Departamento de Ciências da Comunicação, Artes e Tecnologias de Informação

BCodeGen

Ferramenta de Geração de Código para Aplicações Web

Bruno Cipriano

(N.º 2300510)

Relatório da disciplina de Projecto da Licenciatura em Informática

Orientador: Alexandre Pereira

Novembro de 2007

Resumo

Nos projectos típicos de desenvolvimento para a Internet existe um conjunto de operações e funcionalidades comuns que têm de ser implementadas para servirem de base para o desenvolvimento do resto do projecto. Estas operações e funcionalidades estão relacionadas com a criação e manutenção de dados, tipicamente guardados em Sistemas de Gestão de Bases de Dados Relacionais e, são conhecidas como operações CRUD (Create, Retrieve, Update e Delete). Neste tipo de projectos também é típica a criação de um conjunto de interfaces que permitem que utilizadores não-técnicos possam fazer a visualização e a manipulação dos dados .

O objectivo deste projecto é construir uma ferramenta que permita automatizar o desenvolvimento das operações e interfaces referidas, partindo de uma descrição da base de dados e das interfaces desejadas, cuja consequência esperada era a redução do tempo necessário para começar o desenvolvimento dos projectos referidos.

Para alcançar estes objectivos foi desenvolvida uma aplicação ("BCodeGen") capaz de analisar a estrutura de uma base de dados e outro input do utilizador, para criar o código para as operações e interfaces descritas anteriormente. A aplicação permite, também, a integração na base de código gerado de mudanças feitas à estrutura da Base de Dados, fornecendo uma estrutura para que sejam mantidas eventuais adições manuais, o que é útil em projectos nos quais os requisitos a nível de dados mudam a seguir à fase inicial de desenho da Base de Dados.

Para suportar o código criado pela aplicação "BCodeGen" foi também criada uma biblioteca de classes que disponibiliza funcionalidades úteis para o desenvolvimento de aplicações Web. Esta biblioteca pode ser usada como um produto individual, pois não depende de qualquer forma da aplicação de geração de código.

Palavras-Chave:

geração de código, desenvolvimento Web, automatização, Modelo Relacional, Object Relational Mapping

Abstract

In typical web-development projects there are common sets of operations and functionalities that must be developed to serve as a basis for the development of the rest of project. These operations and functionalities are related to the creation and maintenance of data, typically stored in Relational Data Base Management Systems and are known as CRUD operations (Create, Retrieve, Update and Delete). Also typical in these types of projects, is the creation of a set of interfaces that allow the visualization and manipulation of the data by non-technical users.

The objective of this project is to create a tool that allows the automation of the development of these sets of operations and interfaces, from a description of the database and from a description of the desired interfaces. The expected consequence of this was a reduction of the time required to start developing those types of projects.

In order to achieve these goals a software application ("BCodeGen") capable of analyzing a database structure and other user input, to create the code for the operations and interfaces described early, was developed. The application also allows the quick integration of changes made to the database structure into the code base, while offering a structure that allows certain manual additions made to the code to be maintained, which is useful in projects where the data requirements change after the initial design phase.

To support the code created by "BCodeGen", a code library consisting of a set of classes that provide useful functionalities for web-development projects was also developed. This code library can be used as a standalone product, because it is not dependent on the code generation application.

Keywords:

code generation, web development, automation, Relational Model, Object Relational Mapping

Agradecimentos

Gostaria de apresentar os meus agradecimentos a todas as pessoas que disponibilizaram algum tipo de assistência para a realização deste projecto, nos variados níveis do mesmo, começando pela fase inicial de decisão e de elaboração da proposta de projecto, passando pelas sugestões de funcionalidades e comentários técnicos e terminando nos aspectos sobre o relatório.

Além disso gostaria também de agradecer a todos os que contribuíram para a minha aprendizagem ao longo dos últimos quatro anos de frequência da Licenciatura, pois também contribuíram, de certa forma, para este projecto.

Índice

Índice de Figuras

Capítulo 1. Introdução

1.1 Introdução

Nos projectos típicos de desenvolvimento de *sites* dinâmicos para a Internet (e Intranet) existe um conjunto de operações e funcionalidades comuns cuja implementação é necessária para suportar o desenvolvimento dos referidos projectos.

Estas operações são as operações CRUD (Create, Read, Update e Delete) e permitem manipular os dados dinâmicos, tipicamente guardados num sistema de Gestão de Bases de Dados (SGBD).

Além das operações CRUD é também comum a criação de um conjunto de interfaces para facilitar a visualização e manipulação dos dados pelos utilizadores. A este conjunto de interfaces dá-se, normalmente, o nome de BackOffice.

A natureza repetitiva e trabalhosa da criação quer das operações CRUD, quer do BackOffice, torna estas funcionalidades candidatas interessantes para serem criadas através de processos de automatização que permitem agilizar o desenvolvimento dos produtos, permitindo assim alguma poupança de tempo de desenvolvimento e deixando os programadores livres para as tarefas menos repetitivas de cada projecto.

Desta forma, o objectivo principal deste projecto era a criação de uma aplicação que permitisse automatizar a criação dessas duas componentes específicas do desenvolvimento para a Internet e Intranet. A aplicação consistiria de duas partes principais: um gerador de Data Access Layer (Camada de Acesso a Dados) e um gerador de BackOffice.

O público alvo desta ferramenta são programadores que façam desenvolvimento de sites para a Internet (ou Intranet) e que estejam interessados em agilizar o seu processo de desenvolvimento através da utilização de ferramentas de automatização.

Neste relatório e nos seus anexos encontra-se documentada a ferramenta criada, assim como o respectivo processo de desenvolvimento.

1.2 Estrutura do Relatório

No capítulo 2 são apresentadas as linguagens e tecnologias escolhidas para desenvolver este projecto e as razões que levaram à escolha das mesmas.

No Capítulo 3 são descritas questões relacionadas com a Concepção, Arquitectura e Desenvolvimento das várias partes do projecto, assim como o método de trabalho.

