Universidade do Porto

Faculdade de Engenharia

FEUP

Cláudio Emanuel Valério Gomes

Reengenharia da Plataforma de Suporte ao Blog.com

47.3)
C5202
PERc

C
2005

Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

Reengenharia da Plataforma de Suporte ao Blog.com

Relatório do Estágio Curricular da LEIC

Cláudio Emanuel Valério Gomes

Orientador na FEUP: Prof. Jaime E. Villate
Orientador na Portugalmail - Comunicações, S.A.: Eng. Sérgio Carvalho

Agosto de 2005

Universidade do Porto
Faculdade de Engenharia
Biblioteca

004/047.3)/LEIC/FEUP/2004/COMe

Universidade do Porto
Faculdade de Engenharia
Biblioteca
Nº 81403
CDU
Data 15/03/2006

Para a Caroline

Conteúdo

1 Introdução **1**

1.1 A Portugalmail 1
1.2 O Blog.com 2
1.3 O projecto de reengenharia do Blog.com 3
1.4 Organização dos temas abordados 4

2 Internacionalização **6**

2.1 Introdução 6
2.2 Requisitos 7
2.3 Análise tecnológica 7
2.3.1 SmartyML 8
2.3.2 Pear I18N 11
2.3.3 Flaimo's Little Package 11
2.3.4 Aztek 15
2.3.5 Smartest Smarty Multilanguage Support 15
2.3.6 Smarty Gettext 17
2.3.7 A ferramenta escolhida 19
2.4 PoEdit 19
2.5 O processo de tradução 21
2.5.1 Marcação 21
2.5.2 Recolha 21
2.5.3 Salvaguarda da informação 23
2.5.4 Junção de traduções 24
2.5.5 Tradução 24
2.5.6 Formatação 25
2.6 Scripts de automatização 25
2.6.1 Automatização da marcação de texto 26
2.6.2 retrieve.sh 27
2.6.3 append.sh 27
2.6.4 trim.sh 28
2.6.5 format.sh 28

3 Estrutura da árvore da aplicação **29**
3.1 Motivação 29
3.2 Descrição do trabalho 31

4 Refactoring **34**
4.1 O que é Refactoring? 34
4.2 Porque é necessário? 36
4.3 Quando deve ser utilizado? 37
4.4 Onde fazer *refactoring*? 39
4.5 Automatização 42
4.6 Utilização no projecto 43
4.6.1 Eliminação de código duplicado 44
4.6.2 Breadcrumbs e Header 45
4.6.3 Environment 47

5 Redesenho da barra lateral **49**
5.1 Descrição da barra lateral 49
5.1.1 Books 49
5.1.2 Syndication 52
5.1.3 Recent Comments 52
5.1.4 Search 52
5.1.5 Blogrolls 53
5.1.6 Monthly Archives 53
5.1.7 Powered 54
5.1.8 Advertisement 54
5.1.9 Category List 54
5.1.10 Album List 55
5.1.11 Profile 55
5.2 Motivação 55
5.3 Descrição do trabalho 57
5.3.1 O objecto bloco e o polimorfismo 57
5.3.2 Blocos de livros múltiplos 61
5.3.3 Blogrolls. Refactoring do design obsoleto 64
5.3.4 Perfis. De subsecção a secção independente 65

