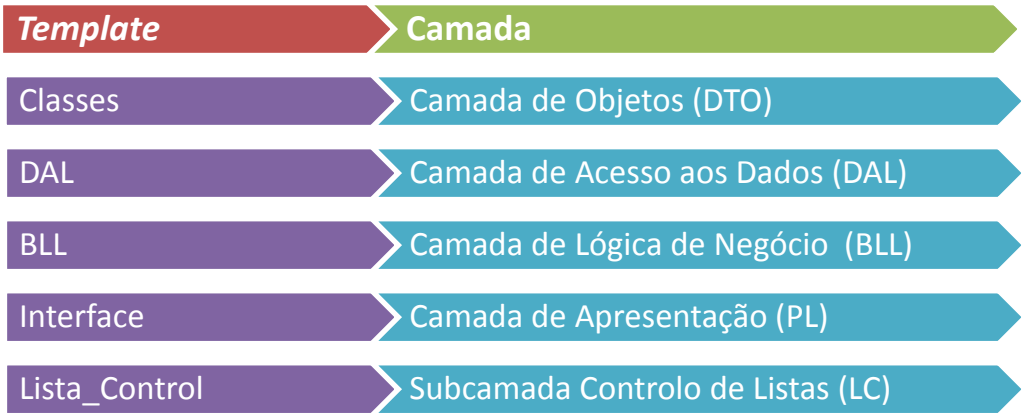


Figura 5.1: Geração de projeto com base em *template*.

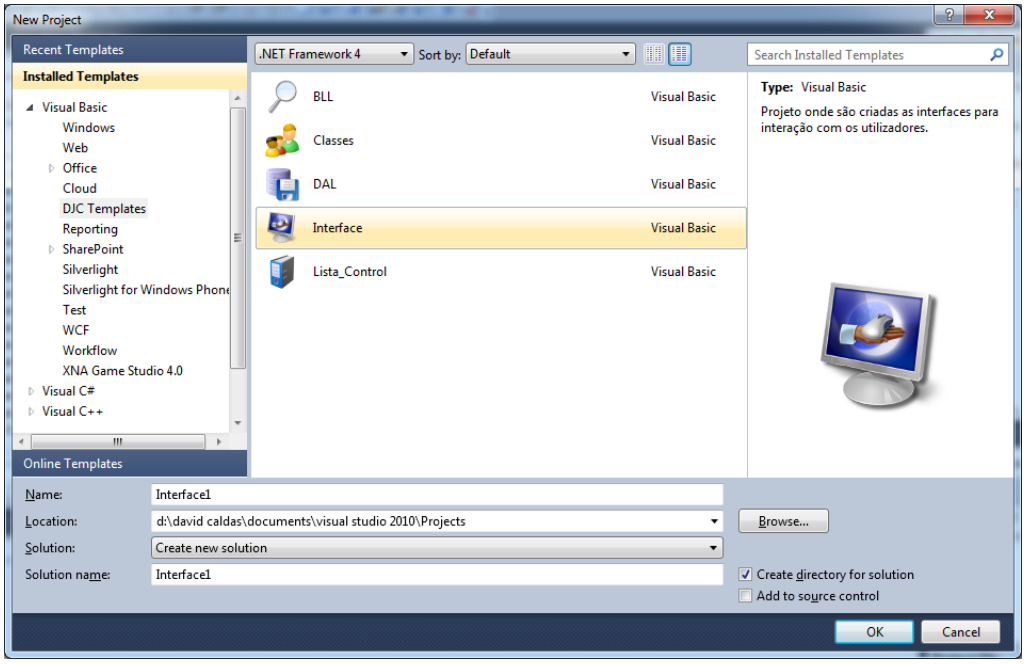


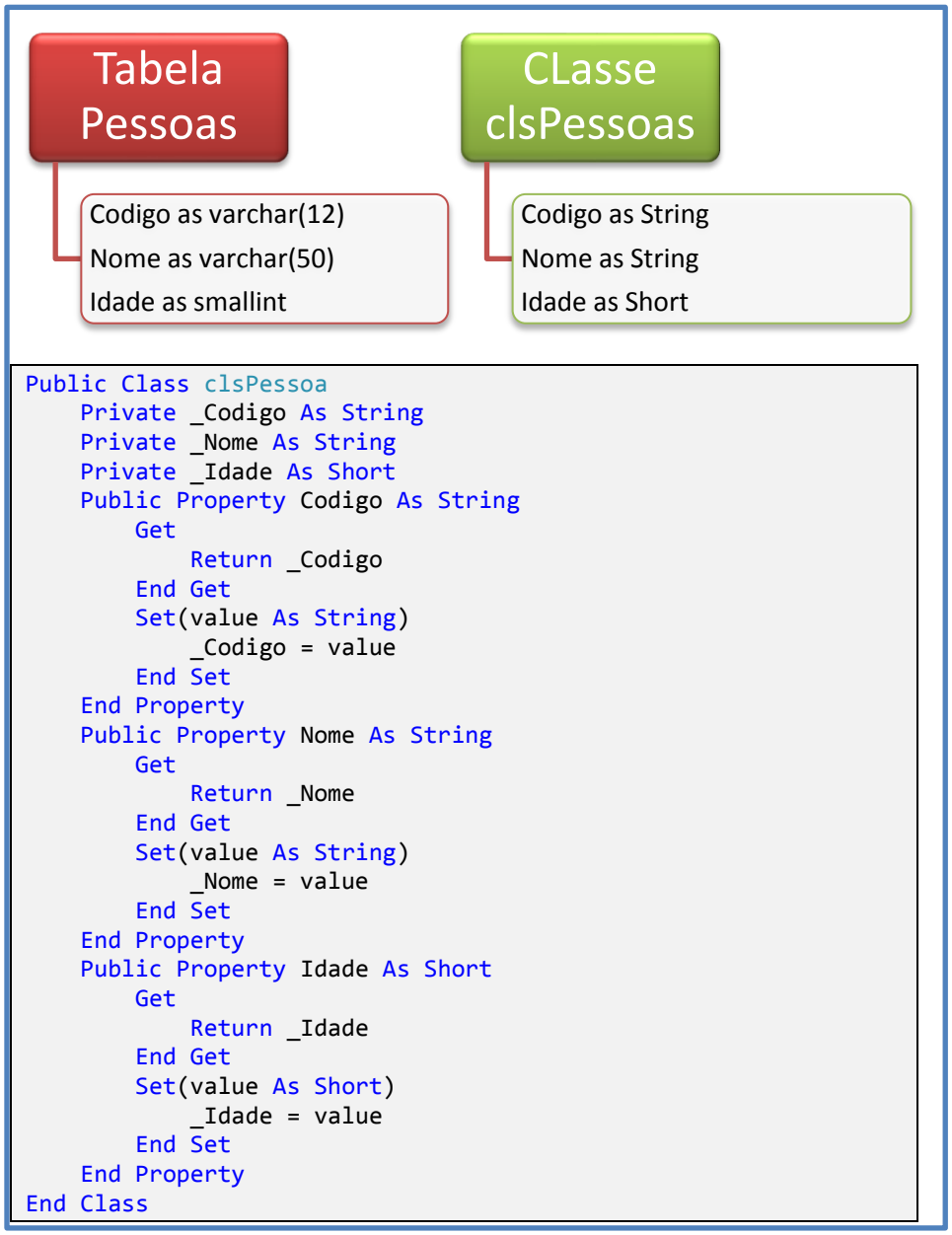
Figura 5.2: Janela de criação de um novo projeto.

Como se pode ver na Figura 5.2 foram criados quatro *templates* (Classes, DAL, BLL e Interface), cada um correspondendo a uma camada da aplicação (ver secção 4.3) e um *template* com designação Lista_Control que corresponde à subcamada que permite controlar a listagem de dados.

5.1.1.1. Classes

Na camada Classes serão criadas as classes utilizadas para a passagem de informação entre as várias camadas. Essas classes serão geradas a partir da aplicação criada na *framework*. Cada uma das tabelas da base de dados origina uma classe nesta camada, em que os atributos das tabelas são as propriedades das classes. A aplicação tem em consideração o nome e o tipo de dados do atributo. Conforme se pode ver na Figura 5.3, a tabela exemplificativa **Pessoas** com o

atributo **Nome** do tipo ***Varchar*** origina uma classe com o nome **clsPessoas** com a propriedade **Nome** do tipo ***String***.




```
Public Class clsPessoa
    Private _Codigo As String
    Private _Nome As String
    Private _Idade As Short
    Public Property Codigo As String
        Get
            Return _Codigo
        End Get
        Set(value As String)
            _Codigo = value
        End Set
    End Property
    Public Property Nome As String
        Get
            Return _Nome
        End Get
        Set(value As String)
            _Nome = value
        End Set
    End Property
    Public Property Idade As Short
        Get
            Return _Idade
        End Get
        Set(value As Short)
            _Idade = value
        End Set
    End Property
End Class
```

Figura 5.3: Exemplo da geração de uma classe na Camada de Objectos com base na estrutura de uma tabela.

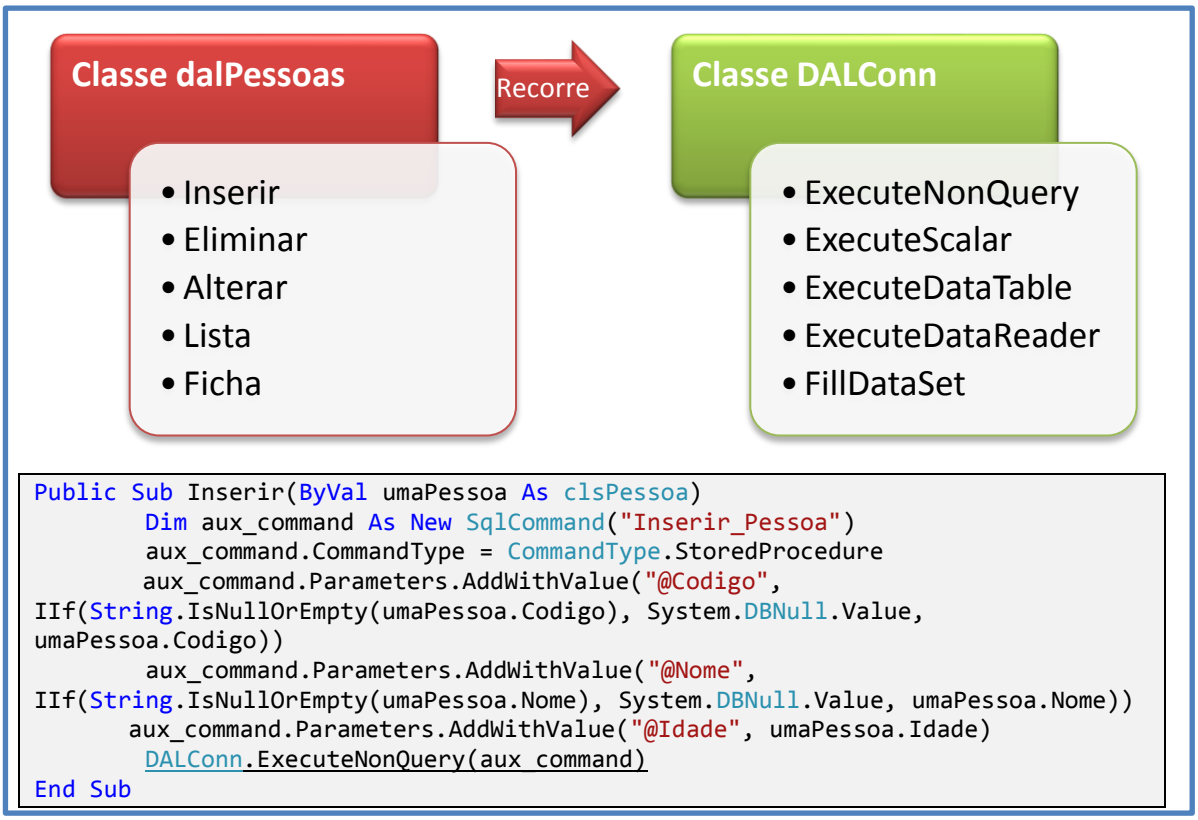
5.1.1.2. DAL

Aqui são criadas todas as classes responsáveis pela comunicação "de" e "para" a base de dados. O *template* contém uma classe *DALConn* responsável por enviar e receber dados da base de dados. Esta classe contém métodos de controlo da abertura e fecho da ligação e transações, e execução de comandos SQL (Structured *Query Language*). Sempre que uma classe desta camada necessita de comunicar com a base de dados, tem apenas de compor o objeto comando e enviá-lo para a *DALConn* invocando a função correta segundo o tipo de operação que se pretende realizar. Esta possui os seguintes métodos, que podem ser invocados para a manipulação de dados:

- **ExecuteNonQuery**: utilizado quando se pretende executar um comando de inserção, atualização ou eliminação. Este devolve um inteiro com o total de registos afetados;

- **ExecuteScalar**: utilizado quando se pretende devolver um único valor da base de dados;
- **ExecuteDataTable**: devolve a consulta à base de dados em forma de *DataTable*;
- **ExecuteDataReader**: devolve a consulta à base de dados em forma de *DataReader*;
- **FillDataSet**: carrega uma tabela de um *DataSet*. O *DataSet* e o nome da tabela são parâmetros da função.

Todos os métodos referidos têm como parâmetro de entrada o comando SQL a ser executado. À semelhança do que sucede na Camada de Classes também aqui as classes são geradas a partir da aplicação, em que cada tabela origina uma classe nesta camada. A aplicação implementa em cada classe os métodos necessários para a realização das operações CRUD. Na Figura 5.4 está exemplificada essa implementação e pode ainda ser visualizado a relação desta camada com a Camada de Classes. Todos os métodos, com exceção do Eliminar, utilizam a classe implementada na Camada de Classes, seja para enviar ou para receber informação.



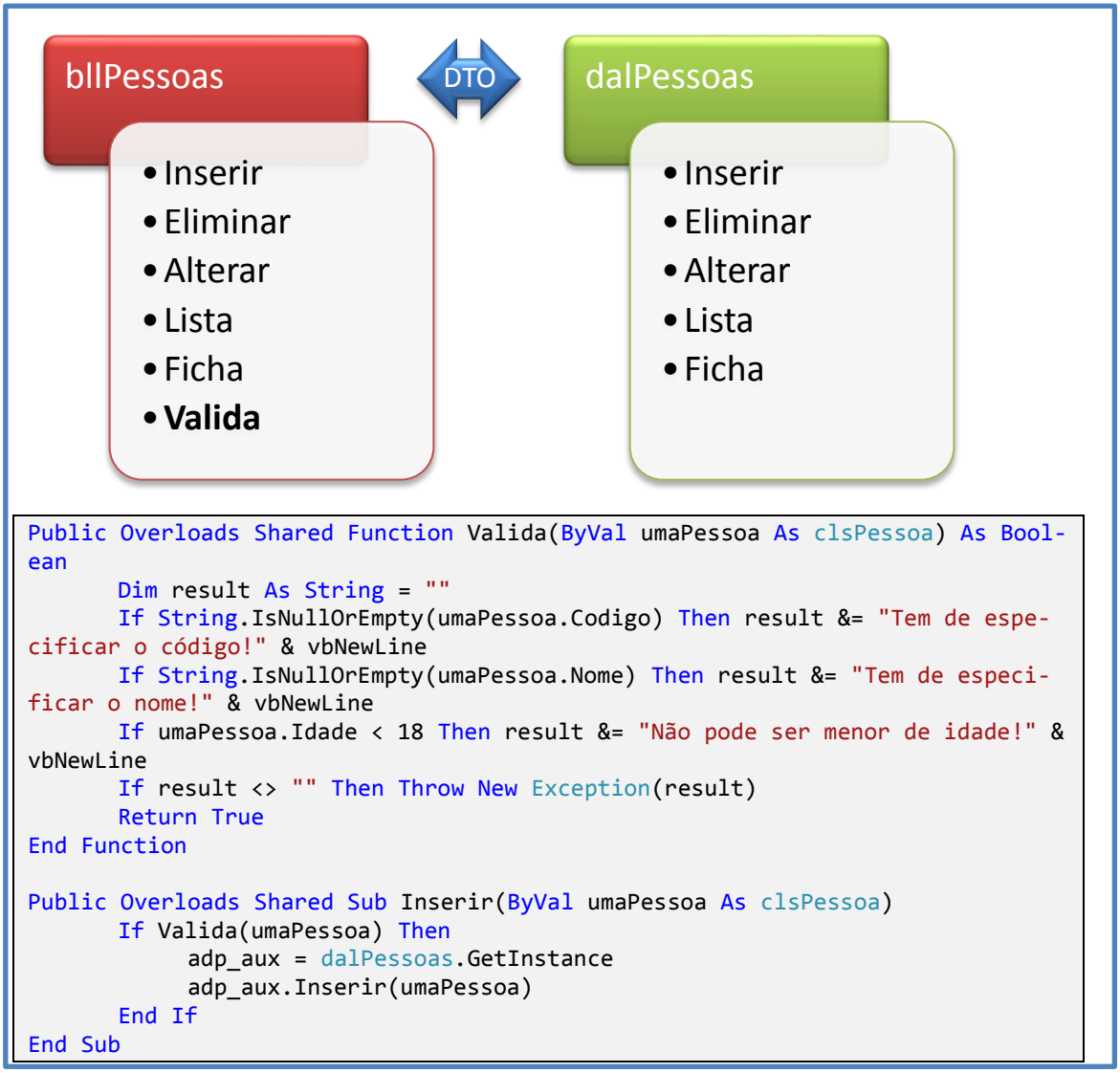

```
Public Sub Inserir(ByVal umaPessoa As clsPessoa)
        Dim aux_command As New SqlCommand("Inserir_Pessoa")
        aux_command.CommandType = CommandType.StoredProcedure
        aux_command.Parameters.AddWithValue("@Codigo",
IIf(String.IsNullOrEmpty(umaPessoa.Codigo), System.DBNull.Value,
umaPessoa.Codigo))
        aux_command.Parameters.AddWithValue("@Nome",
IIf(String.IsNullOrEmpty(umaPessoa.Nome), System.DBNull.Value, umaPessoa.Nome))
       aux_command.Parameters.AddWithValue("@Idade", umaPessoa.Idade)
        DALConn.ExecuteNonQuery(aux_command)
End Sub
```

Figura 5.4: Exemplo da geração de uma classe na Camada de Acesso aos Dados com base na estrutura de uma tabela.

5.1.1.3. BLL

Ao criar um projeto deste tipo, este contém já uma classe bllBase que permite à Camada de Apresentação passar os parâmetros de ligação à Camada de Acesso aos Dados. Nesta camada serão implementadas as regras de negócio. À semelhança do que sucede na Camada de Classes e da Camada de Acesso aos Dados também aqui as classes são geradas a partir da aplicação. A aplicação cria as funções CRUD que, por sua vez, invocam uma função **Valida** que permite validar os dados segundo a regra de negócio. Inicialmente esta função não tem qualquer validação, porque a aplicação não é capaz de, com base no esquema de base de dados, implementar as regras de negócio. Assim, terá que ser o programador a implementá-las. É de referir que, caso seja necessário, a implementação de regras mais elaboradas (exemplo: sempre que é criado uma

compra é enviado um email ao fornecedor a informar que esta se encontra validada), é necessário que o programador altere a função de inserção para implementar essa funcionalidade. A aplicação implementa em cada classe os métodos necessários para a realização das operações CRUD, invocando os métodos das classes implementadas na Camada de Acesso aos Dados. Na Figura 5.5 está exemplificada essa implementação.




```
Public Overloads Shared Function Valida(ByVal umaPessoa As clsPessoa) As Bool-
ean
       Dim result As String = ""
       If String.IsNullOrEmpty(umaPessoa.Codigo) Then result &= "Tem de espe-
cificar o código!" & vbNewLine
       If String.IsNullOrEmpty(umaPessoa.Nome) Then result &= "Tem de especi-
ficar o nome!" & vbNewLine
       If umaPessoa.Idade < 18 Then result &= "Não pode ser menor de idade!" &
vbNewLine
       If result <> "" Then Throw New Exception(result)
       Return True
End Function

Public Overloads Shared Sub Inserir(ByVal umaPessoa As clsPessoa)
       If Valida(umaPessoa) Then
            adp_aux = dalPessoas.GetInstance
            adp_aux.Inserir(umaPessoa)
       End If
End Sub
```

Figura 5.5: Exemplo da geração de uma classe na Camada de Lógica de Negócio com base na estrutura de uma tabela.