No capítulo 4 são apresentadas as conclusões do projecto.

1.3 Nota prévia

Relativamente ao Capítulo 3 convém notar o seguinte: como o Projecto é uma ferramenta de Geração de Código, é relevante documentar o Código que é criado em diversas situações, pois não existe uma estrutura fixa para explicar, mas sim um padrão que é aplicado a cada Base de Dados para a qual se pretende gerar código. Assim sendo, as explicações relativas ao Código Gerado serão feitas da seguinte forma: em primeiro lugar é feita uma descrição abstracta, tipicamente usando a letra X para representar o nome de cada tabela da Base de Dados (em exemplos mais avançados são também usadas as letras Y e Z com o mesmo objectivo). Em segundo lugar é apresentado um exemplo concreto, recorrendo à descrição de uma ou várias tabelas no modelo relacional para explicar o problema, e recorrendo a diagramas UML para representar o código gerado para essa situação específica.

Capítulo 2. Linguagens e Tecnologias

2.1 Linguagens, Tecnologias e Técnicas Envolvidas no Projecto

Este projecto envolve, entre outras, as seguintes linguagens, tecnologias e conceitos relacionados com o desenvolvimento de Software: Programação (C#, PHP5), Programação Orientada por Objectos (Herança, Polimorfismo, Delegação, etc.), UML para documentação das classes criadas, HTML para as interfaces geradas, XML para descrição da Base de Dados, SQL e MySql para guardar e manter os dados.

2.2 Escolha de Linguagens e Tecnologias

Para o código do gerador foi escolhida a linguagem C# da plataforma .Net 2.0 porque:

- É uma das linguagens da plataforma .NET e tem boa integração com os Sistemas Operativos para os quais a ferramenta se destinava (Windows XP e Windows Vista)
- O IDE Visual Studio permite criar protótipos rápidos das Interfaces Gráficas do Software
- Possuí suporte de raíz para XML
- Existe uma versão gratuita (Visual C# Express 205) que suporta tudo o que era necessário para desenvolver o Projecto
- Pessoalmente tinha pouca experiência com C# mas já conhecia algumas das API da plataforma .NET pois já tinha feito vários trabalhos em VB.Net

Para o código gerado foi escolhido o PHP5 porque:

- O PHP é uma linguagem muito comum para o desenvolvimento Web
- Tem suporte para várias características da Programação Orientada por Objectos (como, por exemplo, o mecanismo de Herança)
- Tem uma boa integração com o SGBD MySql
- É gratuita
- Embora não conhecesse especificamente o PHP5 já tinha experiência com PHP4, o que me permitia desenvolver com alguma rapidez

Capítulo 3. Concepção, Arquitectura e Desenvolvimento do Projecto

3.1 Análise Preliminar

Antes de definir a arquitectura do Gerador foi definida a arquitectura e forma de funcionamento do Código Gerado. Para o fazer foram levados em conta os objectivos e requisitos da aplicação. Desta forma ficou definido que seria criado um conjunto de Código de Suporte para as funcionalidades comuns e que este código não seria gerado, por ser código de utilização geral e que não seria alterado em função da estrutura da Base de Dados do utilizador.

Desta forma o Projecto foi dividido em três (3) partes distintas:

1. Concepção, Arquitectura e Desenvolvimento do Código de Suporte
2. Concepção, Arquitectura e Desenvolvimento do Código Gerado:
    1. *Data Access Layer* e *Object Relational Mapping*
    2. BackOffice
3. Concepção, Arquitectura e Desenvolvimento do Gerador

Ao longo deste capítulo vão ser apresentadas várias explicações sobre o que foi feito em cada uma destas três partes e, também, sobre como foi feito.

3.2 Método de Trabalho

O desenvolvimento baseou-se em processos iterativos que consistiam em:

1) Divisão do problema (código a gerar) em várias partes pequenas
2) Escolher uma dessas partes
3) Desenvolver manualmente o código a gerar pela aplicação para resolver essa parte do problema
4) Testar esse código manualmente
5) Implementar o mesmo na ferramenta de geração de código
6) Voltar ao ponto 2, passando para outra pequena parte do código a gerar

As vantagens deste processo relacionam-se com a divisão do problema, o que permite a resolução de pequenos problemas individuais. Além disso, ao testar individualmente as pequenas partes, são

feitos testes graduais ao sistema durante o desenvolvimento, e não apenas no final.

3.3 Desenvolvimento

3.3.1 Código de Suporte

Como já foi referido, a parte inicial do projecto em termos de desenvolvimento consistiu na Concepção, Arquitectura e Desenvolvimento de um conjunto de Módulos compostos por Código de Suporte, utilidade global. Segue-se uma descrição de cada um dos módulos:

O Módulo **HTMLForms** é composto por um conjunto de classes que podem ser usadas para construir formulários HTML sem ser necessário criar manualmente o código HTML correspondente a esses formulários. Suporta formulários (métodos GET e POST) com as seguintes características:

- Inputs do tipo *Text*, *Hidden* e *Password*, assim como Inputs *readonly*
- Criação de Caixas de Selecção (*ComboBox*), Criação de *CheckBoxes*
- Criação de *RadioButtons*, assim como Colecções de *RadioButtons* (com suporte de exclusão mútua entre os vários *RadioButtons*, para que apenas um possa estar escolhido)
- Criação automática de conjuntos de Caixas de Selecção (*ComboBox*) para Datas:
    - *Date* / Datas (Dia, Mês e Ano)
    - *DateTime* / Datas com Hora (Hora, Minuto, Segundo, Dia, Mês e Ano)
    - Ambas incluem validação interna da Data escolhida. Exemplos:
        - não permite que se escolha o dia 31 de um mês que apenas tem 30 dias
        - não permite que se escolha o dia 29 de Fevereiro se o ano escolhido não for bissexto
        - etc.