6 Conclusões **68**

Lista de Figuras

3.1 Estado inicial da raiz da aplicação 30
3.2 Raiz da aplicação reorganizada 32
3.3 Separação dos componentes 33

4.1 Histórico de navegação corrigido 46
4.2 Utilização de breadcrumbs 46

5.1 Barra lateral do Blog.com. 50
5.2 Bloco de livros inicial. 51
5.3 Secção barra lateral reorganizada. 58
5.4 Selecção do tipo de bloco a criar. 59
5.5 Diagrama de classes idealizado. 59
5.6 Polimorfismo originado pela DB_OO. 60
5.7 Associação originada na delegação de responsabilidades. 60
5.8 Múltiplos blocos de livro na mesma barra lateral. 62
5.9 Opção para apresentação do bloco na barra lateral. 62
5.10 Opção de apresentação do bloco desactivada. 63
5.11 Multiplicidade das associações da classe BookBlock. 63
5.12 Indicação do estado de visibilidade dos blocos. 63
5.13 Alerta de procura de livro demasiado genérica. 64
5.14 Multiplicidade das associações da classe Blogroll. 65
5.15 Nova secção Profiles. 66
5.16 Multiplicidade nas associações da classe Profile. 67

Resumo

Este documento descreve o projecto de reengenharia do sítio WEB BLOG.COM, realizado na empresa PORTUGALMAIL, COMUNICAÇÕES S.A., desenvolvido no âmbito do estágio curricular intitulado REENGENHARIA DA PLATAFORMA DE SUPORTE DO BLOG.COM, da LICENCIATURA EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO, da FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO.

O projecto baseava-se em duas fases de desenvolvimento distintas. A primeira fase destinava-se exclusivamente à internacionalização do sítio WEB. Numa segunda fase, pretendia-se uma análise crítica à estrutura do BLOG.COM, da qual resultassem acções de reengenharia para sua melhoria.

A estrutura do documento foi escolhida para reflectir a ordem na qual os eventos decorreram e abordar os diferentes temas isoladamente.

Descreve-se a internacionalização do sítio WEB, incluindo os seus requisitos, análise tecnológica das várias alternativas, implementação e alguns elementos chave da sua utilização. É igualmente descrita a intervenção para reestruturação da árvore da aplicação, com implicações sobretudo ao nível do sistema de ficheiros. É feita uma apresentação da técnica do REFACTORING e seus principais aspectos, dedicando-lhe uma atenção especial dada a sua relevância para este tipo de projecto. São apresentadas algumas das acções onde o REFACTORING foi utilizado, servindo como exemplos de utilização da técnica. Descreve-se por último, o redesenho da barra lateral do editor do BLOG.COM, que por abranger modificações nas funcionalidades e interface com o utilizador, representa a intervenção cujas modificações são mais visíveis do exterior.

Conclui-se que um dos aspectos aliciantes deste projecto terá sido a multiplicidade de tecnologias utilizadas, e que o facto de os seus prazos e objectivos terem sido integralmente cumpridos deveu-se também ao espírito de equipa e condições de trabalho na PORTUGALMAIL - COMUNICAÇÕES, S.A.. Conclui-se ainda que a reengenharia do Blog.com não fica esgotada com este projecto, sendo necessário continuar o trabalho aqui iniciado e expandi-lo à restante plataforma de modo a manter o dinamismo do BLOG.COM.

Capítulo 1

Introdução

Neste capítulo será descrito o âmbito no qual foi desenvolvido o projecto de reengenharia do BLOG.COM. São apresentadas as empresas envolvidas e a sua relação, a origem do BLOG.COM e as motivações para o projecto de reengenharia.

Desde já agradece-se a toda a equipa da Portugalmail - Comunicações, S.A. pelo excelente ambiente de trabalho proporcionado, em particular o orientador do projecto pela empresa, Eng. Sérgio Carvalho e ao Eng. Nuno Lopes cujos contributos para este projecto foram de um valor inegável.

Agradece-se ainda na Faculdade de Engenharia da Universidade do Porto a todos os que tornaram este projecto possível. Pela sua contínua disponibilidade e apoio, o meu profundo agradecimento ao meu orientador pela universidade, o Doutor Jaime Villate. Agradeço ainda aos responsáveis pelos estágios curriculares da Faculdade de Engenharia da Universidade do Porto, Eng. Augusto Sousa e Eng. Raul Vidal pelo apoio facultado nos momentos cruciais de término deste projecto. Agradeço ainda a todo o secretáriado da LEIC, cujo apoio logístico ultrapassou fronteiras.