5.1.1.4. Lista_Control

Quando se cria um projeto deste tipo, este contém já uma classe base com os métodos que as classes filhas necessitam de implementar para que possam ser utilizadas pela Camada de Apresentação. Em todas as classes filhas, que herdam as características da classe base, é necessário implementar os seguintes métodos:

- **Carregar_Parametros**: chamado no construtor destas classes, é nele que são especificadas as configurações de cada listagem, nomeadamente:
    - **frm_Manutencao**: variável que guarda o nome do formulário utilizado para manipular os dados;
    - **Lista_Codigos**: variável que guarda a chave primária ou composta;

- **Listagem_Titulo**: variável que guarda o nome da listagem que aparece como título do formulário de listagem de dados;
- **Requer_Data**: variável que indica se a listagem é ou não filtrada por dados. Esta situação acontece quase sempre na listagem de documentos. Visto estes crescerem infinitamente no tempo não faria sentido abrir uma listagem com todos os documentos criados.
- **Escolher_Colunas**: variável que indica se utilizador pode ou não escolher que colunas deseja visualizar.
- **Ajustar_Colunas**: variável que indica se o utilizador pode ou não ajustar as colunas automaticamente.

- **Carregar_Colunas**: são aqui configuradas as colunas segundo as regras da *UltraGrid*. As configurações mais comuns são:
  - **Texto**: Nome que aparece no cabeçalho da coluna;
  - **Editável**: Possibilidade de alterar os dados da coluna;
  - **Tamanho**: O tamanho da coluna;
  - **Visível**: Possibilidade de alterar visibilidade da coluna para o utilizador;
  - **Posição**: Posição da listagem em que a coluna aparece;
  - **Formatação**: Formatação dos dados no caso de valores numéricos, monetários ou datas.

Existem ainda dois métodos que, dependendo do contexto, podem ou não ser reescritos pelas classes filhas. Estes são:

- **Lista**: chamado pelo formulário de listagem para carregar os dados a apresentar ao utilizador. Este método possui duas declarações, uma sem parâmetros e outra que recebe como parâmetros a data de início e a data de fim. Cada classe filha tem de reescrever o método que se lhe aplica especificando o código necessário para obter os dados. Ao escolher uma das declarações é necessário configurar a variável **Requer_Data** para o formulário de listagem escolher a correta.
- **Pesquisa**: chamado pelo *Usercontrol* de pesquisa. As classes filhas só têm de reescrever este método especificando o código necessário para obter o registo código/nome a partir do código especificado pelo utilizador.

As classes nesta camada recorrem à Camada de Lógica de Negócio para carregar os dados. Na Figura 5.6 está representada essa ligação.

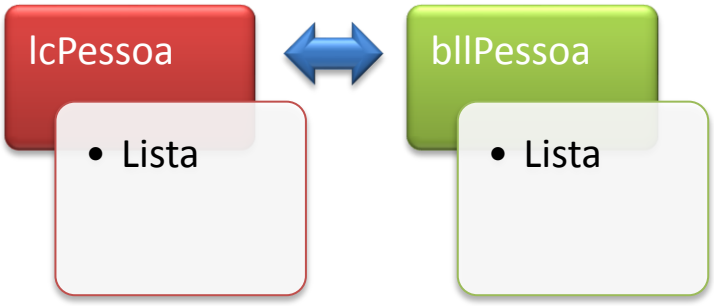


Figura 5.6: Ligação entre a Camada de Controlo de Lista e a Camada de Lógica de Negócio

5.1.1.5. Interface

Quando se cria uma nova aplicação este é o primeiro projeto a criar, uma vez que é este que dá o nome principal à aplicação. Ao criar um projeto deste tipo ele inclui já os objetos base para o funcionamento da aplicação.

- Módulos:
  - **Startup**: este módulo é responsável por carregar os dados iniciais da aplicação, como por exemplo os parâmetros de ligação à base de dados, testar a ligação à base de dados, tratar exceções e carregar o formulário principal da aplicação.
- Formulários:
  - **frmStatus**: utilizado sempre que é necessário informar o utilizador que alguma operação decorre em background, como por exemplo no arranque do sistema, quando o utilizador é informado que o programa está a carregar os parâmetros de ligação à base de dados.
  - **frmMain** (Figura 5.7): é o formulário principal da aplicação e inclui o menu que permite ao utilizador escolher que operação deseja realizar, este já contém dois separadores, o Base e a Lista. No separador Base já está incluida a opção de sair da aplicação bem como a opção de abrir uma ficha de teste, é neste que será incluído as opções de abrir os formulários de manutenção. No separador Lista já esta incluída a opção da listagem de teste, é nele que será incluído as opções de listagem de dados. O menu de contexto inclui a opção de alterar a *string* de ligação à base de dados. Existe ainda uma *StatusBar* que mostra sempre o nome da janela que está ativa, se o *Insert* ou *Caps Lock* estão ativos e as horas do sistema. Dependendo do contexto da aplicação esta informação pode variar.

Figura 5.7: Janela principal da aplicação

- **frmBD**: utilizado para configurar os parâmetros de ligação à base de dados.
- **frmGeneric**: utilizado como pai de todos os formulários para configurar algumas definições inerentes a todos os formulários.
- **frmListagem** (Figura 5.8): utilizado para listar os dados da aplicação. Ele recorre à subcamada Controlo de listas para recolher os dados e as configurações da lista. Esta dispõe de diversas opções que permitem ao utilizador personalizar a listagem de modo a obter a informação que mais lhe convém. Esta pode permitir, segundo a configuração predefinida no objeto de controlo da lista, filtrar a listagem por datas, agrupar os dados por um determinado campo, ajustar o tamanho das colunas, escolher que colunas são mostradas, filtros agregados, exportar para Excel, imprimir e calcular totais por campo (Média, Contar n.º de registos, Máximo, Mínimo e Somatório).

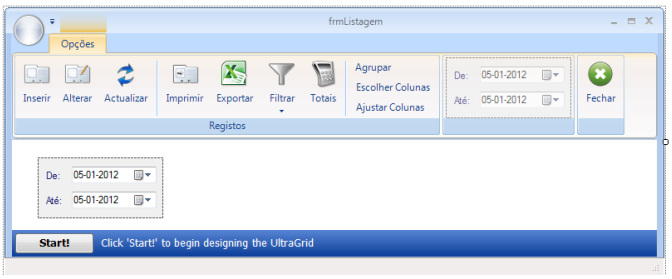


Figura 5.8: Janela de listagem

- **frmReports**: é neste formulário que são carregados e exibidos os relatórios da aplicação.
- **frm_Mnt_Geral**: este serve de base aos formulários de manipulação de dados. Possui o menu que permite ao utilizador escolher que operação realizar (inserir, alterar, eliminar e imprimir). Possui os métodos a usar em cada uma das operações, tendo os formulários filhos que ser reescritos para os adaptar a casos específicos.

- ***Usercontrol***:
    - **ucPesquisa** (Figura 5.9): este *Usercontrol* é utilizado sempre que num formulário é necessário carregar código e nome/descrição de um registo. Este recorre aos objetos de controlo de listas para carregar o registo.

Figura 5.9: *Usercontrol* de Pesquisa

Os *templates* Interface e Lista_Control são apenas utilizados em projetos *Windows Forms*, podendo todos os outros ser utilizadas em projetos para as mais diversas plataformas.

5.1.2. Formulário

Uma vez que as entidades possuem atributos distintos uma das outras, é necessário criar um formulário específico para cada uma delas, tendo-se optado por criar um *template* da janela de manutenção de dados. Esta permite inserir e alterar os dados das tabelas. Ao criar um formulário deste tipo o programador apenas terá, nos casos mais simples, que criar controlos que permitem a interação gráfica com o utilizador, e especificar os objetos da Camada de Objetos e da Camada de Lógica de Negócio a utilizar e adaptar os métodos ***Carregar_Objecto***, ***Carregar_Controlos***, ***Carregar_Dados***, ***txtCodigo_Key_Pesquisa***. O método ***Carregar_Objecto*** obtém os valores que estão nos controlos e coloca-os no objeto da Camada de Objetos para que possam ser passados à Camada de Lógica de Negócio para serem validados e posteriormente enviados para a base de dados. O método ***Carregar_Controlos*** faz o inverso, ou seja, obtém os valores que estão no objeto da Camada de Objetos e coloca-os nos controlos. Os métodos **Carregar_Dados** e **txtCodigo_Key_Pesquisa** escutam respetivamente os eventos ***Carregar_Dados*** e ***Key_Pesquisa*** do objeto *Maskbox*.

5.2. Controlos

- **MaskBox:** tem por base uma caixa de texto (desenvolvida durante no meu projeto final de licenciatura) que permite escolher que tipo de dados é que esta pode conter (texto, números, código postal, email, etc.). Foram-lhe adicionadas funcionalidades para uniformizar algumas tarefas na interface:
    - **Marca de água:** é possível ao programador colocar um texto na caixa de texto que desaparece quando esta recebe o *focus*;
    - **Ligações**: sempre que o tipo de dados for do tipo email, o utilizador pode clicar na caixa de texto e, caso tenha a tecla de *Ctrl* pressionada, é aberto o programa de correio eletrónico predefinido do sistema. O mesmo se passa com os dados do tipo web, com browser predefinido a ser iniciado;
    - **Eventos**: esta caixa de texto representa um papel importante nos formulários de manutenção e no *usercontrol* de pesquisa. É nela que o utilizador especifica o código da entidade (chave primária). A partir dela são despoletadas as ações de

pesquisa de dados no formulário de listagem e carregamento dos dados. Estão aqui contidos dois eventos para gerir estas ações:

- **Carregar_Dados:** sempre que a caixa de texto perde o *focus* ou o utilizador pressiona a tecla *Enter* este evento é acionado para informar o controlador pai que deve carregar os dados;
- **Key_Pesquisa:** sempre que o utilizador pressiona a tecla *F4* este evento é acionado para informar o controlador pai que deve chamar o formulário de pesquisa.

- **Botão de Pesquisa:** este botão encontra-se do lado direito da caixa de texto e quando clicado despoleta o evento de pesquisa.

- **DJC_Grid:** tendo por base a *Ultragrid* da Infragistics foi estendida a classe para dispor de algumas funcionalidades base que possam ser utilizadas sempre que é necessário listar dados, tais como:
  - **Pesquisa**: sempre que o utilizador se posiciona numa coluna e começa a escrever aparece uma caixa de texto onde este digita o texto a procurar. Pressionando a tecla *Enter* a *DataGrid* ordena essa coluna por ordem alfabética e posiciona-se no primeiro registo que cumpre com os requisitos da pesquisa;
  - **Imprimir**: a *Ultragid* faz parte de um conjunto de controlos da empresa Infragistics que inclui já objetos próprios para impressão diretamente da *DataGrid* e a exportação dos mesmos para formato .xls. No entanto, sempre que é necessário exportar ou imprimir é necessário criar um objeto de impressão e/ou de exportação. Assim, optou-se por juntar todos estes objetos num objeto. Como resultado, sempre que um programador precise de imprimir ou exportar apenas terá de chamar os métodos correspondentes.

5.3. Aplicações

São raras as tabelas do modelo de dados onde não são aplicadas as operações CRUD, e estas são quase sempre realizada da mesma forma, ou seja, alterando apenas os seus atributos. Isto torna a tarefa do programador muito repetitiva, uma vez que este tem de escrever o código para as três primeiras camadas em que os métodos e os modos de execução são idênticos, alterando apenas os atributos e as regras de negócio. Assim, foi desenvolvida uma aplicação **Cria_Ficheiros** que se liga à base de dados e com comandos SQL obtém todas as tabelas e respetivos atributos e com base nessa estrutura, cria na Camada de Objetos todos as classes que representam as tabelas do modelo de dados, sendo os atributos das tabelas convertidos em propriedades das classes. A aplicação cria também todas as classes na Camada de Acesso aos Dados e todas as classes na Camada de Lógica de Negócio. Por fim, a aplicação cria ainda os *script*s SQL para a criação de *Stored Procedures* que serão invocadas pela Camada de Acesso aos Dados.

Para retirar a informação das tabelas e respetivos atributos é necessário fazer a ligação entre cinco tabelas de sistema, a **SYSOBJECTS**, a **SYSCOLUMNS**, a **SYSTYPES**, a **Informati-**

on_Schema.TABLE_CONSTRAINTS e a **Information_Schema.CONSTRAINT_COLUMN_USAGE**. A tabela **SYSOBJECTS** permita saber quais as tabelas que existem na base de dados. Esta informação é utilizada nomeadamente para dar o nome às várias classes. A tabela **SYSCOLUMNS** permite saber quais os atributos da tabela e a **SYSTYPES** o nome do tipo de dados de cada um dos atributos. É com base nesta informação que é possível criar os atributos na Camada de Objetos e na Camada de Acesso ao Dados os campos que serão lidos ou enviados para a base de dados, bem como converter o tipo de dados do SQL Server para o tipo de dados do VB.Net. No SQL Server, sempre que é criada uma restrição, chave primária, estrangeira ou composta, este atribui-lhe um nome, sendo então necessário recorrer à tabela **Information_Schema.TABLE_CONSTRAINTS** para saber o nome das restrições associadas a cada uma das tabelas. Por fim é utilizada a tabela **Information_Schema.CONSTRAINT_COLUMN_USAGE** para saber que coluna ou colunas constituem essa restrição. É com base nesta informação que a aplicação cria as operações de alterar e ver a ficha de uma tabela.

Para armazenar esta informação do lado da aplicação criaram-se duas classes: Campo e Tabela. A classe Tabela armazena o nome da tabela, nome singular da tabela e uma lista de objetos do tipo Campo. A classe Campos armazena o nome, o tipo, o tamanho, a precisão, a escala, possibilidade de ser nulo, se é chave da tabela e se tem valor por defeito.

A aplicação começa por carregar uma lista de objetos do tipo Tabela e os respetivos campos para poder preencher os *templates*.

Para cada uma das camadas e *stored procedures* foram criados modelos para que a aplicação pudesse criar as várias classes e *scripts*.

5.3.1. Stored Procedures

A aplicação gera as várias *stores procedures* para um ficheiro de *script* que será executado no SQL Server, tendo-se optado por não executar o *script* diretamente pela aplicação de geração de código para que o programador possa fazer ajustes ao *script* gerado. Para todas as *stored procedures* é criado o código para verificar se já existem na base de dados e, caso existam, são eliminadas para que possam ser novamente criadas. O código seguinte é gerado antes de cada uma das *stored procedures* onde SP_Nome é substituído pelo nome desta.

```
if exists (select * from sysobjects where id =
object_id(N'SP_Nome') and OBJECTPROPERTY(id, N'IsProcedure') =
1)
drop Procedure SP_NOME
go
```

Para a *stored procedure* de Inserir foi definido o modelo seguinte.

```
create Procedure Inserir_Nome_Tabela_Singular
  @Atributos Tipo
as
    Insert into Nome_Tabela(Atributos)
    values(@Atributos)
go
```

Na tabela Tabela 5.1 são apresentadas as alterações realizadas no modelo pela aplicação.

Original	Resultado	Exemplo
Nome_Tabela_Singular	Substitui pelo nome singular da tabela.	Pessoa
@Atributos Tipo	Cria os vários parâmetros de entrada da *stored procedure* segundo os atributos da tabela, separados por vírgula.	@Codigo varchar(12), @Nome varchar(50), @Idade smallint
Nome_Tabela	Substitui pelo nome da tabela.	Pessoas
Atributos	Coloca os vários atributos da tabela, separados por vírgula.	Codigo , Nome, Idade
@Atributos	Coloca os vários parâmetros de entrada da tabela, separados por vírgula.	@Codigo , @Nome, @Idade

Tabela 5.1: Alterações ao modelo da *Stored Procedure* de Inserir dados.

No código seguinte é possivel visualizar como a aplicação cria os vários parâmetros de entrada da *stored procedure*. Para cada um dos campos da tabela é chamada a função Cabeçalho que é responsavel por codificar a informação do campo em *script* sql. A esta é ainda passada a informação se o campo é o ultimo da lista para que esta não coloque “,” no final da linha.

```
For Each i As Campo In umaTabela.Campos
   fw.WriteLine(Cabecalho(i, tr = umaTabela.Campos.Count - 1))
   tr += 1
Next
```

```
Private Function Cabecalho(ByVal umCampo As Campo, ByVal ultimo As Boole-
an) As String
        Dim result As String = ""
        If umCampo.Tipo.IndexOf("char") <> -1 Then
            result = umCampo.Nome & " " + umCampo.Tipo & "(" & umCam-
po.Precisao & ")"
        ElseIf umCampo.Tipo.IndexOf("decimal") <> -1 Then
            result = umCampo.Nome & " " & umCampo.Tipo & "(" & umCam-
po.Precisao & "," & umCampo.Escala & ")"
        Else
            result = umCampo.Nome + " " + umCampo.Tipo
        End If
        If result <> "" Then
            If Not ultimo Then
                result &= ","
            End If
            result = Chr(Keys.Tab) & "@" & result
        End If
        Return result
End Function
```

Este comportamento repete-se em todas as *stored procedures* que tenham como entrada todos os atributos da tabela. Para aquelas que só utilizam as chaves é inserido uma condição antes da chamada da função Cabecalho, sendo esta chamada apenas se o campo for chave.

```
For Each i As Campo In umaTabela.Campos
       If i.Chave Then
           fw.WriteLine(Cabecalho(i, tr = umaTabela.N_Chaves - 1))
           tr += 1
       End If
Next
```

O código seguinte é utilizado para escrever todos os atributos separados por "," no comando de inserção.

```
For Each i As Campo In umaTabela.Campos
      fw.Write(i.Nome & IIf(tr = umaTabela.Campos.Count - 1, "", ","))
      tr += 1
Next
```

Para colocar os parâmetros de entrada o código é idêntico, sendo a única diferença o acrésecimo da "@" no início de cada atributo.

Para a *stored procedure* de Alterar foi definido o modelo seguinte.

```
create Procedure Alterar_Nome_Tabela_Singular
  @Atributos Tipo
as
    Update Nome_Tabela set Atributos = @Atributos
    where Chaves = @Chaves
go
```

Na tabela Tabela 5.2 são apresentadas as alterações que são realizadas no modelo pela aplicação.

Original	Resultado	Exemplo
Nome_Tabela_Singular	Substitui pelo nome singular da tabela.	Pessoa
@Atributos Tipo	Cria os vários parâmetros de entrada da *stored procedure* segundo os atributos da tabela, separados por vírgula.	@Codigo varchar(12), @Nome varchar(50), @Idade smallint
Nome_Tabela	Substitui pelo nome da tabela.	Pessoas
Atributos = @Atributos	Iguala os vários atributos da tabela com os vários parâmetros de entrada, separados por vírgula, à exceção das chaves primárias da tabela.	Nome = @Nome, Idade = @Idade
Chaves = @Chaves	Iguala os vários atributos da tabela com os vários parâmetros de entrada, separados por vírgula, que sejam chave primária da tabela.	Codigo = @Codigo

Tabela 5.2: Alterações ao modelo da *Stored Procedure* de Alterar dados.

Para a criação da condição **Where** da comando de alteração são percorridos todos os campos da tabela e caso seja chave é adicionado à condição.

```
For Each i As Campo In umaTabela.Campos
   If i.Chave Then
     fw.Write(i.Nome & " = " & "@" & i.Nome & IIf(tr = umaTabela.N_Chaves
- 1, "", " and "))
      tr += 1
   End If
Next
```

Para a *stored procedure* de Eliminar foi definido o modelo seguinte.

```
create Procedure Eliminar_Nome_Tabela_Singular
  @Chaves Tipo
```

```
as
    Delete from Nome_Tabela
    where Chaves = @Chaves
go
```

Na tabela Tabela 5.3 são apresentadas as alterações que são realizadas no modelo pela aplicação.