O Módulo **ValidationRules** é integrado no Módulo **HTMLForms** e permite especificar Regras de validação para os formulários construídos com o Módulo HTMLForms. As regras incluídas são:

- IsInt, IsDouble – Verificação de Tipos
- MinLength, MaxLength, LengthBetween – Validação de Tamanho (Length)
- MinValue, MaxValue, ValueBetween – Validação de Valor

O Módulo **HTMLLayout** é composto por um conjunto de classes que permitem construir

elementos HTML de apresentação de dados. As características suportadas são:

- Tabela com Listas de Objectos
  - Permite listar os dados de vários Objectos (em cada linha aparece um objecto)
  - Incluí filtro para mostrar apenas os campos (propriedades) desejados de cada objecto)
  - Pode ser configurado com *links* para as páginas de Detalhe, Edição, Introdução e Eliminação
- Tabela com Detalhe de Objecto
  - Permite apresentar os dados de um Objecto (em cada linha da tabela aparece um campo (propriedade) do Objecto
- *Links* de Navegação
  - Tabela com *links* para Primeiro, Anterior, Seguinte, Último que permite controlar os valores a usar para aplicar navegação a uma lista de registos.

O Módulo **DBMySql** permite efectuar ligações ao SGBD MySql, assim como efectuar *queries* SQL, entre outras actividades relacionadas.

Estes módulos são fixos e não são gerados. Quando o Gerador de BackOffice é executado, estes módulos são copiados para a pasta de destino definida pelo utilizador.

Além de simplificarem o código gerado, estes módulos têm a vantagem de poderem ser usados em projectos nos quais não se recorra ao gerador, pois são autónomos em relação ao código gerado. A aplicação permite que o utilizador faça uma cópia do Código de Suporte para uma pasta sem ter de fazer qualquer geração de código. No Anexo I está disponível um diagrama UML que documenta as várias classes que compõem os módulos HTMLForms e ValidationRules.

3.3.2 Código Gerado

3.3.2.1 Geração de Data Access Layer e Object Relational Mapping

Foi definido numa fase muito inicial que o código gerado iria seguir uma estrutura Orientada por Objectos para o Data Access Layer (Camada de Acesso a Dados). No entanto, o modelo usado pelo MySql é o modelo Relacional, que tem algumas diferenças em relação ao modelo da Programação Orientada por Objectos.

Desta forma era necessário fazer um mapeamento de duas fases:

1) Mapeamento do Modelo Relacional para o Modelo OO – para Selecção de dados
2) Mapeamento do Modelo OO para o Modelo Relacional – para Introdução e Actualização de dados

Este mapeamento iria permitir a conversão dos registos guardados no modelo Relacional para os Objectos que representam esse registos no software (e vice-versa).

3.3.2.1.1 Introdução ao Mapeamento

Em primeiro lugar foi definido que cada Tabela seria representada para uma classe que iria conter os atributos dessa tabela. Ficou também definido que seria criada uma outra classe que seria utilizada para manipulação dos dados dessa Tabela (ou seja, esta segunda classe teria os vários métodos para implementação das operações CRUD, entre outros métodos auxiliares).

Posteriormente ficou também definido que iria existir uma classe que funcionaria como classe base para todas as classes criadas para representação de Tabelas. Foi ainda definida uma classe para base de todas as classes de gestão / manipulação de Tabelas. Para evitar colisões de nomes com outras classes do sistema, estas classes base têm o nome de **BCodeGenObjectBase e BCodeGenManagerBase**, respectivamente. Seguem-se os respectivos diagramas UML:

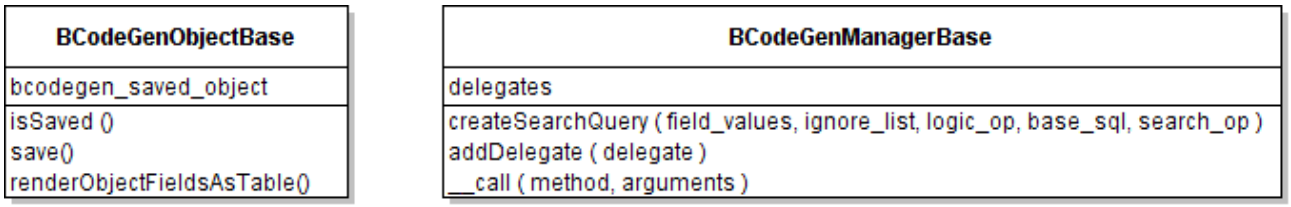

Figura 1: UML Classes BCodeGenObjectBase e BCodeGenManagerBase

Por se tratar de código não gerado, estas duas classes foram, posteriormente, adicionadas à biblioteca de Código de Suporte.

3.3.2.1.2 Classe de Representação de Tabela

Para cada tabela X da Base de Dados são criadas as seguintes classes de representação de dados:

- Classe XBase (subclasse de BCodeGenObjectBase) – ficheiro XBase.php
- Classe X (subclasse de XBase) – ficheiro X.php

A Classe X tem como objectivo ser a classe que é instanciada. Esta classe existe para permitir ao utilizador a extensão (até certo ponto) do código gerado.

O mecanismo de Herança da Programação Orientada por Objectos permite que todos os elementos (quer atributos, quer métodos) das classes **BCodeGenObjectBase** e **XBase** estejam presentes na classe **X**. Se o utilizador desejar adicionar conteúdos à classe **X** pode-o fazer, e esses conteúdos não serão perdidos caso o código seja novamente gerado, pois o gerador apenas reescreve as classes do tipo Base (ou seja, os ficheiros XBase.php). Isto permite, por exemplo, que o utilizador adicione métodos (ou lógica) especiais de formatação de dados à classe concreta X, métodos esses que serão mantidos mesmo que seja necessário voltar a gerar o código para suportar alterações feitas à Base de Dados (pois estas alterações são sempre aplicadas nos ficheiros e classes do tipo Base). Este esquema permite minimizar um problema que alguns geradores têm e que é uma das criticas comuns às ferramentas deste género: a pouca integração em processos de desenvolvimento contínuos e ágeis nos quais há muitas alterações ao modelo de dados.