Por último, agradeço a todas as pessoas que estiveram sempre do meu lado, os meus amigos e em particular a Caroline Massin, com consciência que um simples agradecimento, nunca será suficiente para reflectir o vosso apoio.

1.1 A Portugalmail

A PORTUGALMAIL - COMUNICAÇÕES S.A. é uma PME a operar no domínio das Dot.Com. As suas origens encontram-se na cidade do PORTO, onde ainda mantém os seus escritórios e centro de desenvolvimento, mantendo permanentemente perto de uma dezena de funcionários e colaboradores.

Entre os principais serviços disponibilizados, encontra-se o serviço de correio electrónico, serviço de acesso à INTERNET e serviços de alojamento WEB. Des-

tes, é o serviço de correio electrónico que ocupa a posição de maior destaque, sendo considerado desde a sua criação uma referência no mercado nacional, e oferecendo actualmente uma gama de soluções adaptadas a todas as necessidades, particulares e empresariais.

A PORTUGALMAIL - COMUNICAÇÕES S.A. encontra-se em crescimento, esperando-se o aumento do seu volume de negócios, quer pela disponibilização de novos serviços no mercado nacional, como o anunciado serviço de transmissão de voz, quer pela abertura a novos mercados.

1.2 O Blog.com

Depois de estabelecida a sua posição no mercado nacional, uma das principais limitações à expansão da PORTUGALMAIL - COMUNICAÇÕES S.A. era o facto de os seus serviços estarem orientados e conotados ao mercado português, onde a nível de utilização *per capita* de novas tecnologias é inferior àquela dos restantes parceiros europeus, e muito inferior à de alguns outros países não comunitários, como o JAPÃO ou os ESTADOS UNIDOS DA AMÉRICA. Para além disso, o mercado nacional apresenta neste domínio perspectivas de crescimento inferiores a países menos desenvolvidos mas em rápido crescimento, como a ÍNDIA e a CHINA.

Para contrariar esta tendência, a PORTUGALMAIL - COMUNICAÇÕES S.A. abraçou um novo e ambicioso projecto de expansão, com o intuito de alargar a sua oferta de serviços base a novos mercados. Criou-se assim a BLOG.COM, uma empresa satélite da PORTUGALMAIL, sediada nos ESTADOS UNIDOS DA AMÉRICA mas que mantém a base de operações em PORTUGAL.

A criação desta empresa seguiu as mais recentes tendências da utilização da WEB, no campo da publicação pessoal *online*, tendo sido sobretudo orientado para o mercado norte-americano. Por esta razão o sítio WEB encontra-se exclusivamente em inglês, tendo sido mantida uma variante para o mercado nacional com o domínio BLOG.PT.

A BLOG.COM aposta na simplicidade de utilização e na publicação *online* gratuíta como elementos para o seu sucesso. Recorrendo à sua experiência adquirida nos projectos anteriores, disponibilizou uma oferta de modalidades variada, para se ajustar a várias necessidades distintas.

A modalidade gratuita apresenta já características que lhe permitem ajustar às necessidades da maioria dos utilizadores. Os *blogs* criados nesta modalidade apresentam uma pequena área de publicidade, utilizada como forma de rentabilizar o serviço prestado, e mantêm alguns limites na quantidade de informação transferida mensalmente e no espaço disponibilizado em disco para texto e imagens.

Os autores dos *blogs* mais visitados, e como tal consumidores de banda, e

aqueles com outras necessidades que não se enquadrem nas características da modalidade gratuita, podem encontrar uma modalidade adequada nas versões não gratuitas. Estas versões apresentam limites menores e mais funcionalidades, como a possibilidade de remover a área de publicidade ou de receber informação estatística da utilização do *blog*.