Original	Resultado	Exemplo
Nome_Tabela_Singular	Substitui pelo nome singular da tabela.	Pessoa
@Chaves Tipo	Cria os vários parâmetros de entrada da *stored procedure* segundo as chaves primárias da tabela, separados por vírgula.	@Codigo varchar(12)
Nome_Tabela	Substitui pelo nome da tabela.	Pessoas
Chaves = @Chaves	Iguala os vários atributos chave da tabela com os vários parâmetros de entrada, separados por vírgula.	Codigo = @Codigo

Tabela 5.3: Alterações ao modelo da *Stored Procedure* de Eliminar dados.

Para a *stored procedure* de Listagem foi definido o modelo seguinte.

```
create Procedure ver_Nome_Tabela
as
    Select Atributos
    From Nome_Tabela
Go
```

Na tabela Tabela 5.4 são apresentadas as alterações que são realizadas no modelo pela aplicação.

Original	Resultado	Exemplo
Nome_Tabela	Substitui pelo nome da tabela.	Pessoas
Atributos	Coloca os vários atributos da tabela, separados por vírgula.	Codigo , Nome, Idade

Tabela 5.4: Alterações ao modelo da *Stored Procedure* de Listar dados.

Para a *stored procedure* de ver um registo da tabela (Ficha) foi definido o modelo seguinte.

```
create Procedure ver_Nome_Tabela_Singular
  @Chaves Tipo
as
    Select Atributos
    From Nome_Tabela
    where Chaves = @Chaves

go
```

Na tabela Tabela 5.5 são apresentadas as alterações que são realizadas no modelo pela aplicação.

Original	Resultado	Exemplo
Nome_Tabela_Singular	Substitui pelo nome singular da tabela.	Pessoa
@Chaves Tipo	Cria os vários parâmetros de entrada da *stored procedure* segundo as chaves primárias da tabela, separados por vírgula.	@Codigo varchar(12)
Atributos	Coloca os vários atributos da tabela, separados por vírgula.	Codigo , Nome, Idade
Nome_Tabela	Substitui pelo nome da tabela.	Pessoas
Chaves = @Chaves	Iguala os vários atributos chave da tabela com os vários parâmetros de entrada, separados por vírgula.	Codigo = @Codigo

Tabela 5.5: Alterações ao modelo da *Stored Procedure* de ver um registo da tabela.

5.3.2. Classes

Nesta camada todos os atributos da tabela são convertidos em propriedades da classe. Apresenta-se de seguida o modelo da classe.

```
Public Class clsNome_Tabela
    Private _Atributo As Tipo
    Public Property Atributo As Tipo
        Get
            Return _Atributo
        End Get
        Set(value As Tipo)
            _Atributo = value
        End Set
    End Property
End Class
```

Na tabela **Tabela 5.6** são apresentadas as alterações que são realizadas no modelo pela aplicação.

Original	Resultado	Exemplo
Nome_Tabela	Substitui pelo nome da tabela.	clsPessoas
Atributo	Cria as várias propriedades na classe com base nos atributos da tabela.	Private _Codigo as String Public Property Codigo as String Get Return _Codigo End Get Set(value As String) _Codigo = value End Set End Property

Tabela 5.6: Alterações ao modelo da Camada de Objetos.

No seguinte código é possivel visualizar como a aplicação cria as várias propriedades na classe. Para cada um dos campos da tabela é chamada a função Convert_Tipo, responsável por converter o tipo de dados SQL para tipo de dados VB.NET.

```
For Each c As Campo In i.Campos
      fw.WriteLine("    Private _" & c.Nome & " as " & Con-
vert_Tipo(c.Tipo))
Next
```

```
Private Function Convert_Tipo(ByVal Tipo As String) As String
      Select Case Tipo
            Case "bigint"
                Return "Long"
            Case "binary"
                Return "String"
            Case "bit"
                Return "boolean"
            Case "char"
                Return "Char"
            Case "datetime"
                Return "Datetime"
            Case "decimal"
                Return "Decimal"
            Case "float"
                Return "Single"
            Case "image"
            Case "int"
                Return "Integer"
            Case "money"
                Return "Decimal"
            Case "nchar"
                Return "Char"
            Case "ntext"
                Return "String"
            Case "numeric"
                Return "Decimal"
            Case "nvarchar"
                Return "String"
            Case "real"
                Return "Decimal"
            Case "smalldatetime"
                Return "Datetime"
            Case "smallint"
                Return "Short"
            Case "smallmoney"
                Return "Decimal"
            Case "text"
                Return "String"
            Case "timestamp"
                Return "String"
            Case "tinyint"
                Return "Short"
            Case "uniqueidentifier"
                Return "String"
            Case "varbinary"
                Return "String"
            Case "varchar"
                Return "String"
            Case "xml"
                Return "String"
            Case Else
```

```
                Return "String"
        End Select
        Return "String"
End Function
```

5.3.3. DAL

Nesta camada a aplicação gera em cada classe as funções de Inserir, Eliminar, Alterar, Ficha e Listar.

Para a função Inserir foi definido o seguinte modelo.

```
Public Sub Inserir(ByVal umNome_Tabela_Singula As clsNome_Tabela)
        Dim aux_command As New SqlCommand("Inserir_Nome_Tabela_Singula")
        aux_command.CommandType = CommandType.StoredProcedure
        aux_command.Parameters.AddWithValue("@Atributo",
umNome_Tabela_Singula.Atributo)
        DALConn.ExecuteNonQuery(aux_command)
End Sub
```

Para cada um dos atributos da tabela é adicionado um parâmetro no comando SQL. No código seguinte permite visualizar como é que a aplicação adiciona os vários parâmetros ao comando.

```
For Each c As Campo In umaTabela.Campos
       If Convert_Tipo(c.Tipo).ToLower = "string" Then
          fw.WriteLine("        aux_command.Parameters.AddWithValue(""@"
& c.Nome & """,IIf(String.IsNullOrEmpty(um" & umaTabela.Nome_Singular &
"." & c.Nome & "), System.DBNull.Value, um" & umaTabela.Nome_Singular &
"." & c.Nome & "))")
       Else
          fw.WriteLine("        aux_command.Parameters.AddWithValue(""@"
& c.Nome & """, um" & umaTabela.Nome_Singular & "." & c.Nome & ")")
       End If
Next
```

Este procedimento é idêntico para as funções de Alterar e Eliminar, só que nesta ultima só são utilizadas as chaves da tabela.

Para a função Alterar foi definido o seguinte modelo.

```
Public Sub Alterar(ByVal umNome_Tabela_Singula As clsNome_Tabela)
        Dim aux_command As New SqlCommand("Alterar_Nome_Tabela_Singula")
        aux_command.CommandType = CommandType.StoredProcedure
        aux_command.Parameters.AddWithValue("@Atributo",
umNome_Tabela_Singula.Atributo)
        DALConn.ExecuteNonQuery(aux_command)
End Sub
```

Para a função Eliminar foi definido o seguinte modelo.

```
Public Sub Eliminar(ByVal Chaves As Tipo)
        Dim aux_command As New SqlCommand("Eliminar_Nome_Tabela_Singula")
        aux_command.CommandType = CommandType.StoredProcedure
        aux_command.Parameters.AddWithValue("@Chave", Chaves)
        DALConn.ExecuteNonQuery(aux_command)
End Sub
```

Enquanto que nas funções de Inserir e Alterar estas recebem como parâmetro de entrada um objeto da Camada de Objetos, a função eliminar recebe as chaves da tabela. No código seguinte permite visualizar como é que a aplicação cria os parâmetros de entrada.

```
fw.WriteLine("    Public Sub Eliminar(" & Carregar_Chaves(umaTabela) &
")")
```

```
Private Function Carregar_Chaves(ByVal umaTabela As Tabela) As String
   Dim result As String = ""
   For Each i As Campo In umaTabela.Campos
      If i.Chave Then result += "ByVal " & i.Nome & " as " & Con-
vert_Tipo(i.Tipo) & ", "
   Next
   If result.EndsWith(", ") Then result = result.Substring(0, re-
sult.Length - 2)
   Return result
End Function
```

Para a função Lista foi definido o seguinte modelo.

```
Public Function Lista() As List(Of clsTabela_Nome)
   Dim aux_lista As New List(Of clsTabela_Nome)
   Dim aux_conn As Boolean = False
   Dim dados As SqlDataReader = Nothing
   Try
       Dim aux_command As New SqlCommand("ver_Tabela_Nome")
       aux_command.CommandType = CommandType.StoredProcedure
       aux_conn = DALConn.Iniciar_Conecao
       dados = DALConn.ExecuteDataReader(aux_command)
       If dados IsNot Nothing Then
           While dados.Read
               Dim umTabela_Nome_Singular As New clsTabela_Nome
               umTabela_Nome_Singular.Atributo = IIfs.Item("Atributo") Is
System.DBNull.Value, Nothing, dados("Atributo"))
               aux_lista.Add(umTabela_Nome_Singular)
           End While
           dados.Close()
       End If
       If aux_conn Then DALConn.Fechar_Conecao()
   Catch ex As Exception
            If dados IsNot Nothing Then dados.Close()
            If aux_conn Then DALConn.Fechar_Conecao()
            Throw ex
   End Try
   Return aux_lista
End Function
```

Por cada registo proveniente da tabela é criado e carregado um objeto da Camada de Objetos e adicionada à lista que será devolvida pela função. No código seguinte é possível visualizar como a aplicação gera o código para atribuir os valores provenientes da base de dados com as propriedades da classe.

```
fw.WriteLine("               While dados.Read")
fw.WriteLine("                   Dim um" & umaTabela.Nome_Singular & "
As New cls" & umaTabela.Nome & "")
For Each c As Campo In umaTabela.Campos
   fw.WriteLine("                   um" & umaTabela.Nome_Singular & "."
& c.Nome & " = IIf(dados.Item(""" & c.Nome & """) Is System.DBNull.Value,
Nothing, dados.Item(""" & c.Nome & """))")
Next
```

```
fw.WriteLine("                    aux_lista.Add(um" &
umaTabela.Nome_Singular & ")")
fw.WriteLine("                End While")
fw.WriteLine("                dados.Close()")
fw.WriteLine("            End If")
```

A função Ficha é idêntica à anterior mas recebe como parâmetro as chaves da tabela para ser possível filtrar os dados.

```
Public Function Ficha(ByVal Chaves As Tipo) As clsTabela_Nome
    Ficha = Nothing
    Dim aux_conn As Boolean = False
    Dim dados As SqlDataReader = Nothing
    Try
        Dim aux_command As New SqlCommand("ver_Nome_Tabela_Singular")
        aux_command.CommandType = CommandType.StoredProcedure
        aux_command.Parameters.AddWithValue("@Chave", Chave)
        aux_conn = DALConn.Iniciar_Conecao
        dados = DALConn.ExecuteDataReader(aux_command)
        If dados IsNot Nothing Then
            If dados.Read Then
                Ficha = New clsTabela_Nome
                umTabela_Nome_Singular.Atributo = IIf(dados.Item("Atributo") Is
System.DBNull.Value, Nothing, dados.Item("Atributo"))
            End If
            dados.Close()
        End If
        If aux_conn Then DALConn.Fechar_Conecao()
    Catch ex As Exception
        If dados IsNot Nothing Then dados.Close()
        If aux_conn Then DALConn.Fechar_Conecao()
        Throw ex
    End Try
End Function
```

O modo de carregar as chaves é idêntico ao da função Eliminar.

5.3.4. BLL

Os modelos das várias funções nesta camada são idênticos visto que esta, no modo mais simples, apenas efetua a validação dos dados nas operações Inserir e Alterar. A aplicação cria em todas as classes a função de Validação com o seguinte modelo.

```
Public Overloads Shared Function Valida(ByVal umNome_Tabela_Singular As
clsNome_Tabela) As Boolean
   Dim result as String = ""

   'If   String.IsNullOrEmpty(umNome_Tabela_Singular.)   Then   result   &=
"Erro!" & vbNewLine

   If result <> "" Then Throw New Exception(result)
   Return True
End Function
```

Para a função Inserir foi definido o seguinte modelo.

```
Public Overloads Shared Sub Inserir(ByVal umNome_Tabela_Singular As
clsNome_Tabela)
    If Valida(umNome_Tabela_Singular) Then
        adp_aux = dalNome_Tabela.GetInstance
        adp_aux.Inserir(umNome_Tabela_Singular)
    End If
```

```
End Sub
```

Para a função Eliminar foi definido o seguinte modelo.

```
Public Overloads Shared Sub Eliminar(ByVal Chaves As Tipo)
    adp_aux = dalNome_Tabela.GetInstance
    adp_aux.Eliminar(Chaves)
End Sub
```

Para a função Alterar foi definido o seguinte modelo.

```
Public Overloads Shared Sub Alterar(ByVal umNome_Tabela_Singular As
clsNome_Tabela)
    If Valida(umNome_Tabela_Singular) Then
        adp_aux = dalNome_Tabela.GetInstance
        adp_aux.Alterar(umNome_Tabela_Singular)
    End If
End Sub
```

Para a função Lista foi definido o seguinte modelo.

```
Public Overloads Shared Function Lista() As List(Of clsNome_Tabela)
    adp_aux = dalNome_Tabela.GetInstance
    Return adp_aux.Lista
End Function
```

Para a função Ficha foi definido o seguinte modelo.

```
Public Overloads Shared Function Ficha(ByVal Chaves As Tipo) As
clsNome_Tabela
    adp_aux = dalNome_Tabela.GetInstance
    Return adp_aux.Ficha(Chaves)
End Function
```

5.4. Conclusão

Com base nesta informação, a aplicação consegue gerar todas as classes e *scripts* necessários para a criação das operações básicas a aplicar a uma tabela.

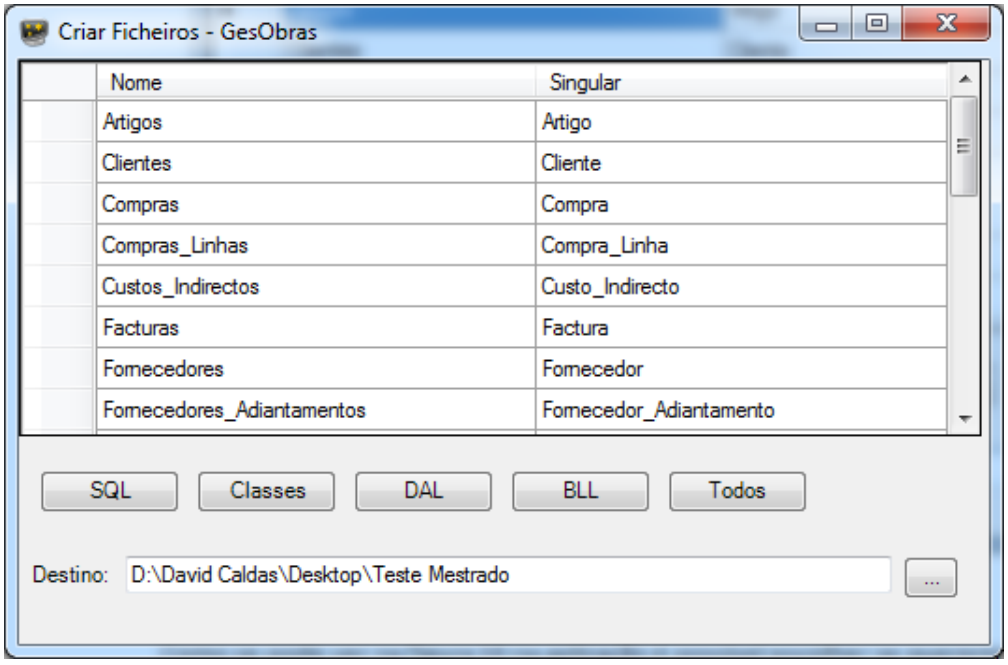


Figura 5.10: Aplicação de criação de ficheiros

Como se pode observar na Figura 5.10, na aplicação é possível escolher se queremos criar o *script* para a base de dados ou as classes para cada uma das camadas, ou gerar tudo ao mesmo tempo e qual a pasta de destino do código gerado. Durante a criação do código é necessário saber qual o nome singular das tabelas, para que o nome de alguns procedimentos e variáveis façam sentido, por isso é necessário preencher a coluna dos nomes singulares, está já vem inicialmente preenchida com os nomes no plural sendo só necessário fazer os ajustes para passar a singular.

5.5. Caso prático

Recorrendo à tabela de Artigos (Figura 5.11) do nosso caso de estudo apresentamos de seguida as mais-valias da utilização da *framework* desenvolvida. Iremos mostrar os passos necessários para a manipulação dos artigos por parte do utilizador, desde a criação dos *Stored Procedures* até ao formulário de manipulação de dados.

Artigos	
PK	**Artigo**
FK1	Nome Unidade

Figura 5.11: Entidade Artigos

O primeiro passo é a criação da tabela de artigos no Sistema de Gestão de Base de Dados (SGBD). Depois de criada a tabela executa-se a aplicação ***Criar_Ficheiros***. A aplicação cria os *Stored Procedures* (**Inserir_Artigo**, **Alterar_Artigo**, **Eliminar_Artigo, ver_Artigos** e **ver_Artigo**), a classe ***clsArtigos*** com as propriedades Artigo, Nome e Unidade, a classe ***dalArtigos*** com as funções Inserir, Eliminar, Alterar, Lista e Ficha e a classe ***bllArtigos*** com as funções Inserir, Eliminar, Alterar, Lista e Ficha.

Todas as linhas de código apresentadas de seguida são integralmente geradas pela aplicação.