A classe XBase contém os seguintes atributos:

- Todos os campos da tabela X
- Para cada campo da tabela X que pertença à Chave Primária será criado um campo extra de Backup desse valor, cujo objectivo é permitir que o utilizador faça alterações ao valor da Chave Primária – por exemplo, estes atributos são usados, internamente, pelos métodos de Update e Delete
- Atributo bcodegen_saved_object que tem o valor *true* quando o objecto representa um registo da tabela (ou seja, quando o objecto foi obtido da BD através de uma *query* de *Select*, ou quando o objecto já foi foi introduzido na BD através de uma *query* de *Insert*)

A classe XBase tem também os seguintes métodos:

- isSaved – Permite saber se o registo foi obtido da BD ou introduzido na mesma
- save – Permite definir como *true* o valor do atributo bcodegen_saved_object
- renderObjectFieldsAsTable – Permite construir uma tabela HTML em que cada linha da tabela apresenta um dos campos do objecto e o respectivo valor
- get – Permite obter o valor de um dos campos do objecto
- set – Permite definir o valor de um dos campos do objecto
- getPKFieldBackup – Permite obter o valor de um dos campos de backup de Chave Primária
- setPKFieldBackup – Permite definir o valor de um dos campos de backup de Chave

Primária

- getPKFields – Permite obter um *array* com os nomes dos campos do objecto que pertencem à Chave Primária
- getPKAsArgs – Permite obter uma string com os valores da Chave Primária formatados para utilização num URL HTTP (p.e. para serem usados pelo método GET dos formulários).

No Anexo II estão disponíveis mais detalhes sobre cada um destes métodos, nomeadamente ao nível dos parâmetros que cada um recebe.

Para exemplificar o código gerado nesta situação foi criado um diagrama UML simplificado onde se mostra a representação da classe BCodeGenObjectBase e de uma classe CarroBase cuja tabela respectiva tem a seguinte estrutura no modelo relacional:

CARRO (matrícula, data_aquisição, comentários)

Segue-se o modelo UML referido:

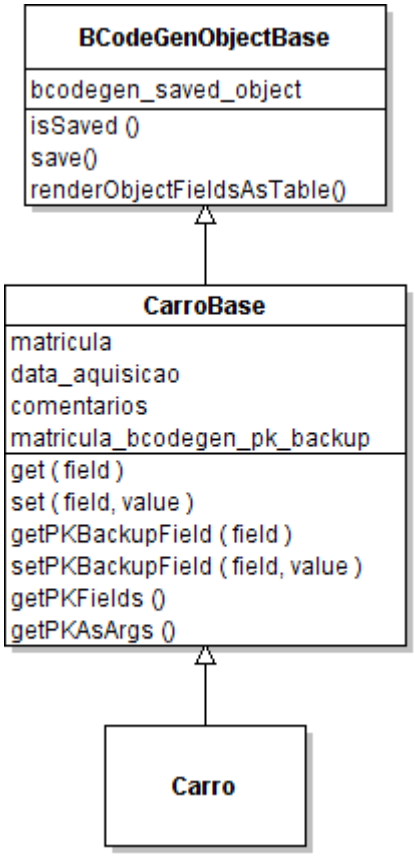

Figura 2: UML Classes BCodeGenObjectBase, CarroBase e Carro

Como se pode verificar a Classe CarroBase tem os três atributos da tabela, assim como um quarto, matricula_bcodegen_pk_backup, que serve para Backup da Chave Primária. Além disso tem os métodos já referidos para obter e definir valores, obter o *array* com a lista de atributos pertencentes à Chave Primária e obter a string com os atributos e valores para utilização em endereços HTTP. Tem também os atributos e métodos herdados da superclasse BCodeGenObjectBase.

3.3.2.1.3 Classe de Manutenção de Tabela

Para cada tabela X da Base de Dados são criadas as seguintes classes de representação de dados:

- XManagerBase (subclasse de BCodeGenManagerBase) – ficheiro XManagerBase.php
- XManager (subclasse de XManagerBase) – ficheiro XManager.php

Tal como no caso da Classe de Representação de Tabela, o esquema de duas classes tem como objectivo permitir alguma extensão do código gerado sem que a mesma seja feita na classe do tipo Base que é reescrita em caso de nova geração do código.

A Classe XManagerBase tem os seguintes atributos:

- defaults - um *array* com os valores por defeito de cada campo
- search_op - um *array* com o tipo de operador a usar para cada campo na pesquisa. Por exemplo, os campos de Texto ficam com o operador de pesquisa "LIKE", enquanto que os campos Numéricos ficam com o operador de pesquisa "=".
- delegates - um *array* com os objectos que sejam criados para actuarem como *delegates* desta classe
- o_db - o objecto de ligação ao SGBD

A Classe XManagerBase tem também os seguintes métodos:

- XManagerBase – construtor da classe
- insertX – permite introduzir um objecto da classe X na tabela correspondente da Base de Dados
- getX – permite obter da tabela da Base de Dados um determinado registo, identificado pela sua Chave Primária
- listX – permite obter um *array* de objectos da classe X a partir da tabela da Base de

Dados

- countRecords – permite obter o número de registos na tabela da Base de Dados
- updateX – permite actualizar na Base de Dados os dados de um objecto X
- deleteX – permite eliminar da Base de Dados os dados de um objecto X
- searchX – permite obter um *array* de objectos da classe X que cumpram um conjunto de condições
- prepareValues – permite preparar os valores de um objecto para serem usados numa *query* SQL de *Insert* ou de *Update*

No Anexo III estão disponíveis mais detalhes sobre cada um destes métodos, nomeadamente ao nível dos parâmetros que cada um recebe.