1.3 O projecto de reengenharia do Blog.com

Desde a sua versão inicial, o BLOG.COM tem vindo a sofrer repetidas intervenções, de modo a manter o seu serviço actualizado face às tendências e tecnologias em permanente mutação da WEB, e a reforçar a sua posição como serviço de utilização simples. Estas intervenções obrigam a alterações do código que nem sempre se adequam da melhor forma na estrutura de código e interfaces com o utilizador anteriores, levando a uma degradação progressiva da aplicação.

Por outro lado, a estratégia da empresa ditava que havia chegado o momento de deixar de produzir um serviço orientado para o mercado norte-americano e proceder à sua internacionalização, para uma melhor penetração nos restantes mercados, como os europeus e asiáticos.

O conjunto de modificações necessárias tornava a altura propícia para um projecto de reengenharia geral do sítio WEB. O projecto continha dois objectivos distintos para corresponder às necessidades indicadas nos parágrafos anteriores. Como primeiro objectivo, pretendia-se dotar o BLOG.COM de capacidade plurilinguística. Para tal seria necessário efectuar o estudo das principais formas de internacionalizar um sítio com as características do BLOG.COM, e baseado nas conclusões desse estudo, proceder à sua implementação. Como segundo objectivo, pretendia-se que fosse feita uma análise crítica à aplicação, de modo a identificar possíveis melhorias que pudessem ser efectuadas para contrariar a degradação progressiva do código.

A segunda componente do projecto, consiste assim num conjunto de várias modificações relativamente pequenas à aplicação, que variam tanto no tipo como em dimensão da intervenção.

Quanto ao tipo, podem-se agrupar as intervenções efectuadas em três categorias distintas:

Refactorings Representam intervenções no código sem impacto visível no comportamento externo da aplicação e imperceptíveis ao utilizador. Visam obter melhorias qualitativas no código, como melhor legibilidade ou resistência à introdução de erros.

Alterações de funcionalidade Representam alterações que visam acrescentarem ou modificarem as funcionalidades disponibilizadas ao utilizador. Estas

alterações afectam o comportamento externo da aplicação com impacto na interface com o utilizador.

Alterações à disposição no sistema de ficheiros Representam uma reorganização da aplicação no sistema de ficheiros de modo a obter vantagens qualitativas. À semelhança dos *refactorings*, também não introduzem alterações ao nível do comportamento externo da aplicação. Torna-se necessário distinguir este tipo de intervenção dos *refactorings* pois neste caso a intervenção não é feita no código, não se enquadrando assim no conceito de *refactoring*, como veremos no capítulo 4.

Quando às dimensões das intervenções, encontramos desde modificações muito localizadas, a apenas algumas linhas de código e normalmente associadas a acções de *refactoring*, até alterações já extensas, que requerem a modificação e criação de um número significativo de classes e com impacto na estrutura da base de dados.

1.4 Organização dos temas abordados

Este documento foi estruturado de forma a abordar em capítulos próprios os diferentes tipos de intervenção, conforme descritos em 1.3, dado estes constituírem problemas distintos que requerem abordagens próprias. Os capítulos encontram-se ainda organizados de forma a reflectir a evolução cronológica do projecto.

Desta forma, será abordado no capítulo 2 a internacionalização do BLOG.COM. Por a solução adoptada ser baseada numa solução já existente, que como veremos conjuga o SMARTY e o GETTEXT, e não no desenvolvimento de uma solução feita à medida e construída de raiz, é dado mais ênfase na descrição das várias possibilidades analisadas que motivaram a escolha em detrimento da implementação em si.

Os capítulos seguintes descrevem a segunda componente do projecto. O capítulo 3 descreve a acção de reestruturação da aplicação ao nível do sistema de ficheiros, normalmente designada árvore da aplicação. São indicadas as motivações que deram origem a esta intervenção, a técnica utilizada para reestruturar a árvore e a estrutura final adoptada.