- ***Scripts* SQL *(Stored Procedures):*** serão invocadas pela Camada de Acesso aos Dados
    - **Inserir_Artigo:** usada para inserir um artigo, esta recebe como parâmetros o código, nome e unidade do artigo. Como se pode ver a *Stored Procedure* chama-se **Inserir_Artigo** e não **Inserir_Artigos** isto acontece porque a aplicação utiliza neste caso o nome da tabela no singular que foi especificado na coluna dos nomes singulares. Este princípio é utilizado sempre que faça sentido no nome de algumas variáveis/objetos e funções/procedimentos.

```
create Procedure Inserir_Artigo
   @Artigo nvarchar(15),
   @Nome nvarchar(50),
   @Unidade nvarchar(5)
as
    Insert into Artigos(Artigo,Nome,Unidade)
    values(@Artigo,@Nome,@Unidade)
go
```

- **Alterar_Artigo:** usada para alterar um artigo, esta recebe como parâmetros o código do artigo a ser alterado e o novo nome e unidade do artigo.

```
create Procedure Alterar_Artigo
   @Artigo nvarchar(15),
   @Nome nvarchar(50),
   @Unidade nvarchar(5)
as
    Update Artigos set Nome = @Nome,
Unidade = @Unidade
    where Artigo = @Artigo
go
```

- **Eliminar_Artigo:** usada para eliminar um artigo, esta recebe como parâmetro o código do artigo a ser eliminado.

```
create Procedure Eliminar_Artigo
   @Artigo nvarchar(15)
as
    Delete from Artigos
    where Artigo = @Artigo
go
```

- **ver_Artigos:** usada para ver todos os artigos.

```
create Procedure ver_Artigos
as
    Select Artigo,Nome,Unidade
    From Artigos
go
```

- **ver_Artigo:** usada para ver um artigo, esta recebe como parâmetro o código do artigo a ser visualizado.

```
create Procedure ver_Artigo
   @Artigo nvarchar(15)
as
    Select Artigo,Nome,Unidade
    From Artigos
    where Artigo = @Artigo
go
```

- **clsArtigos**: Classe Artigos criada na Camada de Objetos.

```
Public Property Artigo() As String
    Get
        Return _Artigo
    End Get
```

```
    Set(ByVal value As String)
        _Artigo = value
    End Set
End Property
Public Property Nome() As String
    Get
        Return _Nome
    End Get
    Set(ByVal value As String)
        _Nome = value
    End Set
End Property
Public Property Unidade() As String
    Get
        Return _Unidade
    End Get
    Set(ByVal value As String)
        _Unidade = value
    End Set
End Property
```

- **dalArtigos:** Classe Artigos criada na Camada de Acesso aos Dados e possui os seguintes métodos:
    - **Inserir:** este procedimento recebe como parâmetro um objeto do tipo **clsArtigo**, cria um comando SQL, invocando a *Stored Procedure* **Inserir_Artigo**, carrega-o com os dados do objeto e envia esse comando para a função **ExecuteNonQuery** da classe **DLConn**.

```
Public Sub Inserir(ByVal umArtigo As clsArtigos)
    Dim aux_command As New SQLCommand("Inserir_Artigo")
    aux_command.CommandType = CommandType.StoredProcedure
aux_command.Parameters.AddWithValue("@Artigo",
IIf(String.IsNullOrEmpty(umArtigo.Artigo), System.DBNull.Value, umArti-
go.Artigo))

aux_command.Parameters.AddWithValue("@Nome",IIf(String.IsNullOrEmpty(umAr
tigo.Nome), System.DBNull.Value, umArtigo.Nome))

aux_command.Parameters.AddWithValue("@Unidade",IIf(String.IsNullOrEmpty(u
mArtigo.Unidade), System.DBNull.Value, umArtigo.Unidade))
    DALConn.ExecuteNonQuery(aux_command)
End Sub
```

    - **Alterar:** este procedimento é idêntico ao procedimento de inserir só altera a *Stored Procedure* invocada (**Alterar_Artigo**).

```
Public Sub Alterar(ByVal umArtigo As clsArtigos)
```

```
    Dim aux_command As New SQLCommand("Alterar_Artigo")
    aux_command.CommandType = CommandType.StoredProcedure
    aux_command.Parameters.AddWithValue("@Artigo",
IIf(String.IsNullOrEmpty(umArtigo.Artigo), System.DBNull.Value, umArti-
go.Artigo))
    aux_command.Parameters.AddWithValue("@Nome",
IIf(String.IsNullOrEmpty(umArtigo.Nome), System.DBNull.Value, umArti-
go.Nome))
    aux_command.Parameters.AddWithValue("@Unidade",
IIf(String.IsNullOrEmpty(umArtigo.Unidade), System.DBNull.Value, umArti-
go.Unidade))
    DALConn.ExecuteNonQuery(aux_command)
End Sub
```

- **Eliminar**: este procedimento recebe como parâmetro o código do artigo a alterar, cria um comando SQL, invocando *a Stored Procedure* **Eliminar_Artigo**, carrega o comando com o código do artigo a eliminar e envia esse comando para a função **ExecuteNonQuery** da classe **DLConn**.

```
Public Sub Eliminar(ByVal Artigo as String)
    Dim aux_command As New SQLCommand("Eliminar_Artigo")
    aux_command.CommandType = CommandType.StoredProcedure
    aux_command.Parameters.AddWithValue("@Artigo", Artigo)
    DALConn.ExecuteNonQuery(aux_command)
End Sub
```

- **Lista:** esta função não tem parâmetro de entrada, esta devolve uma lista de objetos do tipo **clsArtigos**. Esta função cria um comando SQL, invocando a *Stored Procedure* **ver_Artigos**, esse comando para a função **ExecuteDataReader** da classe **DLConn**, esta devolve um **SQLDataReader** com os dados dos artigos, este é percorrido linha a linha para carregar a lista de artigos.

```
Public Function Lista() As List(Of clsArtigos)
    Dim aux_lista As New List(Of clsArtigos)
    Dim aux_conn As Boolean = False
    Dim dados As SQLDataReader = Nothing
    Try
        Dim aux_command As New SQLCommand("ver_Artigos")
        aux_command.CommandType = CommandType.StoredProcedure
        aux_conn = DALConn.Iniciar_Conecao
        dados = DALConn.ExecuteDataReader(aux_command)
        If dados IsNot Nothing Then
            While dados.Read
                Dim umArtigo As New clsArtigos
                umArtigo.Artigo = IIf(dados.Item("Artigo") Is Sys-
tem.DBNull.Value, Nothing, dados.Item("Artigo"))
```

```
                umArtigo.Nome = IIf(dados.Item("Nome") Is Sys-
tem.DBNull.Value, Nothing, dados.Item("Nome"))
                umArtigo.Unidade = IIf(dados.Item("Unidade") Is Sys-
tem.DBNull.Value, Nothing, dados.Item("Unidade"))
                aux_lista.Add(umArtigo)
            End While
            dados.Close()
        End If
        If aux_conn Then DALConn.Fechar_Conecao()
    Catch ex As Exception
        If dados IsNot Nothing Then dados.Close()
        If aux_conn Then DALConn.Fechar_Conecao()
        Throw ex
    End Try
    Return aux_lista
End Function
```

- **Ficha:** esta função recebe como parâmetro o código do artigo a visualizar e devolve um objeto do tipo **clsArtigo** com os dados. O funcionamento desta função é idêntico ao da anterior, só que neste caso o *Stored Procedure* invocando é **ver_Artigo**.

```
Public Function Ficha(ByVal Artigo as String) As clsArtigos
    Ficha = Nothing
    Dim aux_conn As Boolean = False
    Dim dados As SQLDataReader = Nothing
    Try
        Dim aux_command As New SQLCommand("ver_Artigo")
        aux_command.CommandType = CommandType.StoredProcedure
        aux_command.Parameters.AddWithValue("@Artigo", Artigo)
        aux_conn = DALConn.Iniciar_Conecao
        dados = DALConn.ExecuteDataReader(aux_command)
        If dados IsNot Nothing Then
            If dados.Read Then
                Ficha = New clsArtigos
                Ficha.Artigo = IIf(dados.Item("Artigo") Is Sys-
tem.DBNull.Value, Nothing, dados.Item("Artigo"))
                Ficha.Nome = IIf(dados.Item("Nome") Is Sys-
tem.DBNull.Value, Nothing, dados.Item("Nome"))
                Ficha.Unidade = IIf(dados.Item("Unidade") Is Sys-
tem.DBNull.Value, Nothing, dados.Item("Unidade"))
            End If
            dados.Close()
        End If
        If aux_conn Then DALConn.Fechar_Conecao()
    Catch ex As Exception
```

```
        If dados IsNot Nothing Then dados.Close()
        If aux_conn Then DALConn.Fechar_Conecao()
        Throw ex
    End Try
End Function
```

- **bllArtigos:** Classe Artigos criada na Camada de Lógica de Negócio. Os procedimentos desta classe são idênticos entre eles, recebem os parâmetros da Camada de Apresentação, nas funções de inserir e alterar valida-os e envia-os para a Camada de Acesso aos Dados.

```
Private Shared adp_aux as dalArtigos


Public Overloads Shared Function Valida(ByVal umArtigo As clsArtigos) As
Boolean
   Dim result as String = ""
   'If String.IsNullOrEmpty(umArtigo.) Then result &= "Erro!" & vbNewLine
   If result <> "" Then Throw New Exception(result)
   Return True
End Function
Public Overloads Shared Sub Inserir(ByVal umArtigo As clsArtigos)
    If Valida(umArtigo) Then
        adp_aux = dalArtigos.GetInstance
        adp_aux.Inserir(umArtigo)
    End If
End Sub
Public Overloads Shared Sub Eliminar(ByVal Artigo as String)
    adp_aux = dalArtigos.GetInstance
    adp_aux.Eliminar(Artigo)
End Sub
Public Overloads Shared Sub Alterar(ByVal umArtigo As clsArtigos)
    If Valida(umArtigo) Then
        adp_aux = dalArtigos.GetInstance
        adp_aux.Alterar(umArtigo)
    End If
End Sub
Public Overloads Shared Function Lista() As List(Of clsArtigos)
    adp_aux = dalArtigos.GetInstance
    Return adp_aux.Lista
End Function
Public Overloads Shared Function Ficha(ByVal Artigo as String) as clsAr-
tigos
    adp_aux = dalArtigos.GetInstance
    Return adp_aux.Ficha(Artigo)
```

```
End Function
```

Seguidamente é necessário que o programador crie a classe de Artigos para a Camada de Controlo de Listas (*lcArtigos*). Aqui é necessário configurar a classe para que o formulário de listagem mostre os dados conforme o pretendido. Neste caso é especificado o nome do formulário de manutenção (***frm_Mnt_Artigos***), a chave primária da tabela (***Artigo***), o título da janela de listagem (***Lista de Artigos***), especificado que esta listagem não é filtrada por datas, que o utilizador pode escolher as colunas que deseja visualizar e que pode ajustar o tamanho das mesmas. Depois é necessário configurar as colunas da *DataGrid*, bem como especificar no método Lista qual a classe da Camada de Lógica de Negócio e desta o método a ser chamado (***bllArtigos.Lista***).

```
Public Overrides Sub Carregar_Parametros()
    frm_Manutencao = "frm_Mnt_Artigos"
    Lista_Codigos.Add("Artigo")
    Listagem_Titulo = "Lista de Artigos"
    Requer_Data = False
    Active_Band = 0
    Escolher_Colunas = True
    Ajustar_Colunas = True
End Sub
Public Overrides Sub Carregar_Colunas(ByRef dataGrid As DJC_Grid.MyGrid)
    Dim UltraGridBand1 As UltraGridBand = New UltraGrid-
Band(dataGrid.DataMember, -1)
    Dim UltraGridColumn1 As UltraGridColumn
    Dim INDEX As Short = 0
    UltraGridColumn1 = New UltraGridColumn("Artigo")
    UltraGridColumn1.Header.Caption = "Código"
    UltraGridColumn1.CellActivation = Activation.NoEdit
    UltraGridColumn1.Width = 100
    UltraGridColumn1.Header.VisiblePosition = INDEX
    INDEX += 1
    UltraGridBand1.Columns.Add(UltraGridColumn1)

    UltraGridColumn1 = New UltraGridColumn("Nome")
    UltraGridColumn1.Header.Caption = "Nome"
    UltraGridColumn1.CellActivation = Activation.NoEdit
    UltraGridColumn1.Width = 100
    UltraGridColumn1.Header.VisiblePosition = INDEX
    INDEX += 1
    UltraGridBand1.Columns.Add(UltraGridColumn1)
    UltraGridColumn1 = New UltraGridColumn("Unidade")
    UltraGridColumn1.Header.Caption = "Unidade"
```

```
    UltraGridColumn1.CellActivation = Activation.NoEdit
    UltraGridColumn1.Width = 100
    UltraGridColumn1.Header.VisiblePosition = INDEX
    INDEX += 1
    UltraGridBand1.Columns.Add(UltraGridColumn1)
    dataGrid.DisplayLayout.BandsSerializer.Add(UltraGridBand1)
    dataGrid.DisplayLayout.ViewStyle = ViewStyle.SingleBand
End Sub

Public Overrides Function Lista() As Object
   Return bllArtigos.Lista
End Function

Public Overrides Function Pesquisa(ByVal Codigo As String) As Clas-
ses.clsSimples
    Return bllArtigos.Pesquisa(Codigo)
End Function
```

Para que esta classe possa ser utilizada no *ucPesquisa* é necessário implementar o método Pesquisa e criar o método de pesquisa na Camada de Lógica de Negócio e na Camada de Acesso aos Dados. Na Camada de Lógica de Negócio o método é idêntico aos restantes. Na Camada de Acesso aos Dados o método é igual ao que permite ver a ficha do artigo, sendo neste caso apenas carregados os dados referentes ao código e ao nome do artigo. Este método devolve um objeto do tipo simples (***clsSimples***) que contém as propriedades código e nome.

Na Camada de Apresentação é criado o formulário de manutenção (Figura 5.12) que será usado para manipular os dados dos artigos.

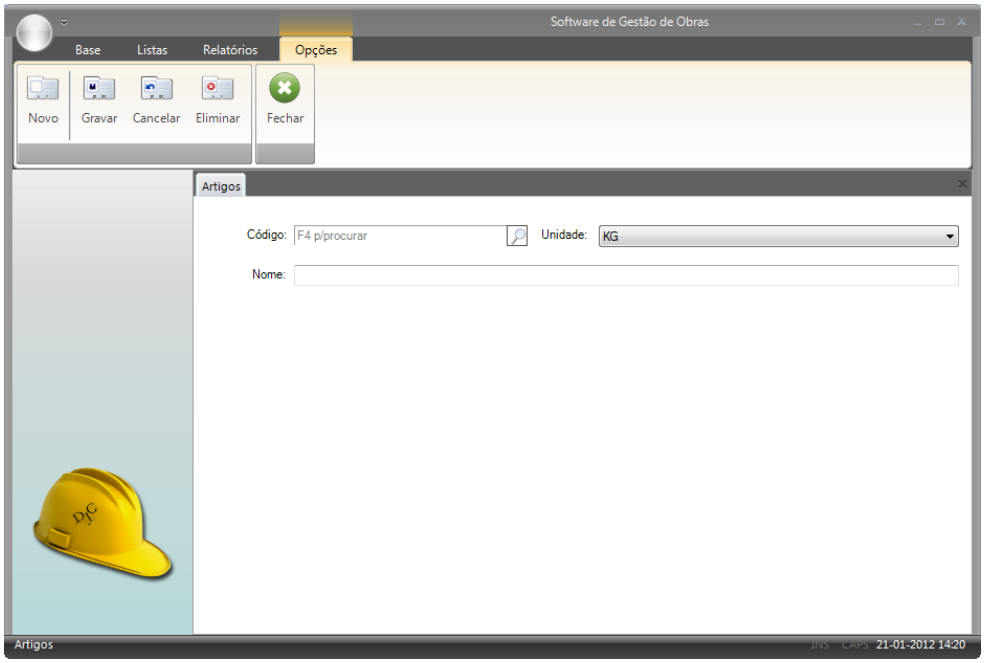


Figura 5.12: Formulário de manutenção de artigos

Aqui é necessário inserir um controlo do tipo **MaskBox** para o código **txtCodigo**, uma caixa de combinação para a unidade **cboUnidade** e uma **MaskBox** para o nome do artigo **txtNome**. De seguida é necessário adaptar os métodos **Carregar_Objecto**, **Carregar_Controlos**, **Carregar_Dados**, **txtCodigo_Key_Pesquisa**:

```
Private Sub Carregar_Objecto()
    If umArtigo Is Nothing Then umArtigo = New clsArtigos
    umArtigo.Artigo = txtCodigo.Text
    umArtigo.Nome = txtNome.Text
    umArtigo.Unidade = cboUnidade.SelectedValue
End Sub
Private Sub Carregar_Controlos()
    txtCodigo.Text = umArtigo.Artigo
    txtNome.Text = umArtigo.Nome
    If Not String.IsNullOrEmpty(umArtigo.Unidade) Then _
   cboUnidade.SelectedValue = umArtigo.Unidade
End Sub
Private Sub txtCodigo_Key_Pesquisa(ByVal sender As Object) Handles txtCo-
digo.Key_Pesquisa
    Dim frm As New frmListagem(New lcArtigos)
    frm.Importar_Dados(New MaskBox() {txtCodigo})
    aux_focus = txtCodigo
    showform(Me.MdiParent, frm)
End Sub
Private Sub Carregar_Dados() Handles txtCodigo.Carregar_Dados
   If txtCodigo.Text <> "" Then
        If txtCodigo.Text <> Codigo Then
            umArtigo = bllArtigos.Ficha(txtCodigo.Text)
            If umArtigo IsNot Nothing Then
                Codigo = umArtigo.Artigo
                Carregar_Controlos()
             Else
                Codigo = ""
             End If
        End If
   Else
      Limpar_Campos()
   End If
End Sub
```

Como se pode ver no método **txtCodigo_Key_Pesquisa**, para que o formulário de listagem apresente a lista de artigos basta passar o objeto do tipo **lcArtigos** como parâmetro do construtor. O formulário listagem, ao ser carregado, chama o seu método **Carregar_Dados**:

```
Public Overridable Sub Carregar_Dados()
   adp_aux.Carregar_Colunas(dgListagem)
   If adp_aux.Requer_Data Then
     dgListagem.DataSource = adp_aux.Lista(dtpInicio.Value, dtpFim.Value)
   Else
      dgListagem.DataSource = adp_aux.Lista
   End If
   UltraStatusBar1.Text = "Total: " & dgListagem.Rows.Count & " regis-
to(s)."
   If Not dgListagem.Rows.FirstVisibleCardRow Is Nothing Then
      dgListagem.ActiveRow = dgListagem.Rows.FirstVisibleCardRow
      dgListagem.ActiveCell = dgListagem.ActiveRow.Cells(0)
      dgListagem.ActiveRow.Selected = True
   End If
End Sub
```

Este método analisa se a lista requer ou não datas e carrega da *DJC_Grid*, invocando o método adequado.

No caso da pesquisa do código é ainda passado ao formulário de listagem o controlo **txtCodigo** para que este possa ser carregado com o código escolhido pelo utilizador.

```
Private Sub frmListagem_KeyDown(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown
   If e.KeyValue = Keys.F4 Or e.KeyValue = Keys.Escape Then
      If (aux_importar IsNot Nothing OrElse aux_form IsNot Nothing) And
dgListagem.ActiveRow IsNot Nothing And e.KeyValue = Keys.F4 Then
         If aux_importar IsNot Nothing Then
            For i As Integer = 0 To adp_aux.Lista_Codigos.Count - 1
               If dgListagem.ActiveRow.VisibleIndex <> -1 Then
                  aux_importar(i).Carrega = dgLista-
gem.ActiveRow.Cells(adp_aux.Lista_Codigos.Item(i)).Text
               End If
            Next
         End If
      End If
      Me.Close()
   End If
End Sub
```

Relativamente ao *Usercontrol* este funciona do mesmo modo que um formulário de manutenção só que neste caso apenas são carregados o código e o nome.

Capítulo 6 - **Desenvolvimento**

Após a fase de análise do problema, criação do modelo de dados, especificação do *software* e desenvolvimento da *framework*, passou-se para o desenvolvimento da aplicação.

6.1. Modelo de desenvolvimento

Para o desenvolvimento deste *software* optou-se por utilizar o modelo de desenvolvimento em Cascata (Sommerville, 2007). Este define cinco fases para o desenvolvimento do *software*, estando estas representadas na Figura 6.1.

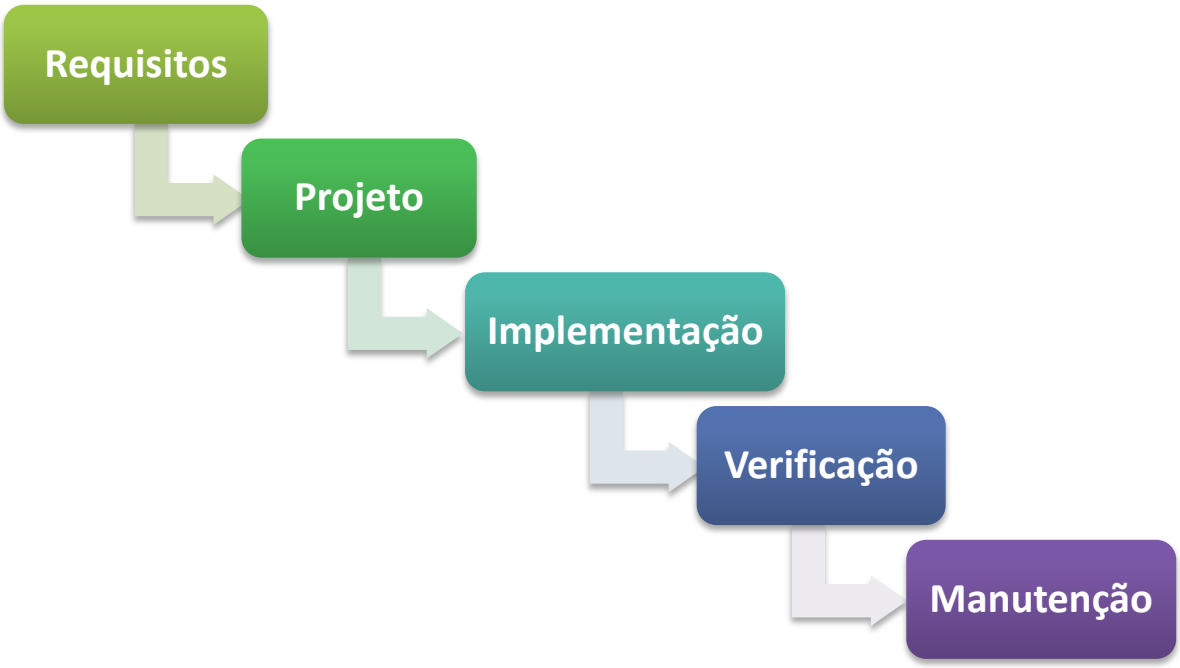


Figura 6.1: Modelo em Cascata

Na fase de Requisitos é feita a análise dos requisitos do sistema. Na fase de Projeto é feita a modelação dos dados e definida a arquitetura do *software*. Na fase seguinte, Implementação, é feita a implementação do *software*. Na fase de Verificação são realizados os testes para confirmar se a aplicação corresponde objetivos especificados. Na última fase, Manutenção, é feita a manutenção da mesma.