3.3.2.1.4 Caso Particular I – Tabela com Chave Primária Composta

Como já foi referido o gerador cria, nas classes do tipo Base, atributos de Backup para cada um dos campos que pertençam à Chave Primária da tabela. No exemplo anterior já se verificou essa geração para tabelas com uma Chave Primária composta por um único atributo. Segue-se um exemplo com um diagrama UML de uma classe gerada para representar uma tabela que tem uma Chave Primária composta por três (3) atributos, e que corresponde à seguinte tabela em modelo relacional:

FUNCIONÁRIOCARRO (funcionário, carro, data_requisição, data_entrega)

Neste caso a Chave Primária é composta por três atributos: Funcionário – Chave Estrangeira referente à tabela Funcionário; Carro – Chave Estrangeira referente à tabela Carro; e data_requisição.

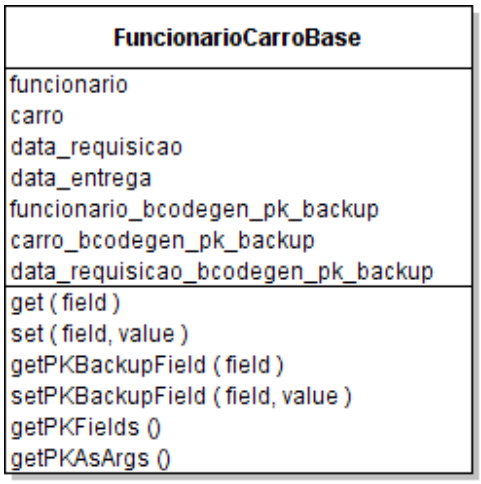

Figura 3: UML Classe FuncionarioCarroBase

3.3.2.1.5 Caso Particular II – Tabelas com Chaves Estrangeiras

Nos casos em que uma tabela tenha uma ou mais Chaves Estrangeiras, o valor dos atributos construídos para representar os atributos que sejam Chave Estrangeira será um objecto da classe que representa a tabela à qual cada uma das Chaves Estrangeiras se refere.

Para exemplificar este caso vamos recorrer a duas tabelas de exemplo, Funcionário e Cargo, que têm a seguinte estrutura relacional:

FUNCIONÁRIO (número, nome, telefone, telemóvel, morada, cargo)

CARGO (número, título)

Neste caso a tabela Funcionário tem uma chave estrangeira que é o atributo "cargo" que se refere à tabela Cargo (que tem como Chave Primária o atributo "numero"). De seguida apresenta-se um diagrama UML simplificado onde se pode verificar a associação existente entre as duas classes:

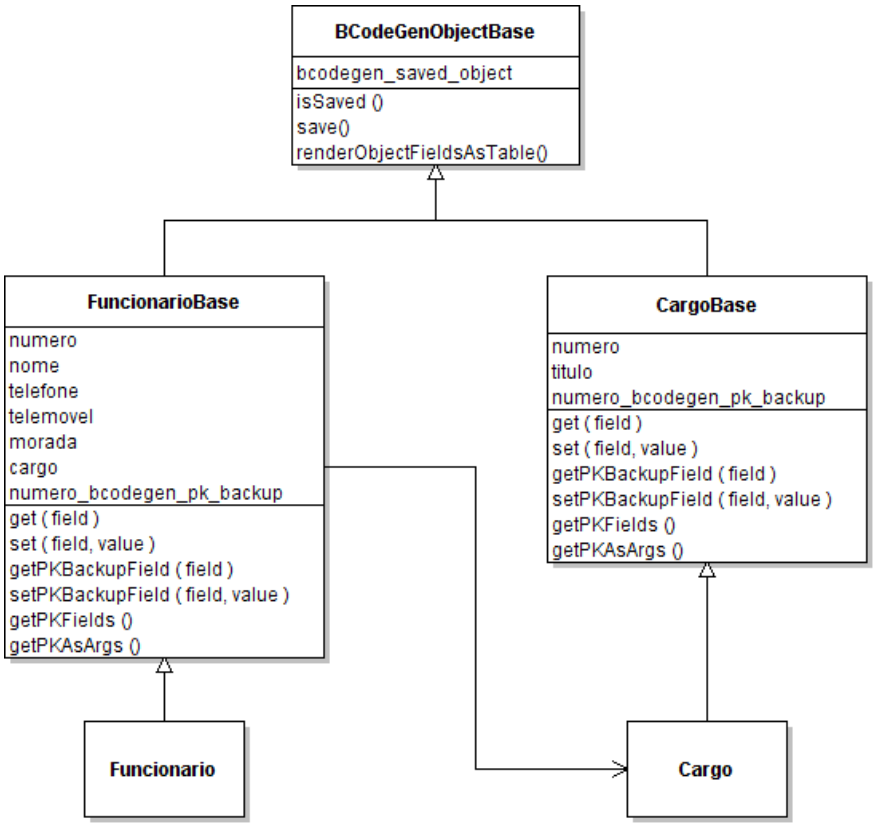

Figura 4: UML Classes FuncionarioBase e CargoBase

Em termos de implementação o que acontece é o seguinte: sempre que alguém fizer um pedido de um objecto da classe X, esse objecto trará associados todos os objectos de eventuais classes Y, ..., Z, que estejam associados ao mesmo através de Chaves Estrangeiras, em vez dos respectivos valores

(numéricos, etc).

Nota: o gerador apenas suporta Chaves Estrangeiras simples, não suporta Chaves Estrangeiras Compostas (mais do que uma chave para a mesma tabela de referência).

3.3.2.1.6 Caso Particular III – Tabelas com Associações Baseadas em Tabelas Auxiliares

No caso anterior foi abordado o exemplo de uma tabela (Funcionário) que estava associada a outra tabela (Cargo) sem uma tabela intermédia de mapeamento da associação. Esse é um caso comum para relações N-1 (como a do exemplo anterior) e em relações 1-1. No entanto, em relações de N-N ("Muitos para Muitos") é necessária a existência de uma tabela intermédia de mapeamento.

Nestas situações é útil ter um conjunto de métodos extra que permitam fazer manipulações diferentes dos dados. Por esta razão a classe XManagerBase tem alguns métodos extra, que se listam a seguir:

- addYToZ (objY, objZ, [...]) - permite criar uma associação entre Y e Z
- addZToY (objZ, objY, [...]) - tem o mesmo funcionamento que o anterior, apenas varia o nome por uma questão de legibilidade (este método recorre ao anterior)
- removeYFromZ (objY, objZ) - remove todas as associações entre Y e Z
- removeZFromY (objZ, objY) - tem o mesmo funcionamento que o anterior, apenas varia o nome por uma questão de legibilidade (este método recorre ao anterior)
- listYByZ (objZ, start, count) - permite obter um *array* com os objectos da classe Y que representam os registos da tabela Y que estão associados ao registo da tabela Z representado por um objecto da classe Z (objZ). Suporta navegação (parametros start e count).
- listZByY (objY, start, count) - permite obter um *array* com os objectos da classe Z que representam os registos da tabela Z que estão associados ao registo da tabela Y representado por um objecto da classe Y (objY). Suporta navegação (parametros start e count).

Onde Y e Z representam as outras tabelas envolvidas na associação e [...] representa outros eventuais parâmetros pertencentes à Chave Primária da tabela de associação. Além disso, cada uma das classes Y e Z fica com a classe X como "*delegate*". Os "delegates" permitem que uma classe faça a delegação de uma tarefa a outra a classe, ficando a segunda classe responsável pela execução dessa tarefa. O mecanismo de delegação é implementado usando o método especial __call do PHP5, que permite saber quando é feita uma chamada a uma função não existente. Este método foi implementado na classe BCodeGenManagerBase (estando assim disponível em todas as classes do

tipo Manager criadas pelo gerador) e tem lógica que permite percorrer a lista (array) de "*delegates*" de uma classe à procura de um determinado método. Se o mesmo existir num dos *delegates*, é enviado o pedido de execução a esse *delegate*. Se não existir um método com esse nome o código está preparado para lançar uma Excepção avisando o utilizador que tentou usar um método desconhecido. Para isto o código recorre ao mecanismo de excepções do PHP5.

Segue-se um exemplo desta situação, tendo por base as seguintes tabelas:

FUNCIONÁRIO (número, nome, telefone, telemóvel, morada, cargo)

FUNCIONARIOPROJECTO (número_funcionário, número_projecto, data_entrada)

PROJECTO (número, título, cliente)

Segue-se o diagrama de classe em UML que representa a classe FuncionarioProjectoManagerBase e que demonstra a associação de Agregação que permite a delegação de tarefas pelas classes FuncionarioManagerBase e ProjectoManagerBase.

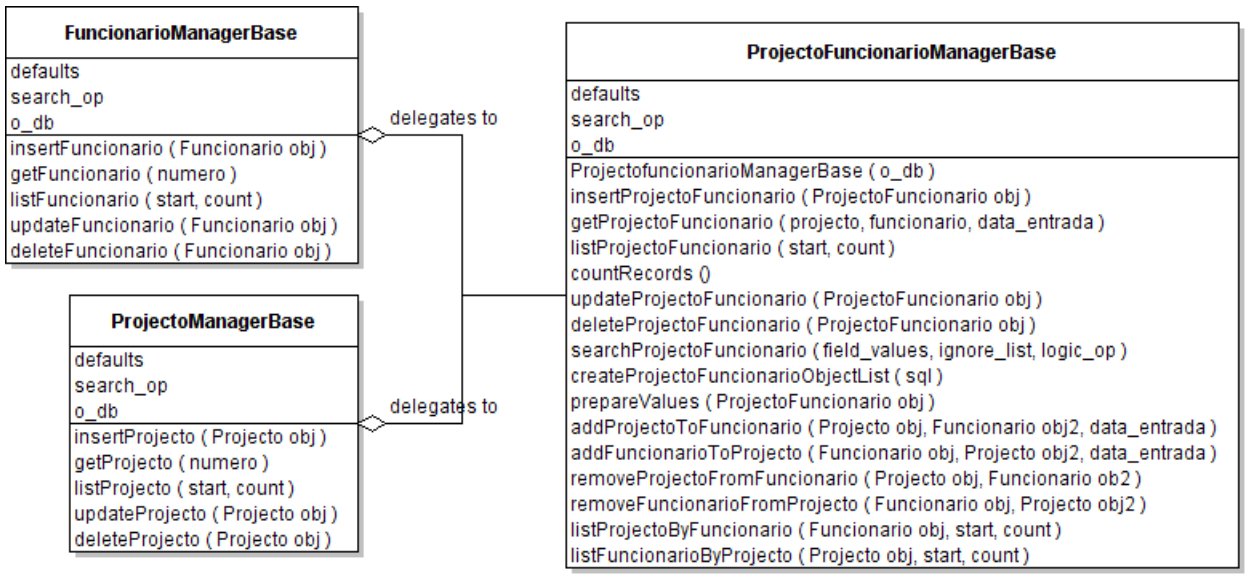

Figura 5: UML Classes Demonstração de Delegação

Nota: o gerador apenas suporta relações de N-N com duas tabelas e não suporta auto-junções (junções para a mesma tabela).

3.3.2.1.7 Geração de SQL

Para construir os métodos responsáveis pelas operações CRUD é necessário construír as respectivas *queries* SQL. Praticamente todas as *queries* utilizadas são criadas durante a fase de geração, recebendo apenas alguns parâmetros durante a fase de execução. A excepção a esta regra é a *query* de pesquisa, que embora seja parcialmente construída durante a fase de geração, acaba, tipicamente, por ser muito alterada (não apenas em termos de parâmetros mas também em termos

de estrutura das condições pois a pesquisa depende das escolhas do utilizador que pode ignorar alguns campos, usar o operador AND ou o operador OR, etc.). Era possível construir o código de forma a que todas as *queries* fossem completamente geradas na fase de execução. Isto permitia evitar alguma duplicação de código ao longo das várias classes que são geradas. No entanto dessa forma o código gerado poderia ser mais pesado, pois teria de englobar mais ciclos e concatenações de strings (para definição das listas de campos a seleccionar, assim como para construir as condições de cada *query*, entre outras actividades relacionadas). A forma escolhida torna possível encontrar um equilíbrio entre a qualidade do código e a *performance* do mesmo.