Sendo este projeto de realização individual, e com etapas bem definidas, concluiu-se que este seria o modelo mais adequado ao desenvolvimento deste projeto.

6.2. Aplicação

Foi criada a base de dados, de acordo com o modelo de dados, seguindo-se a criação de um projeto do tipo interface ao qual se deu o nome **GesObras**. Foram depois criados os restantes projetos (Classes, DAL, BLL). Na Figura 6.2 apresenta-se a Solução tal como é representada no *Visual Studio 2010* depois da criação de todos os projetos.

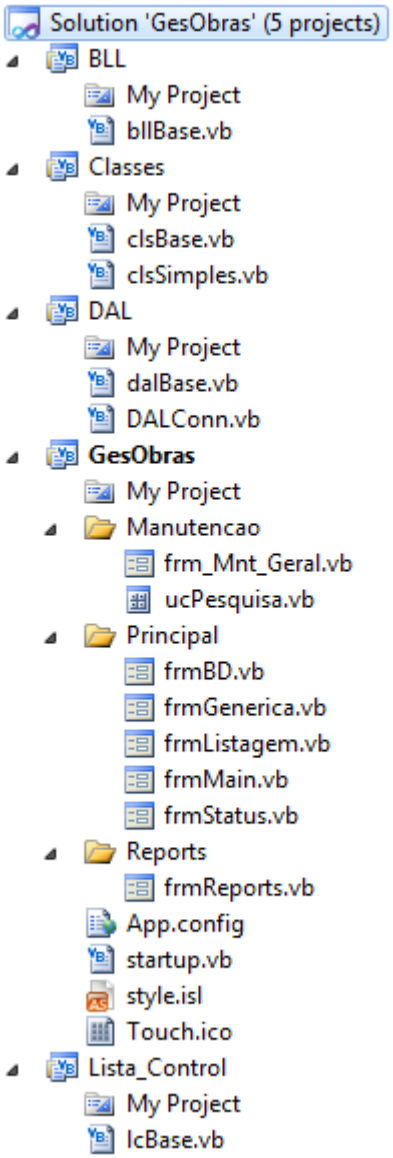


Figura 6.2: Solução depois da criação de todos os projetos

Após a criação de todos os projetos, executou-se a aplicação **Cria_Ficheiros** para gerar o código para as três camadas base e o *script* SQL para a criação dos *Stored Procedure*. Passou-se depois à criação dos formulários e das classes para a camada **Lista_Controlo**.

Os formulários que permitem gerir os clientes, fornecedores, custos indiretos e secções são idênticos ao formulário de gestão de artigos apresentado no caso prático, não sendo por isso caso de análise.

O formulário de gestão de funcionário (Figura 6.3) difere dos anteriores, uma vez que um funcionário pode ter diversos custos/hora. No formulário existe uma *DataGrid* que permite ao utilizador especificar o tipo de hora e o preço/hora.

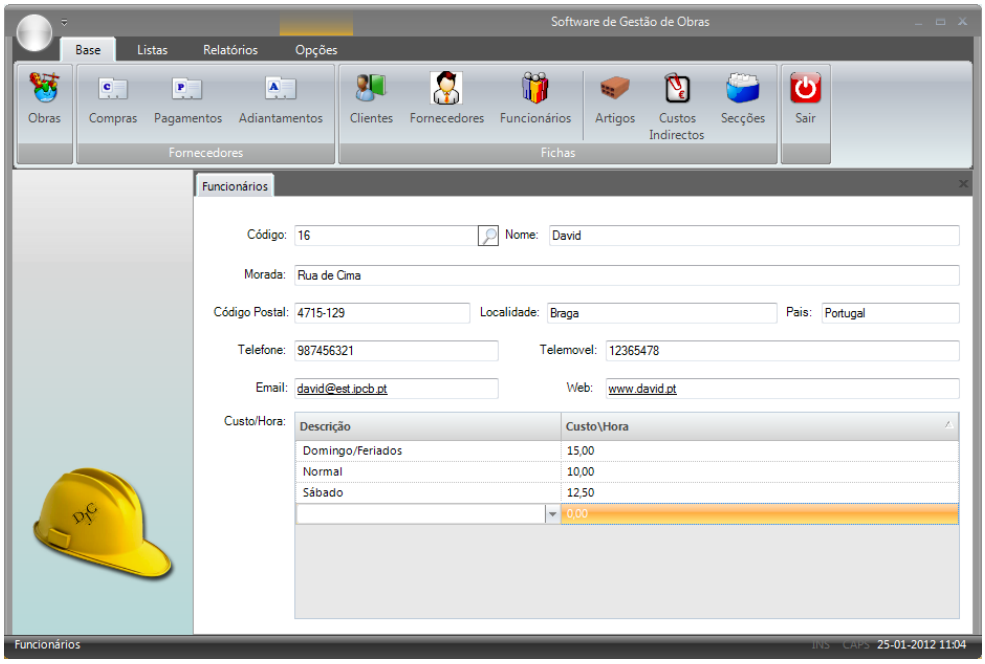


Figura 6.3: Formulário de gestão de funcionários

A aplicação desenvolvida para a criação de código automático não prevê esta situação, sendo por isso necessário fazer alterações às classes geradas. Primeiro, é necessário alterar a classe ***clsFuncionarios*** e adicionar-lhe uma lista do tipo ***clsFuncionarios_Custo_Hora***, para que as horas acompanhem a ficha do funcionário. No código seguinte está representada essa alteração.

```
Private _CustoHora As List(Of clsFuncionarios_Custo_Hora)
```

De seguida, é necessário alterar a classe ***dalFuncionarios*** para que ao executar as funções CRUD à ficha esta aplique também as alterações na tabela de custos/hora.

Para o caso do procedimento de "inserir", foi necessário criar uma transação, para que caso exista algum erro numa das operações seja possível reverter as operações. Depois de inserir os dados referentes à ficha do funcionário, são inseridos os dados referentes ao custo/hora do mesmo. No código seguinte está representado a função "inserir" já com as alterações mencionadas anteriormente. Nele é possível ver a criação da transação, a inserção dos dados relativos à ficha do funcionário e por fim a chamada à função ***Inserir_Lista*** da classe ***dalFuncionarios_Custo_Hora*** para inserir os dados referentes aos vários custo/hora do funcionário.

```
Public Sub Inserir(ByVal umFuncionario As clsFuncionarios)
    Dim iniciar As Boolean = False
    Try
        iniciar = DALConn.Iniciar_Transaction
        Dim aux_command As New SQLCommand("Inserir_Funcionario")
        aux_command.CommandType = CommandType.StoredProcedure
        aux_command.Parameters.AddWithValue("@Funcionario",
IIf(String.IsNullOrEmpty(umFuncionario.Funcionario), System.DBNull.Value,
umFuncionario.Funcionario))
```

```
        aux_command.Parameters.AddWithValue("@Nome",
IIf(String.IsNullOrEmpty(umFuncionario.Nome), System.DBNull.Value, umFun-
cionario.Nome))

        aux_command.Parameters.AddWithValue("@Morada",
IIf(String.IsNullOrEmpty(umFuncionario.Morada), System.DBNull.Value, um-
Funcionario.Morada))

        aux_command.Parameters.AddWithValue("@CodigoPostal",
IIf(String.IsNullOrEmpty(umFuncionario.CodigoPostal), Sys-
tem.DBNull.Value, umFuncionario.CodigoPostal))

        aux_command.Parameters.AddWithValue("@Localidade",
IIf(String.IsNullOrEmpty(umFuncionario.Localidade), System.DBNull.Value,
umFuncionario.Localidade))

        aux_command.Parameters.AddWithValue("@Pais",
IIf(String.IsNullOrEmpty(umFuncionario.Pais), System.DBNull.Value, umFun-
cionario.Pais))

        aux_command.Parameters.AddWithValue("@Telefone",
IIf(String.IsNullOrEmpty(umFuncionario.Telefone), System.DBNull.Value,
umFuncionario.Telefone))

        aux_command.Parameters.AddWithValue("@Telemovel",
IIf(String.IsNullOrEmpty(umFuncionario.Telemovel), System.DBNull.Value,
umFuncionario.Telemovel))

        aux_command.Parameters.AddWithValue("@Email",
IIf(String.IsNullOrEmpty(umFuncionario.Email), System.DBNull.Value, um-
Funcionario.Email))

        aux_command.Parameters.AddWithValue("@WEB",
IIf(String.IsNullOrEmpty(umFuncionario.WEB), System.DBNull.Value, umFun-
cionario.WEB))

        DALConn.ExecuteNonQuery(aux_command)

        dalFuncionarios_Custo_Hora.Inserir_Lista(umFuncionario.CustoHora,
umFuncionario.Funcionario)

        If iniciar Then DALConn.Commit_Transaction()

    Catch ex As Exception

        If iniciar Then DALConn.Rollback_Transaction()

        Throw New Exception(ex.Message)

    End Try

End Sub
```

No procedimento de “alterar” verifica-se situação idêntica, foi necessário criar uma transação e depois de alterar os dados referentes à ficha do funcionário são alterados os dados referentes ao custo/hora do mesmo, enquanto que no procedimento “eliminar” não existe essa necessidade visto que o SGBD gera a relação entre as tabelas e, ao eliminar um funcionário, são eliminados os custos/hora referentes a esse mesmo funcionário. Ao carregar a ficha do funcionário são também carregados os dados dos Custos/Hora. Na listagem de funcionários não é carregada essa informação.

Este comportamento repete-se sempre que exista uma ligação 1..N entre tabelas. Na Figura 6.4 estão representadas as entidades com este comportamento.

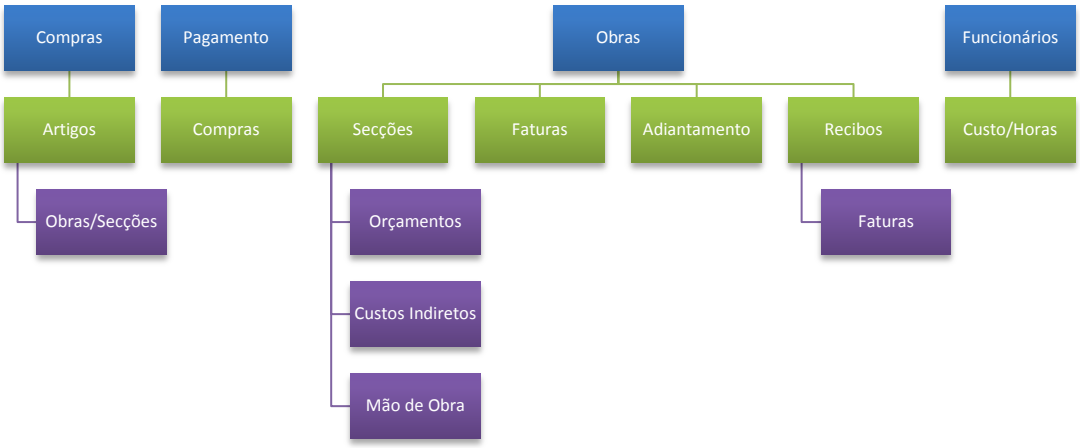


Figura 6.4: Relação entre entidades

O formulário de gestão de compras (Figura 6.5) permite registar as compras efetuadas aos fornecedores e distribuir os artigos adquiridos pelas diversas obras.

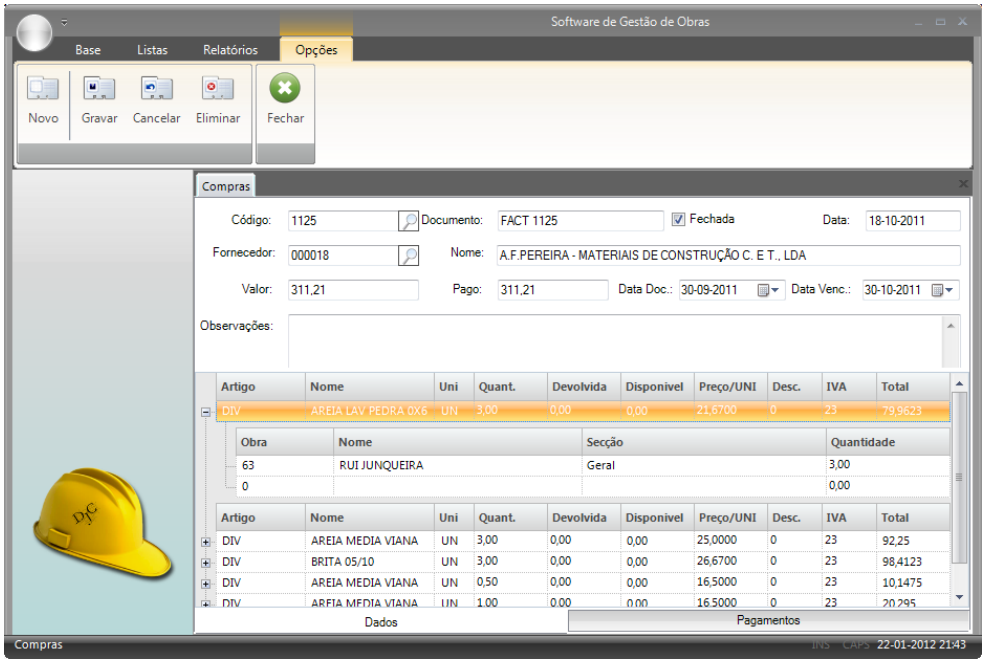


Figura 6.5: Formulário de compras de material

Neste formulário é necessário identificar qual o fornecedor a que foi efetuada a compra, a data do documento e a data de vencimento do mesmo. Caso seja necessário, é possível escrever alguma anotação na compra e qual o documento do fornecedor a que se refere a compras (N.º Fatura, N.º Guia de Transporte, etc.). Depois de criado o cabeçalho do documento é necessário especificar os artigos adquiridos. Na *DataGrid* é necessário especificar o código do artigo e, caso seja necessário, é possível alterar a descrição, a unidade de compra, a quantidade adquirida, a quantidade de mercadoria devolvida, o preço unitário, a percentagem de desconto e a taxa de IVA. Para cada uma das linhas de artigos é necessário especificar a obra, a respetiva secção a que se destina e a quantidade de material. Neste formulário é possível visualizar o valor total da compra e o valor já pago ao fornecedor. Este formulário possui ainda um separador ***Pagamentos*** que permite ver os documentos de pagamento que liquidaram o documento de compra.

O formulário de gestão de adiantamento a fornecedores (Figura 6.6) permite registar os adiantamentos efetuados aos fornecedores.

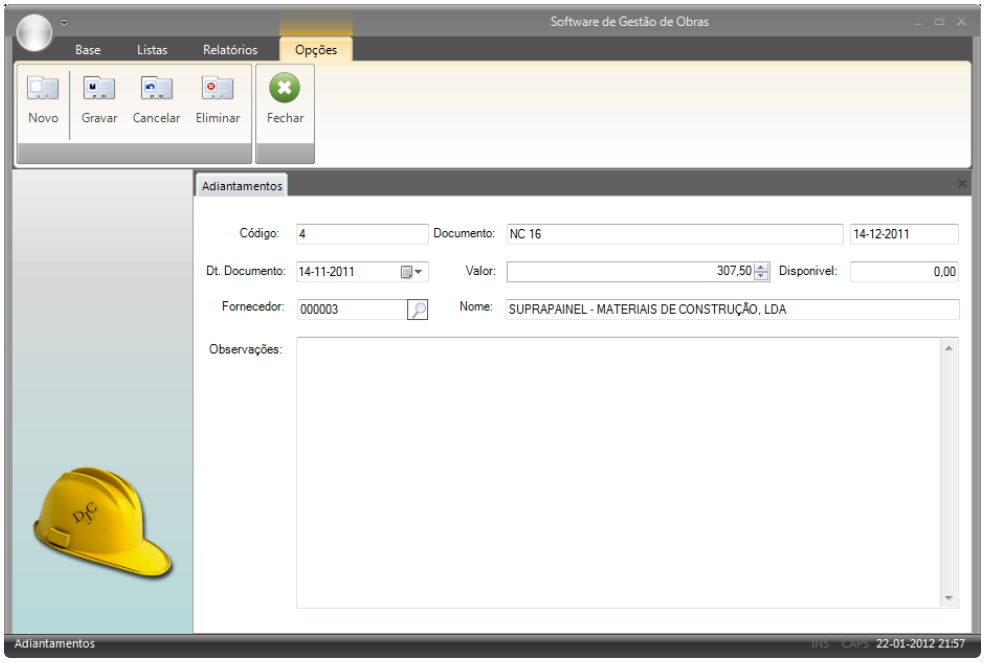


Figura 6.6: Formulário de adiantamento a fornecedores

Neste formulário é necessário especificar o fornecedor para o qual se vai efetuar um adiantamento e o valor. É também possível visualizar o valor ainda disponível. Caso seja necessário é possível especificar um documento usado na transação e uma observação.

No formulário de gestão de pagamento (Figura 6.7) a fornecedores são registados os pagamentos aos fornecedores. Ao escolher um fornecedor surgem na *DataGrid* todas as compras ainda por liquidar. Depois, é necessário especificar o valor que será pago ao fornecedor e distribuí-lo pelas várias compras. O valor tem de ser afeto na totalidade. Caso exista um adiantamento é possível especificar o valor a usar deste, podendo ser a totalidade ou parte dele. É ainda possível especificar o número do documento do programa da faturação que comprova este movimento e uma observação.

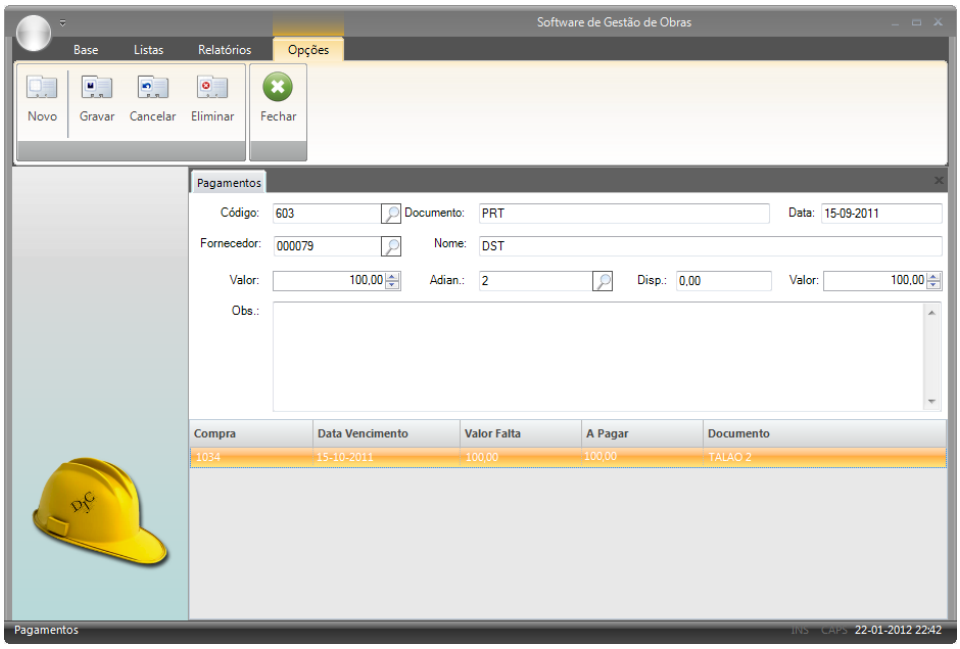


Figura 6.7: Formulário de pagamentos a fornecedores

O formulário de gestão de obras (Figura 6.8) é o mais complexo de todos. Aqui é possível gerir todos os dados referentes a obras. Para facilitar a implementação de todas as funcionalidades optou-se pela criação de *usercontrols* para cada uma das opções.