3.3.2.2 Geração de BackOffice

O Gerador de BackOffice baseia-se na criação de um conjunto de interfaces HTML que podem ser utilizadas para gerir a Base de Dados. É aqui que a utilidade do Código de Suporte se revela, em particular os módulos HTMLForms, HTMLLayout e ValidationRules, pois os mesmos, em conjunto com o Código Gerado pelo Gerador de Data Access Layer, permitem construir interfaces gráficas para gestão da Base de Dados de forma muito simples e rápida, através de criação de objectos e de chamadas a métodos desses objectos.

O diagrama que se segue mostra a forma como o Código gerado para BackOffice depende do Código de Suporte e do Código gerador pelo Gerador de Data Access Layer.

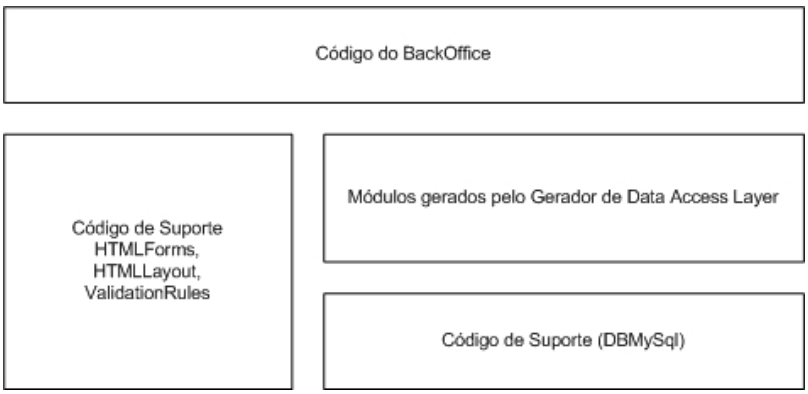

Figura 6: Dependências do Código do BackOffice

As páginas (interfaces) que o Gerador de BackOffice permitir construir são as seguintes:

- Listagem de Registos (com possibilidade de adicionar Navegação)
- Detalhe de Registo
- Introdução de Registo
- Actualização de Registo

- Eliminação de Registo
- Pesquisa de Registo (Configurável pelo utilizador final)
- Página de Index com Links para as páginas de Listagem, Introdução e Pesquisa de cada tabela
- Página de Login e Página de Logout

O utilizador do Gerador pode escolher os campos que quer que apareçam em cada página, assim como se os campos são editáveis, se têm regras de validação, entre outras configurações. Isto permite que o utilizador omita, por exemplo, dos formulários de Introdução e Edição, os campos que pertencem à Chave Primária (e que em muitos casos não interessa mudar).

A página de Pesquisa de Registo além de poder ser configurada pelo utilizador do Gerador, permite também ao utilizador (final) do BackOffice a configuração da query que pretende fazer, o que permite que o mesmo que ignore alguns dos campos e que defina se pretende usar o operador lógico "And" ou o operador lógico "Or". Isto torna a página de Pesquisa numa ferramenta mais flexível.

Foi também criado um mecanismo de autenticação através do qual o utilizador pode proteger o acesso às tabelas. Esta autenticação é configurada durante a criação do BackOffice, e o utilizador pode escolher usar, para tabela de autenticação, uma das tabelas da sua estrutura, ou criar uma nova. Caso escolha criar uma nova tabela, o Gerador garante que o nome escolhido não existe já na lista de tabelas e, caso não exista, cria o código necessário para gerir essa tabela (incluindo as classes Base e **ManagerBase** que são criadas para as restantes tabelas). A tabela de autenticação, quer seja nova, quer seja uma existente, tem um método extra na sua classe de Manutenção (**XManagerBase**) que é o método *login*, que recebe como parâmetros duas strings (*username* e *password*) e devolve *true* (caso seja um utilizador registado) ou *false* (caso não seja um utilizador registado).

3.3.3 O Gerador de Código

3.3.3.1 Componentes do Gerador

O Gerador de Código foi desenvolvido em C# para ser utilizado como uma aplicação de Desktop.

Foram construídos dois Wizards, cada um responsável pela geração de uma parte do Projecto:

1) Wizard de Geração de Data Access Layer (Camada de Acesso a Dados)
2) Wizard de Geração de BackOffice

Como a geração de código é feita a partir de uma descrição da base de dados, é necessário obter essa descrição de algum lado e ter estruturas de dados que permitam representar e manipular essa informação dentro do programa. A Base de Dados é descrita através de XML, que é interpretado de forma a obter as várias tabelas e respectivos campos. Para fazer representar a Base de Dados internamente foram criadas algumas classes cujo objectivo é representar uma Base de Dados no Modelo Relacional. Estas são as classes Database, Table e Field. Para mais informações sobre essas classes deve ser consultado o Anexo IV.

Além de representar a Base de Dados, era também necessário manipular os dados obtidos, para fazer a geração de código propriamente dita. Isto é feito por três classes: **ClassCreator**, **BackOfficeInterfaceCreator** e **TableOptions**.

A classe **ClassCreator** é responsável pela criação do código do Data Access Layer. Esta classe relaciona-se com as classes usadas para representação da Base de Dados já referidas no ponto anterior. Além de criar as classes representação e manipulação das várias tabelas, esta classe permite gerar também um ficheiro "Base.php" no qual são feitas as inclusões de todos os ficheiros criados, através da função require_once do PHP5. Este ficheiro serve para facilitar a utilização do código gerado, pois basta o utilizador incluir o mesmo ficheiro noutro ficheiro PHP para ter acesso a todo o código gerado.