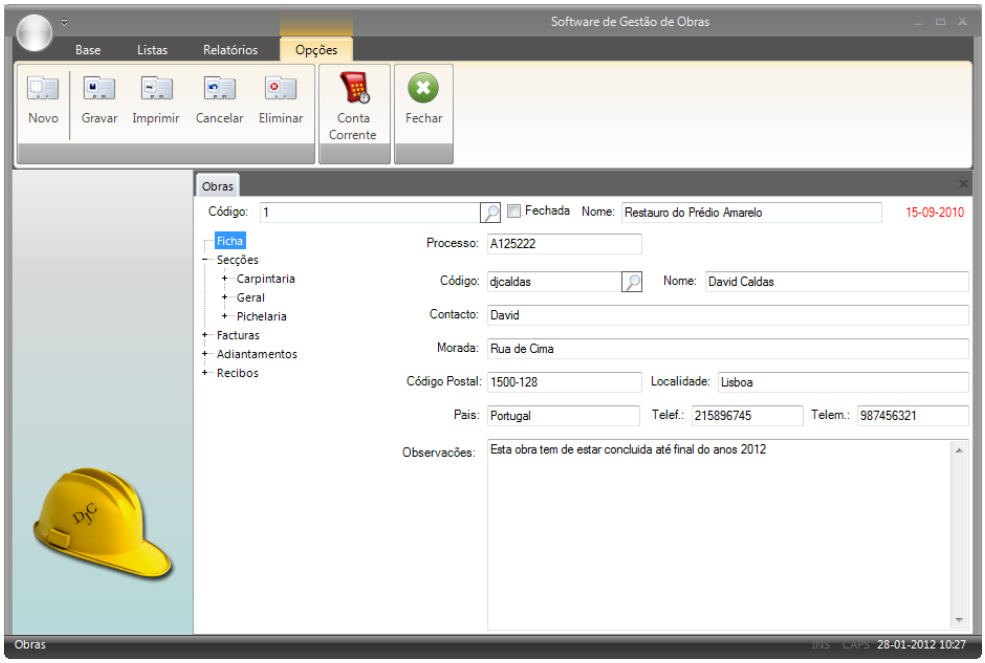


Figura 6.8: Formulário de gestão de obras

Assim, foi necessário criar *usercontrols* para a manipulação e análise dos dados. O *usercontrol* é carregado no lado direito do formulário segundo a opção escolhida na *treeview* que se encontra no lado esquerdo do formulário. Sempre que a opção **Ficha** é escolhida carrega-se o *usercontrol* **ucpDados**, responsável por gerir a ficha da obra, nomeadamente no que respeita ao cliente a que pertence a obra, nome e local da obra, número de processo e observações referentes à obra.

Ao escolher a opção **Secções** é carregado o *usercontrol* **ucpAnalisa_Seccoes**. Aqui é possível saber quanto foi gasto em cada uma das secções, sendo por isso um *usercontrol* de análise de dados. Todos os *usercontrols* de análise de dados são constituídos por dois painéis, um com um gráfico (Figura 6.9) e outro com uma *DataGrid* (Figura 6.10). A *DataGrid* permite analisar os valores que deram origem ao gráfico.

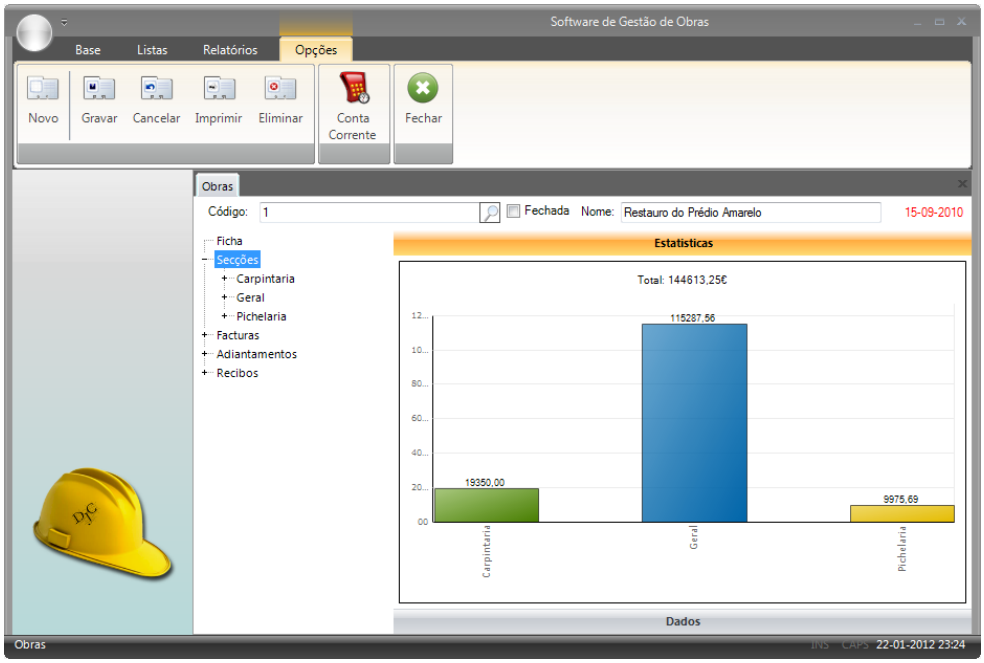


Figura 6.9: Gráfico de análise de custos das secções

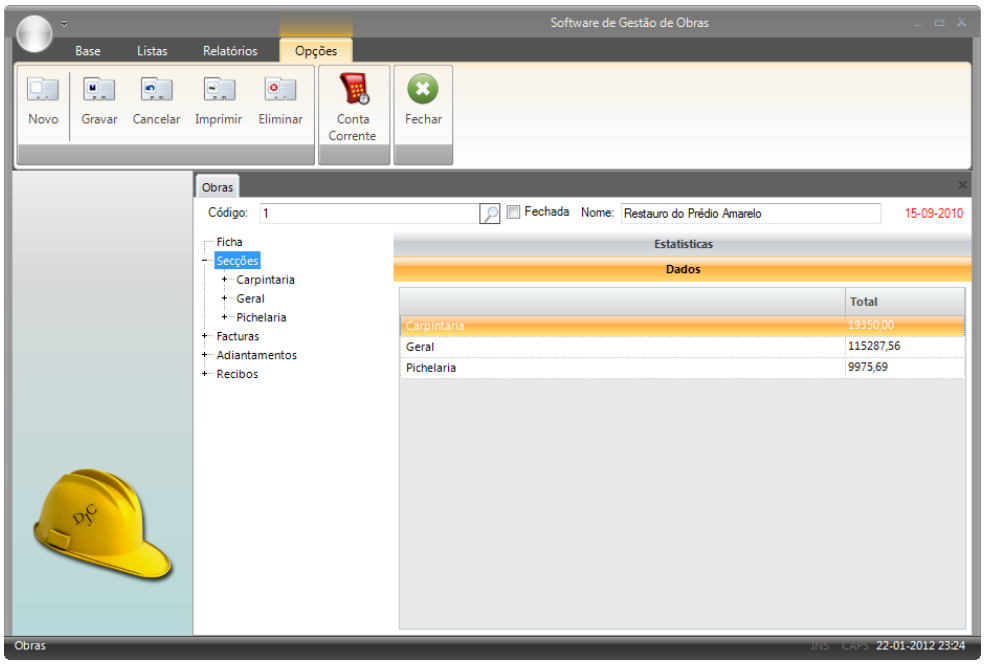


Figura 6.10: *DataGrid* de análise de custos das secções

Ao selecionar esta opção com o botão do lado direito do rato surge um menu de contexto que permite inserir uma nova secção.

Escolhendo uma das secções (exemplo: Carpintaria, na Figura 6.11) é carregado o *usercontrol* que permite analisar a secção, onde é possível ver o que foi gasto em compras, mão-de-obra e custos indiretos, e ainda comparar com os valores orçamentados para a secção. Para além dos

dois painéis já referidos, existe ainda um terceiro painel onde é possível escrever uma observação em relação à secção.

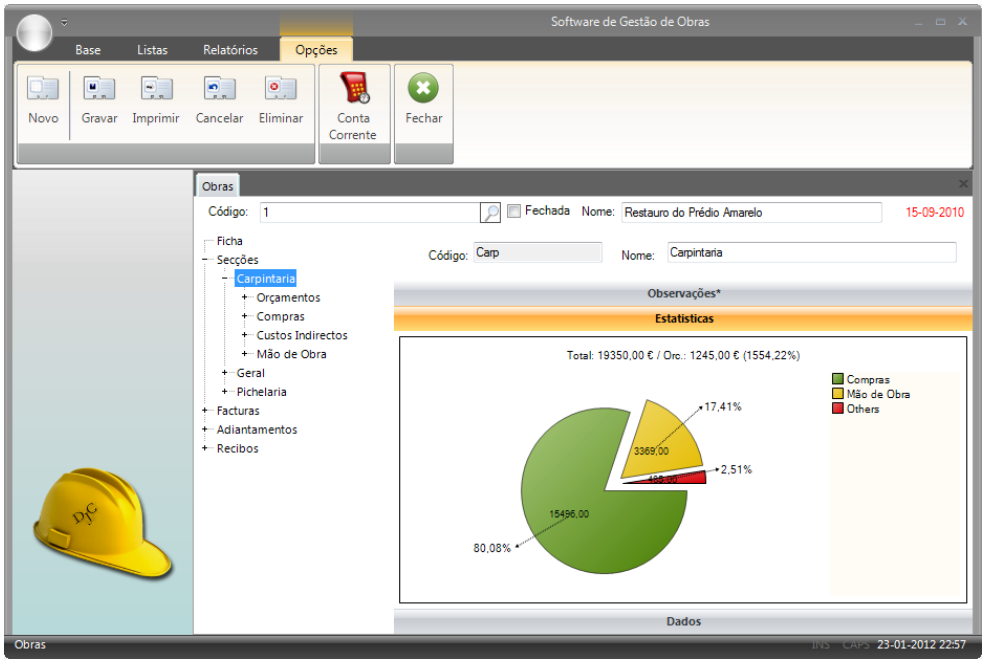


Figura 6.11: Gráfico de análise de uma secção

Ao escolher a opção **Orçamentos** (Figura 6.12) em qualquer uma das secções é carregado o *usercontrol* que permite gerir os orçamentos. Este *usercontrol* é constituído por uma *DataGrid* onde são colocados os dados.

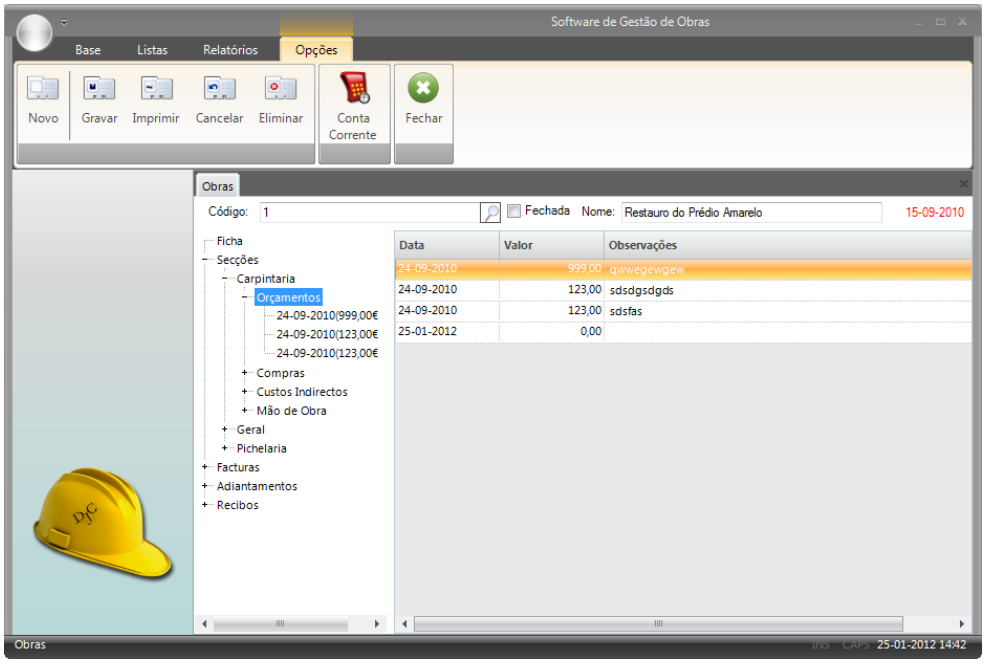


Figura 6.12: *Usercontrol* de gestão de orçamentos

Na opção de orçamento existe ainda a possibilidade de abrir um orçamento específico (Figura 6.13).

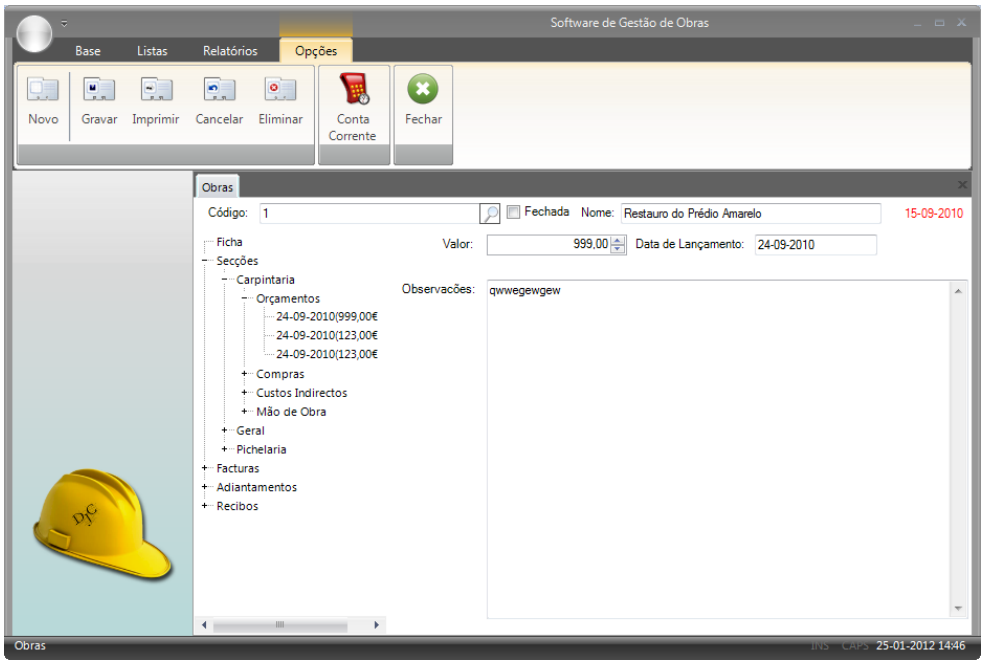


Figura 6.13: Ficha de um orçamento

Na opção **Compras** (Figura 6.14) é possível analisar quanto foi gasto nesta secção por fornecedor.

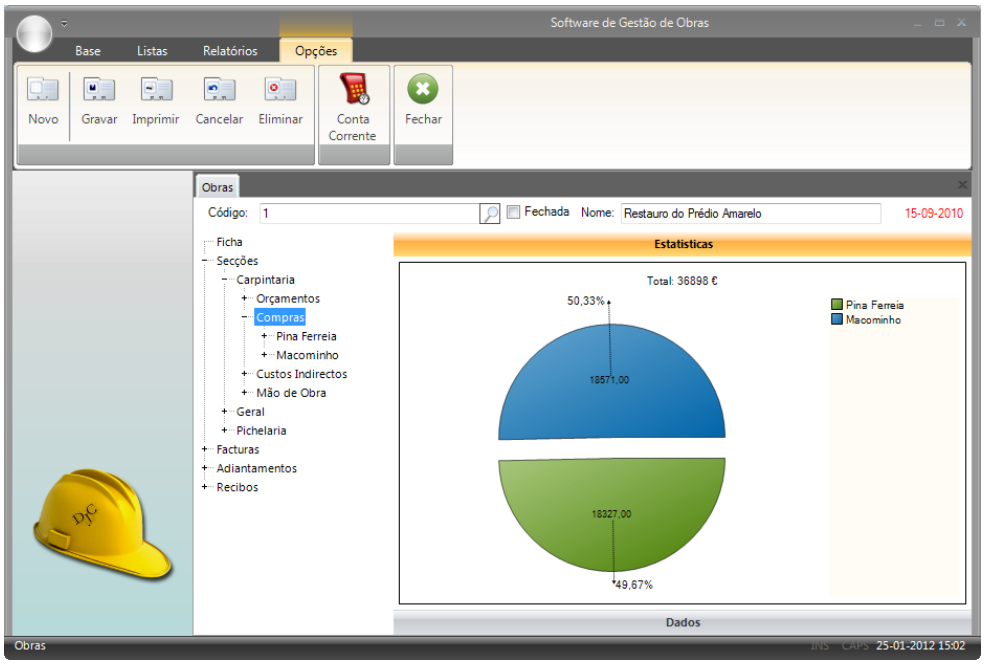


Figura 6.14: Análise de gastos por fornecedor

Ao escolher um dos fornecedores, é possível analisar quanto se gastou em cada um dos artigos adquiridos a este (Figura 6.15).

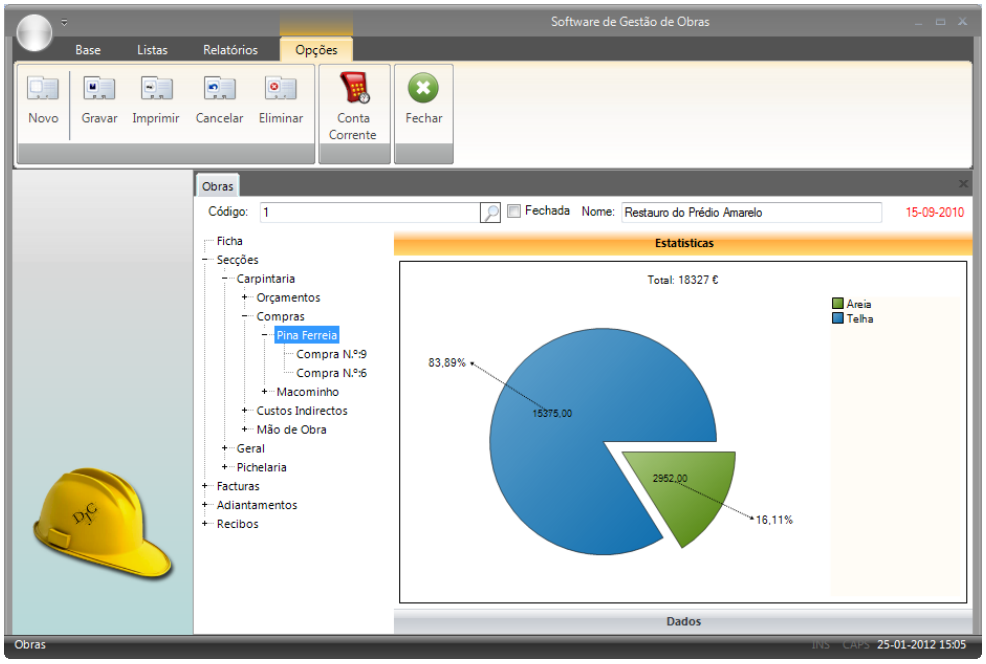


Figura 6.15: Análise de custos por artigo

Dentro da opção dos fornecedores é possível escolher o documento referente à aquisição de artigos e visualizar em detalhe a compra (Figura 6.16).

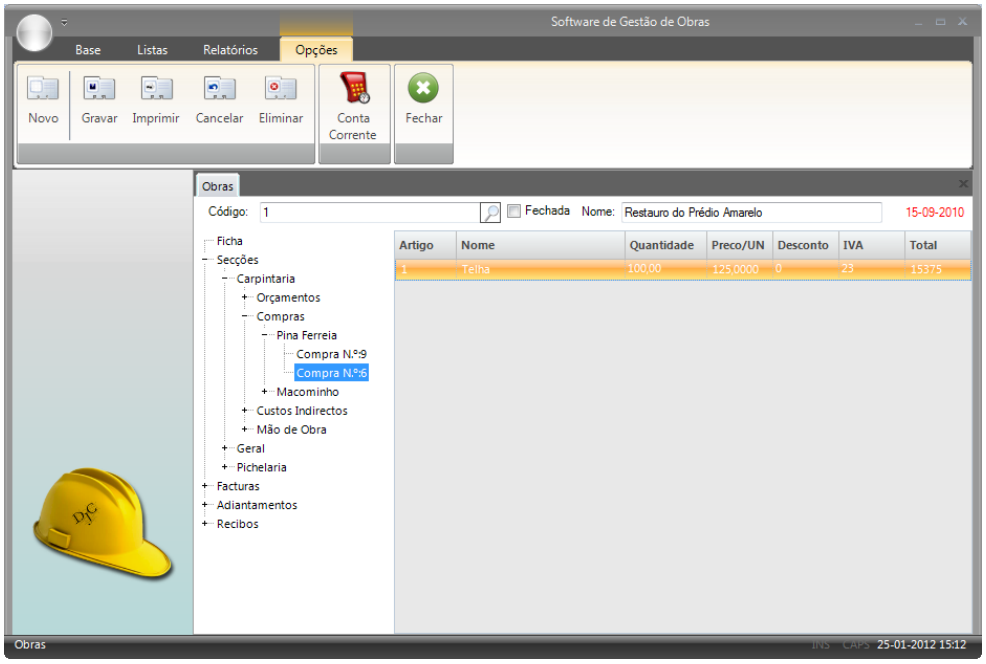


Figura 6.16: Análise das linhas do documento de compra

Ao escolher a opção **Custos Indiretos** (Figura 6.17) é possível analisar quanto foi gasto em cada tipo de custo.

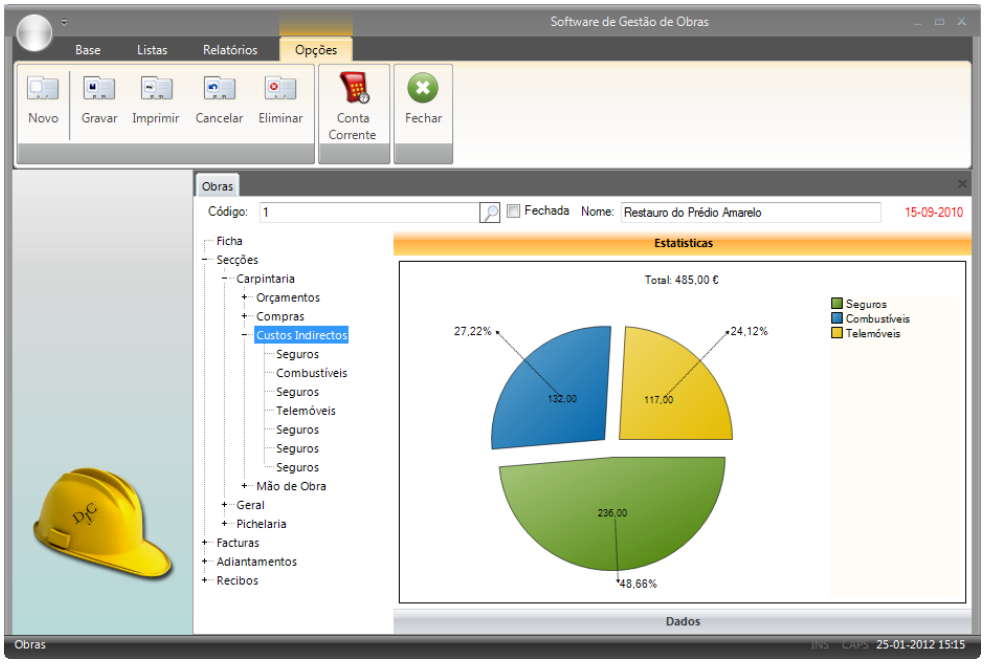


Figura 6.17: Análise de gastos por custos indiretos

Clicando com o botão do lado direito sobre a opção escolhida (Custos Indiretos) permite registar um novo custo indireto, sendo carregado o *usercontrol* **ucpCustos_Indirectos** (Figura 6.18). Ao clicar num gasto é possível visualizá-lo e alterá-lo. Este comportamento repete-se nas opções Mão-de-obra, Faturas, Adiantamento e Recibos.

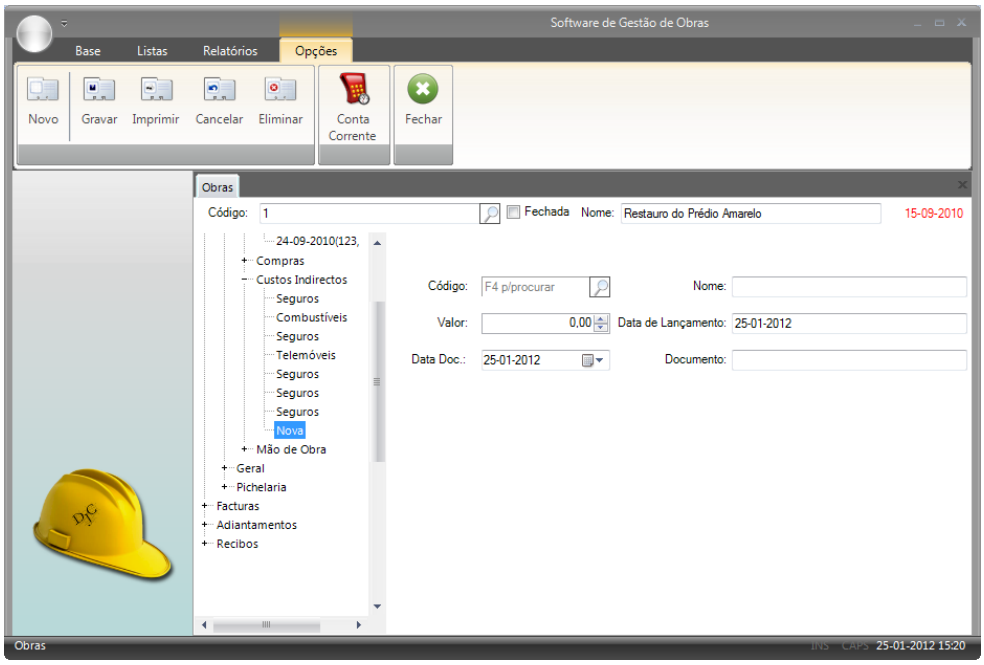


Figura 6.18: Registo de custos indiretos

Ao escolher a opção **Mão-de-obra** (Figura 6.19) é possível analisar o total gasto por funcionário em horas e em custos.

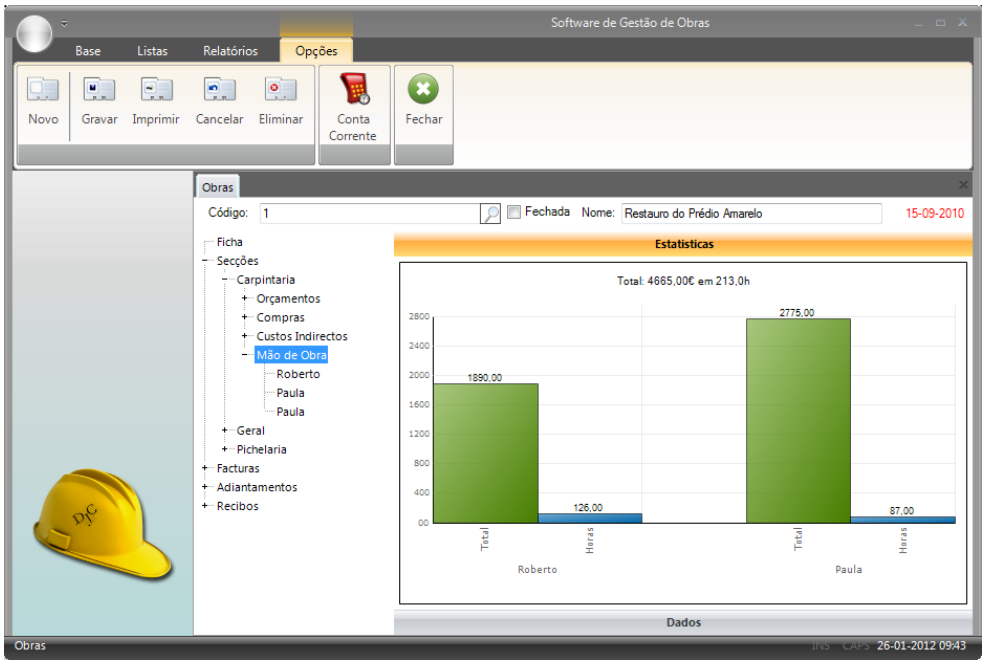


Figura 6.19: Análise da mão-de-obra

Ao criar um registo de mão-de-obra (Figura 6.20) é possível escolher o funcionário e, para este, qual o custo/hora que será utilizado e quantas horas trabalhou. Este custo/hora é de referência sendo possível alterá-lo em cada registo.

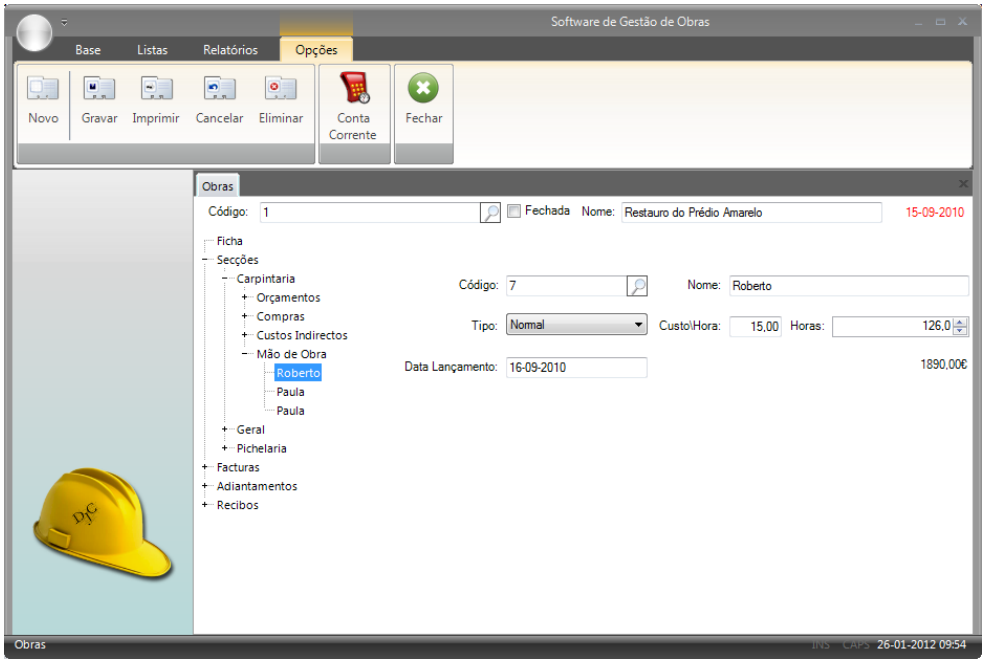


Figura 6.20: Registo de mão-de-obra

Na opção **Fatura** (Figura 6.21) é possível analisar quanto foi faturado ao cliente e quanto foi pago por este.

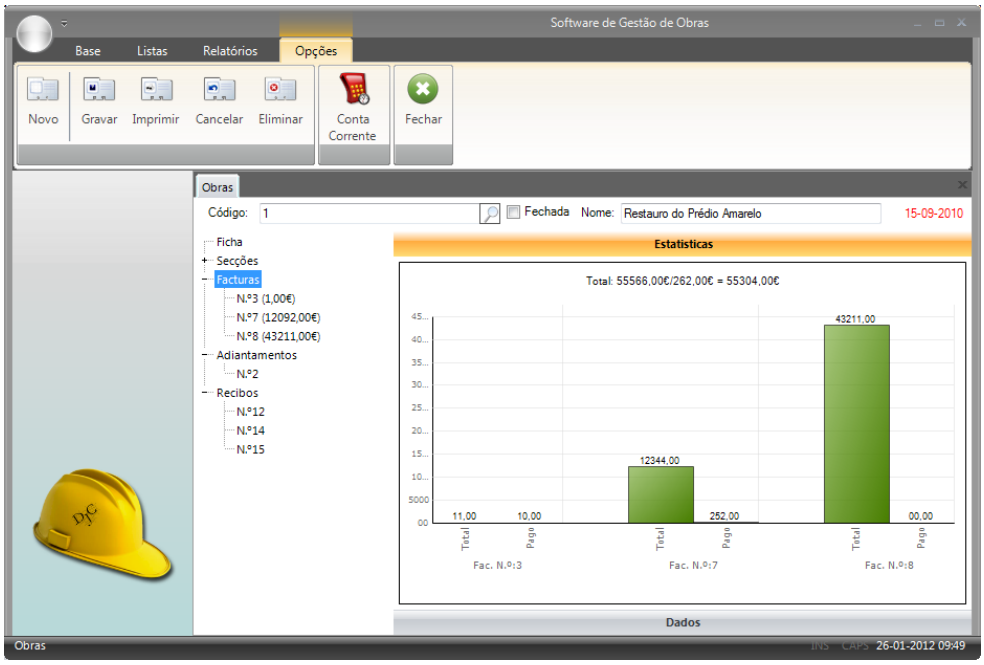


Figura 6.21: Análise dos valores faturados

Ao criar um registo de Fatura (Figura 6.22) é possível especificar o valor total, o valor do IVA, a data do documento e de vencimento do mesmo. Ao visualizar um registo já criado é possível ver quanto já foi pago desta fatura e quanto falta pagar.

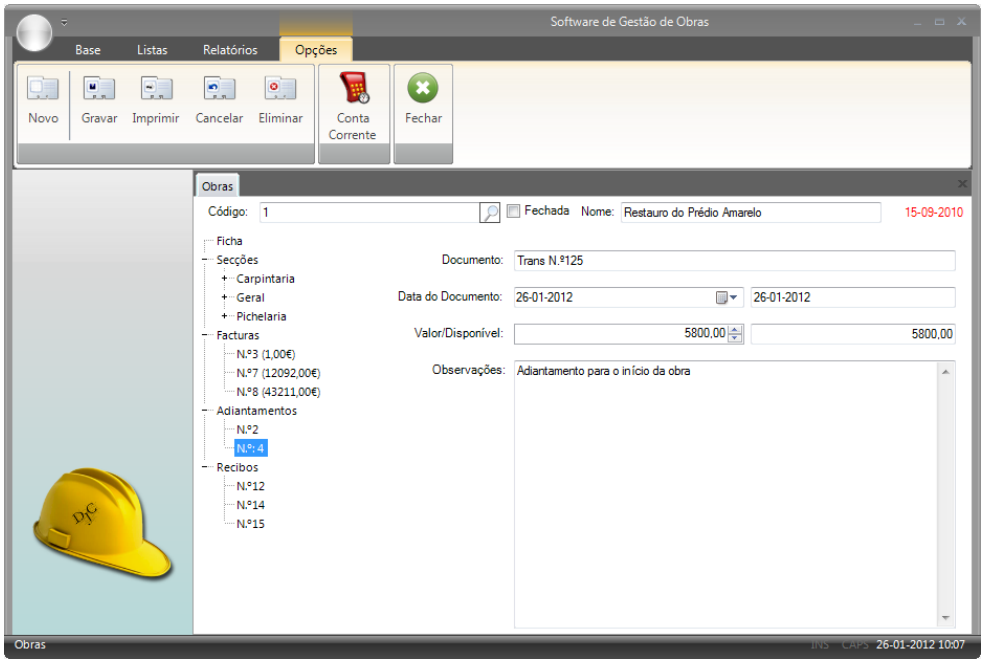


Figura 6.22: Registo de fatura

Na opção **Adiantamento** (Figura 6.23) é possível saber quanto foi adiantado pelo cliente e quanto desse valor está ainda disponível.

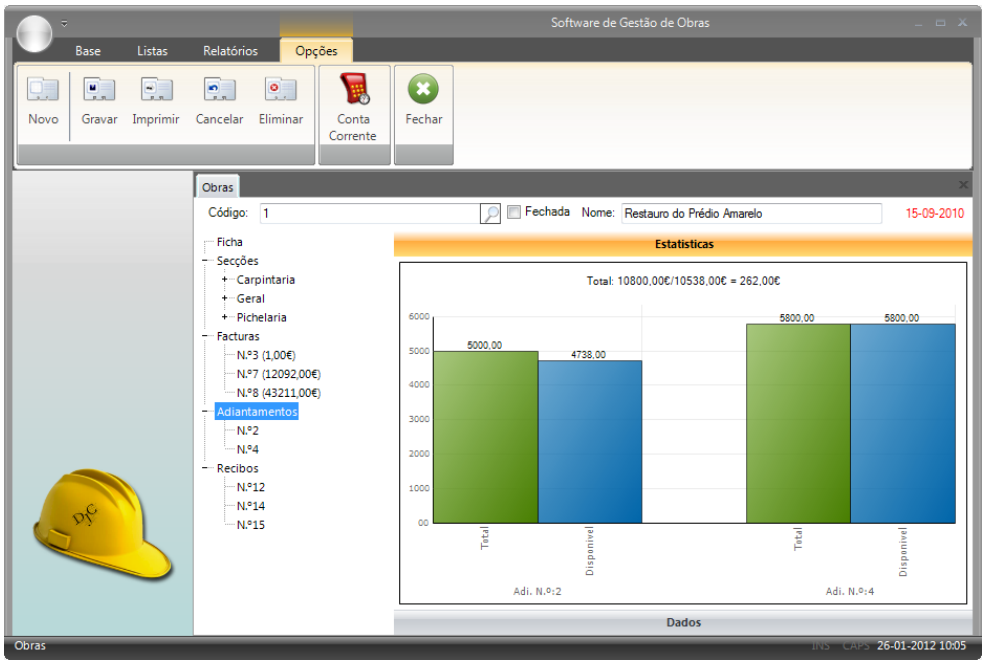


Figura 6.23: Análise dos adiantamentos

Ao criar um registo de Adiantamento (Figura 6.24) é possível especificar o número do documento, data do mesmo e valor. Ao visualizar a ficha é possível saber quanto do valor está ainda disponível.

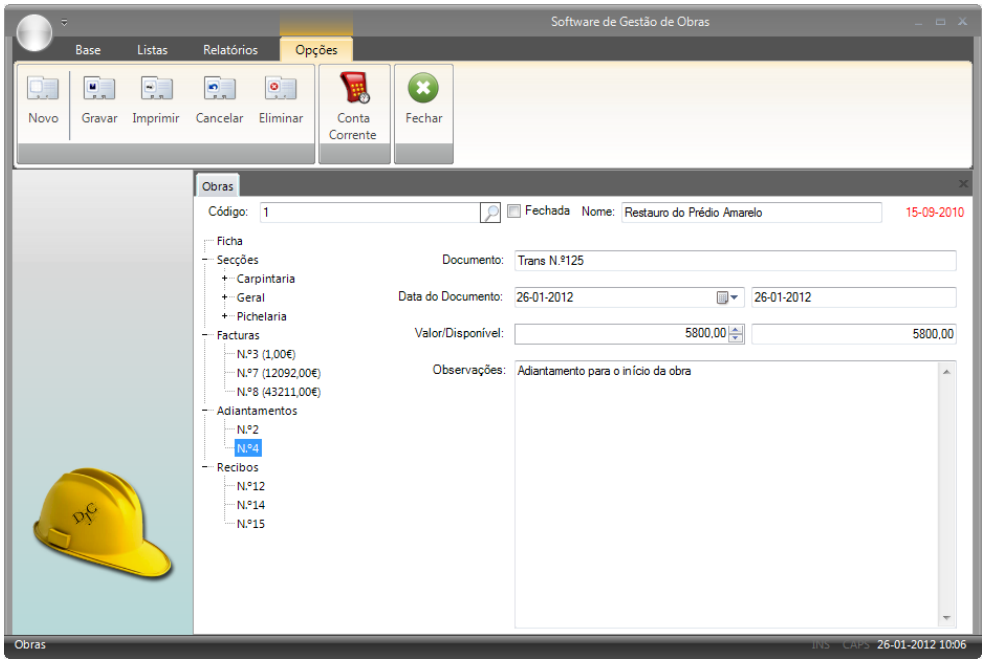


Figura 6.24: Registo de adiantamentos

Na opção **Recibos** (Figura 6.25) é possível analisar a relação entre o que foi pago pelo cliente por adiantamento ou mediante solicitação da construtora.