A classe **BackOfficeInterfaceCreator** é responsável pela criação do código do BackOffice. É composta por um conjunto de métodos que permitem criar cada uma das páginas do BackOffice (por exemplo: createListFile para criação do ficheiro de Listagem, createDetailFile para o ficheiro de Detalhe, entre outros). Contém também alguns métodos utilitários que auxiliam na criação das páginas. Os métodos de criação de páginas recebem como parâmetros dois objectos: um objecto da classe Table e um objecto da classe TableOptions. Funcionam, desta forma, por tabela. Ao serem invocados, estes métodos percorrem a lista de campos da tabela passada por parâmetro, e fazem verificações de condições diversas. Por exemplo, ao criar a página de Listagem, o código verifica, para cada campo, se o mesmo está marcado no objecto TableOptions para ser listado. Se estiver, esse campo é escrito para o código PHP como membro de um *array* ($fields) onde ficam todos os campos a apresentar na página de Listagem.

Por último, a classe **TableOptions** é utilizada para guardar as opções do utilizador relativamente aos ficheiros a criar no BackOffice, assim como os campos a mostrar em cada ficheiro, as validações a implementar, entre outras opções. Quando o utilizador finaliza a configuração das páginas a gerar, o Gerador percorre todas as tabelas e verifica se existe um objecto da classe TableOptions associado a essa tabela. Se existir então o utilizador configurou algumas

páginas para a mesma pelo que o Gerador recorre à classe BackOfficeInterfaceCreator e aos seus métodos para criar as páginas. Se não existir um objecto TableOptions para a tabela em causa então o utilizador não configurou nada para essa Tabela, e a mesma pode ser ignorada. Para mais informações sobre estas três classes deve ser consultado o diagrama UML do Anexo V.

3.3.3.2 Método de Geração de Código

O método usado para geração de código baseia-se na escrita para ficheiros de instruções pré-definidos que são alteradas com informação sobre a Base de Dados (quais as Tabelas, quais os campos de cada Tabela, quais os campos que pertencem à Chave Primária, etc.). Existiam outras opções para fazer a geração, nomeadamente o uso de *templates* e a substituição de *keywords* definidas nos mesmos (que acabaria por ser semelhante à solução adoptada). A solução implementada foi escolhida por ser bastante directa em termos de implementação.

3.3.3.3 Interface Gráfica do Gerador

O Gerador possuí uma interface gráfica composta por vários formulários e *UserControls*. Os *UserControls* foram usados para possibilitar a reutilização dos ecrãs, pois os dois *Wizards* têm ecrãs semelhantes (por exemplo, ambos os *Wizards* têm o ecrã onde o utilizador define a pasta e nome do projecto) pelo que era importante ter uma forma de partilhar essas partes comuns entre os dois *Wizards* para evitar duplicação dos mesmos.

Algumas partes dos formulários e *UserControls* são geradas dinamicamente. Por exemplo isto acontece no BackOfficeFieldConfigurationUC que apresenta os atributos das várias tabelas e permite escolher o que aparece em cada página do BackOffice. Neste *UserControl* é criada uma tabela dinâmica com um número variável de componentes visuais (CheckBoxes, ComboBoxes, Botões, etc.) em função do nome e tipo dos atributos da tabela a ser apresentada em cada momento.

Foi ainda criada uma Interface (*UCValidationInterface*) cujo objectivo é definir uma estrutura que todos os *UserControls* têm de seguir para serem integrados nos *Wizards*. Esta interface define três métodos que têm de ser implementados pelos *UserControls* a usar nos *Wizards*:

- isValid – verifica que o input feito no *UserControl* é válido
- showErrors – mostra os erros eventualmente encontrados no input
- save – guarda os dados para utilização posterior

Com a criação desta Interface tornou-se relativamente simples a introdução de novos UserControls aos *Wizards*, pois esta simplifica a sua integração.

Capítulo 4. Conclusões

4.1 Conclusão

Foi possível desenvolver uma aplicação de geração de código capaz de construir código PHP5 funcional para algumas das funcionalidades mais comuns do desenvolvimento para a Internet, cumprindo assim os objectivos do projecto. O Código de Suporte, que não fazia parte da ideia inicial da aplicação, acabou por se tornar útil, tendo facilitado a construção do Gerador de BackOffice, e revelando-se também com uma componente interessante mesmo quando utilizada fora do contexto do Gerador de Código.

Em relação aos requisitos ficou por terminar a implementação da obtenção directa da estrutura da Base de Dados a partir do SGDB MySql. O código base está parcialmente desenvolvido, faltando, no entanto, alguns pormenores, assim como a integração desse código com a interface gráfica do Gerador. No entanto a falta desta característica não impede o uso da ferramenta, pois a descrição da estrutura da Base de dados usando XML funciona correctamente.

Além de ter levantado vários problemas de modelação e de desenvolvimento/implementação cuja resolução foi interessante e serviu para aprendizagem, o desenvolvimento deste trabalho permitiu também aplicar algumas técnicas que já conhecia mas que ainda não tinha aplicado de forma prática, como, por exemplo, a técnica de Delegação. Permitiu, também, ficar a conhecer melhor as ferramentas envolvidas.

4.2 Perspectivas de Trabalho Futuro

A ferramenta ainda tem algum espaço para crescer em várias direcções, mas existem duas áreas que teriam prioridade caso algum dia volte a trabalhar neste projecto. A primeira dessas áreas seria concluir a implementação da obtenção directa da estrutura da Base de Dados a partir do SGDB MySql. A segunda área seria a implementação de um esquema de autenticação (para o BackOffice) mais flexível, no qual os utilizadores tivessem vários níveis de acesso e não apenas um como actualmente.

Bibliografia

Pereira, Alexandre, “Engenharia da Programação” (Apontamentos disponíveis em http://escola.mediateca.pt/ep/manuais/)

Almeida, António José, “Bases de Dados” (Apontamentos da Disciplina de Bases de Dados)