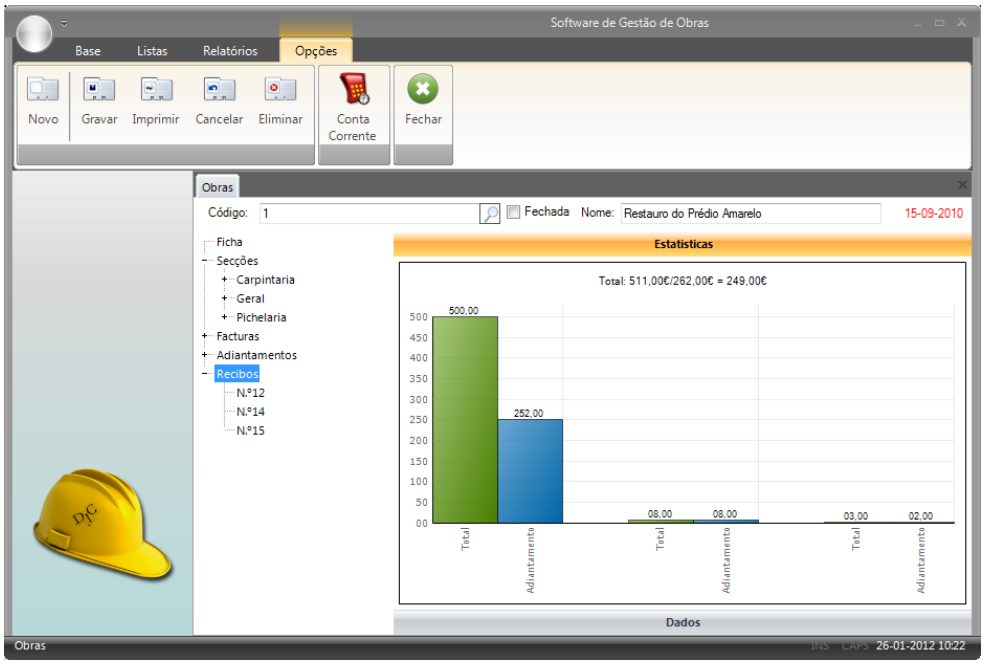


Figura 6.25: Análise dos recibos

Ao criar um Recibo (Figura 6.26) é possível especificar o valor deste, a data do documento e na *DataGrid* aparecem quais as faturas ainda por liquidar. O valor do recibo pode ser repartido por uma ou mais faturas. No caso de haver adiantamento é possível usá-lo, bastando para isso introduzir o código e especificar o valor que será utilizado.

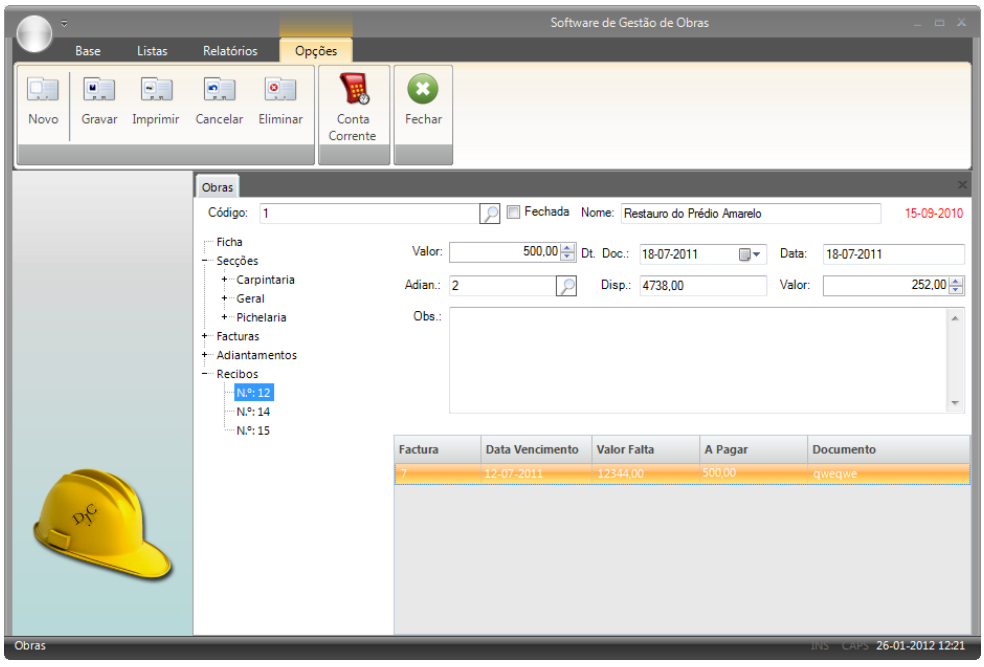
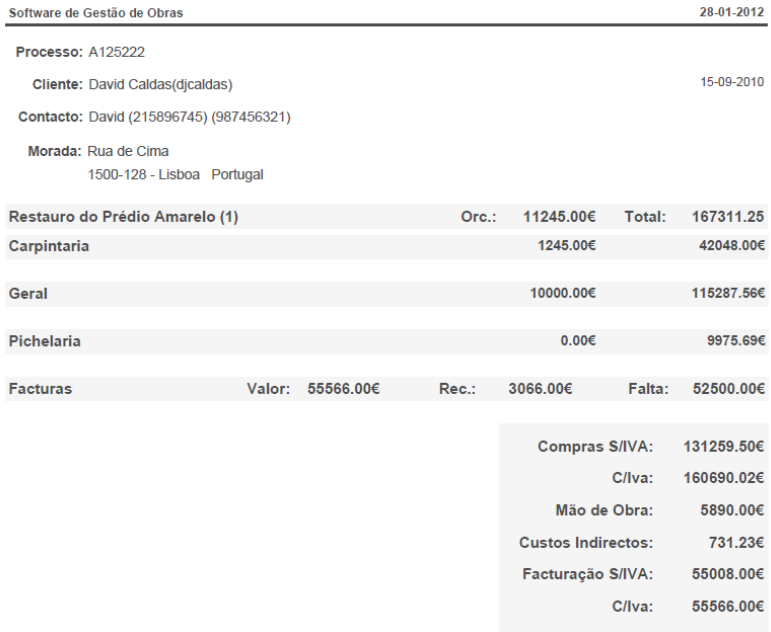


Figura 6.26: Registo de recibo

Neste formulário é ainda possível imprimir o relatório de obra que lista todos os dados referentes aos custos da obra (Figura 6.27).



Software de Gestão de Obras 28-01-2012

Processo: A125222

Cliente: David Caldas(djcaldas) 15-09-2010

Contacto: David (215896745) (987456321)

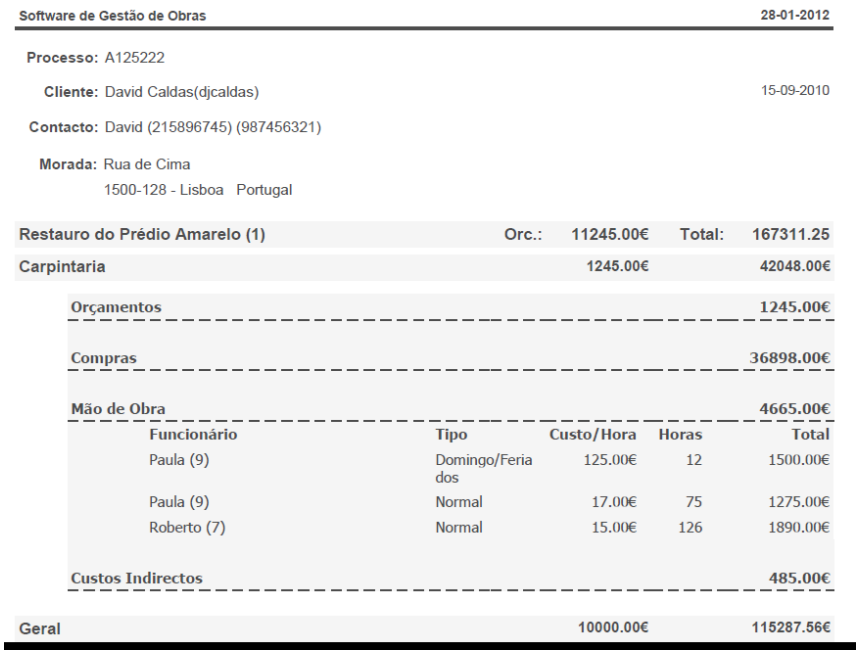
Morada: Rua de Cima
1500-128 - Lisboa Portugal

Restauro do Prédio Amarelo (1)			Orc.:	11245.00€	Total: 167311.25
Carpintaria				1245.00€	42048.00€
Geral				10000.00€	115287.56€
Pichelaria				0.00€	9975.69€
Facturas	Valor:	55566.00€	Rec.:	3066.00€	Falta: 52500.00€

Compras S/IVA:	131259.50€
C/Iva:	160690.02€
Mão de Obra:	5890.00€
Custos Indirectos:	731.23€
Facturação S/IVA:	55008.00€
C/Iva:	55566.00€

Figura 6.27: Relatório da obra

Inicialmente, aparecem os dados resumidos por secção sendo possível expandir cada uma delas para ver mais detalhes (Figura 6.28).



Software de Gestão de Obras 28-01-2012

Processo: A125222

Cliente: David Caldas(djcaldas) 15-09-2010

Contacto: David (215896745) (987456321)

Morada: Rua de Cima
1500-128 - Lisboa Portugal

Restauro do Prédio Amarelo (1)			Orc.:	11245.00€	Total: 167311.25
Carpintaria				1245.00€	42048.00€
Orçamentos					1245.00€
Compras					36898.00€
Mão de Obra					4665.00€

Funcionário	Tipo	Custo/Hora	Horas	Total
Paula (9)	Domingo/Feriados	125.00€	12	1500.00€
Paula (9)	Normal	17.00€	75	1275.00€
Roberto (7)	Normal	15.00€	126	1890.00€

Custos Indirectos		485.00€
Geral	10000.00€	115287.56€

Figura 6.28: Detalhes do relatório da obra

Exista ainda uma opção no formulário para visualizar a **Conta Corrente** da obra (Figura 6.29) onde é possível analisar a relação entre faturas emitidas e os recibos.

Software de Gestão de Obras 28-01-2012

Processo: A125222

Cliente: David Caldas(djcaldas) 15-09-2010

Contacto: David (215896745) (987456321)

Morada: Rua de Cima
1500-128 - Lisboa Portugal

Documentos		Data	Valor	Saldo
3	Fact 1º Pagamento	30-09-2010	-11.00€	-11.00€
14	Adiantamento N.º2	13-09-2011	8.00€	-3.00€
15	Em dinheiro + Ad N.º2	13-09-2011	3.00€	0.00€
7	Fact 2º Pagamento	16-06-2011	-12,344.00€	-12,344.00€
12	Cheque Nº125 + Ad N.º2	18-07-2011	500.00€	-11,844.00€
16	Cheque Nº14444 CGD	26-01-2012	1,000.00€	-10,844.00€
8	Fact 3º Pagamento	16-06-2011	-43,211.00€	-54,055.00€
16	Cheque Nº14444 CGD	26-01-2012	1,555.00€	-52,500.00€

Adiantamentos		Data	Valor	Saldo
2	qqweqwe	30-06-2011	4,738.00€	-47,762.00€
4	Trans N.º125	26-01-2012	5,800.00€	-41,962.00€

Figura 6.29: Conta Corrente da obra

Todos os formulários apresentados anteriormente são acedidos a partir do separador Base (Figura 6.30) no menu do formulário principal.

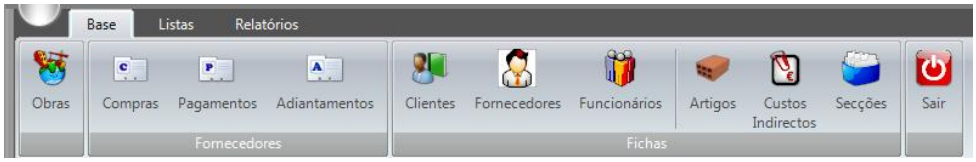


Figura 6.30: Menu Base

No separador Listas (Figura 6.31) foram acrescentadas as listagens das opções anteriores mais a listagem das faturas em aberto, das compras em aberto e do registo da mão-de-obra nas obras.

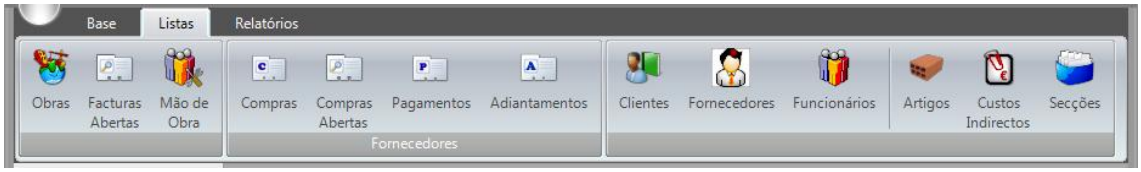


Figura 6.31: Menu Listas

Foi acrescentado no menu um novo separador (Relatórios) onde foram criadas duas opções:

- Compras abertas/fornecedor: permite ver as compras que estão em aberto, por fornecedor;
- Faturas abertas/cliente: permite ver as faturas em aberto, por cliente.

Na figura seguinte (Figura 6.32) está representado o relatório de compras em aberto por fornecedor. Este é inicialmente carregado com todas as compras em aberto de todos os fornecedores mas é possível filtrar os dados por um fornecedor específico.

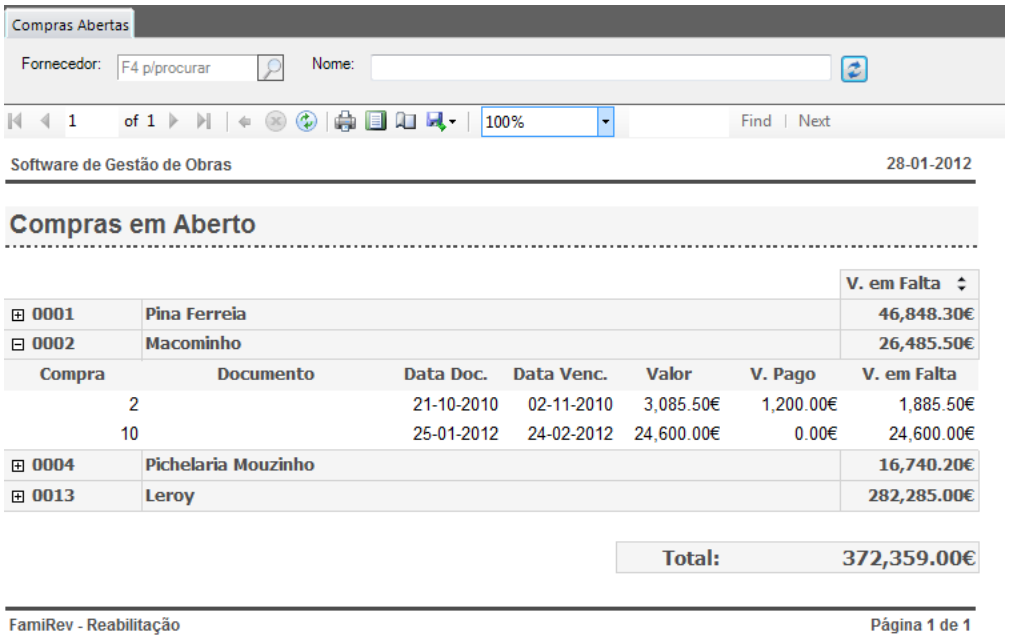


Figura 6.32: Relatório de compras em aberto

Capítulo 7 - **Conclusão**

Sempre que se desenvolve um projeto, seja ou não num contexto informático, existe, sempre uma obtenção de novos conhecimentos que nos fazem evoluir quer como pessoa, ou como neste caso particular, como Informáticos/Analistas/Programadores. A realização deste projeto veio impulsionar a obtenção de conhecimentos importantes no desenvolvimento de *software*. Relativamente ao desenvolvimento desta aplicação de gestão e análise de custos de obra, esta permitiu-me evoluir enquanto analista e programador, visto ter sido necessário modelar e implementar informaticamente um modelo de negócio de raiz. A criação da *framework* foi o que mais impulsionou a evolução como analista e programador, uma vez que com ela tive a necessidade de:

- Esquematizar a relação entre as várias camadas que constituem uma aplicação;
- Uniformizar as operações CRUD;
- Criar uma aplicação que a partir da estrutura da base de dados cria o código das classes das camadas DTO,DAL e BLL;
- Criar uma relação entre formulários de manutenção e de listagem e deste com a camada de controlo de listas;
- Criar *templates* de projetos e de formulário.

Quantos aos objetivos propostos para este projeto, este foram plenamente atingidos, encontrando-se a aplicação a ser utilizada na empresa e dando provas da sua utilidade, conforme se exige a um projeto aplicado. Durante o período de utilização, esta tem sido uma ferramenta útil na análise de custos das obras, bem como na orçamentação de novas empreitadas. Pode-se assim afirmar, com toda a satisfação, que os objetivos foram alcançados com sucesso.

Capítulo 8 - **Trabalho Futuro**

O desenvolvimento de uma *framework* dificilmente pode ser dado por concluído. Existem frequentemente alterações a implementar, quer digam respeito à otimização de desempenho ou à criação de novas funcionalidades que surgem com a utilização regular da *framework* em novos projetos.

No que se refere à aplicação de Gestão de obras aqui apresentada, e de modo a ser possível tirar um maior partido, quer da própria aplicação quer da estrutura criada, é necessário a criação de mais relatórios tendo em vista a análise de outras variáveis do negócio.

Referências

.NET Framework 4. (2011). *.NET Framework Conceptual Overview*. Obtido em 13 de 09 de 2011, de MSDN – Explore Windows, Web, Cloud, and Windows Phone Software Development: http://msdn.microsoft.com/library/zw4w595w.aspx

Builder, R. (2012). *Introdução ao Report Builder 3.0*. Retrieved 01 11, 2012, from http://technet.microsoft.com/pt-pt/library/dd220460.aspx

Controls, W. F. (2011). *Windows Forms Controls - NetAdvantage for Windows Forms - User Experience Controls and Components*. Retrieved 04 15, 2011, from http://www.infragistics.com/dotnet/netadvantage/winforms.aspx

Express, V. B. (2012). *Visual Basic 2010 Express*. Retrieved 01 06, 2012, from Microsoft Visual Studio: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-basic-express

Improxy. (2011). *Improxy - Software gestão obras*. Retrieved 11 25, 2011, from http://www.improxy.pt/Produtos/Gest%C3%A3odeobras/tabid/1027/Default.aspx

LiveSolutions. (2011). *LiveSolutions - Software de Gestao para Internet*. Retrieved 11 25, 2011, from http://www.livesolutions.pt/?m=zMarketObras

Magnisoft. (2011). *Software para Gestão de Obras e Orçamentação - Magnisoft OranGest OBRAS*. Retrieved 11 25, 2011, from http://www.magnisoft.pt/OranGest-OBRAS.aspx

N-Tier. (2012). *N-Tier Data Applications Overview*. Retrieved 01 15, 2012, from http://msdn.microsoft.com/library/bb384398.aspx

PRIMAVERA BSS. (2011). *PRIMAVERA BSS Portugal - Powered by PRIMAVERA WebCentral*. Retrieved 11 25, 2011, from http://www.primaverabss.com/pt/PortalRender.aspx?PageID=ed54225c-d823-4c93-bfdb-41925a0db024

Service, S. R. (2011). *SQL Reporting Services | Business Intelligence | Microsoft SQL Server*. Retrieved 12 28, 2011, from http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/reporting-services.aspx

Silva, A., & Videira, C. (2001). *UML, Metodologias e Ferramentas CASE*. Centro Atlântico.

Sommerville, I. (2007). *Engenharia de Software*. Pearson Addison-Wesley.

Sql Server. (2012). *Free Database Software | Database Applications | SQL Server Express*. Retrieved 01 06, 2012, from Microsoft SQL Server Express Edition: http://www.microsoft.com/sqlserver/en/us/editions/express.aspx