No capítulo 4 descreve-se a utilização da técnica de *refactoring* no projecto. A leitura das primeiras secções deste capítulo introduz os conceitos principais do *refactoring*, necessários para compreender a motivação para as intervenções e alguma da terminologia utilizada. É feita uma apresentação inicial da técnica, que reflecte o período dedicado à análise e aprendizagem da mesma. Descreve-se a motivação para a sua utilização, quando e onde deve ser utilizada e uma lista com os *refactorings* mais frequentes.

Após a apresentação do *refactoring* são referidas algumas das intervenções efectuadas em que recorreu-se a esta técnica. Por estas tenderem a ser de dimensões muito reduzidas e bastante localizadas não é prático apresentar a sua totalidade. De modo a conseguir-se caracterizar o tipo de trabalho efectuado, o conjunto de *refactorings* apresentado reúne o tipo de *refactorings* efectuados mais frequentemente assim como aqueles que individualmente apresentam um maior impacto no desenho do código.

No capítulo 5 descreve-se o redesenho de um dos componentes principais dos *blogs*, a sua barra lateral. É feita uma apresentação inicial dos vários constituintes da barra lateral, que reflecte a análise efectuada para determinar as suas principais características e forma de utilização. Os elementos reunidos nesta análise são utilizados de seguida para descrever várias questões identificadas, que em conjunto constituem a motivação para a intervenção, descrevendo-se por último a sua implementação.

Capítulo 2

Internacionalização

2.1 Introdução

Ao desenvolver um sítio WEB, é frequente disponibiliza-lo numa única língua. Isto pode ter origem num conjunto muito variado de factores, como por exemplo a selecção de um público-alvo, a predominância de um determinado mercado face aos restantes ou o facto de a implementação numa só língua ser mais simples. Existem entretanto situações em que se torna vantajoso, ou mesmo necessário, mudar para um sistema que permita apresentar o conteúdo numa variedade maior de línguas. Ao processo de transição que permite dotar um sítio WEB desta capacidade, denominou-se internacionalização. O termo é longo, pelo que é frequente encontrá-lo na sua forma abreviada. Esta tem origem no termo em Inglês *internacionalization*. Utiliza-se a primeira e a última letra da palavra, com uma menção numérica a quantas letras se encontram entre as duas, resultando em I18N.

Na realidade, internacionalizar um sítio Web pode significar mais do que apenas traduzi-lo em várias línguas. Pode significar dota-lo da capacidade de, para cada país suportado, conseguir apresentar a informação nos formatos locais, como a hora, moeda, unidades de medida, língua e mesmo os costumes culturais. Esta utilização mais vasta é facultativa e depende apenas das necessidades de cada um. Ao conjunto de dados relativo a um local geográfico específico chama-se um *locale.*

A internacionalização representa uma acção de generalização em que partimos de uma situação específica para um único local, para uma outra mais genérica, em que podemos suportar vários locais. Associado à internacionalização encontramos também o termo "localização". Este refere o acto de a um sítio já internacionalizado, ser adicionada informação para um novo local. A localização representa uma acção de especificação, em que a um sistema genérico adicionamos informação específica dum local. Por analogia a I18N, localização aparece

frequentemente na forma abreviada L10N.

O BLOG.COM foi desenvolvido privilegiando o mercado dos EUA. A interface disponibilizada inicialmente encontrava-se apenas em Inglês. Com este projecto pretende-se proceder à sua internacionalização, de modo a poder apresentar ao utilizador uma selecção de línguas distintas.

2.2 Requisitos

Desde o início do projecto foi determinado um conjunto de requisitos que a plataforma de internacionalização deveria cumprir. Alguns destes requisitos resultaram de reuniões, na sua maioria informais, com os principais actores envolvidos no processo. Outros restantes requisitos foram elaborados considerando as necessidades de utilização que poderiam surgir e pela análise do estado actual da tecnologia no que respeita à internacionalização de sítios WEB.

Assim, pretende-se que:

- Seja possível efectuar a tradução directamente em templates SMARTY.
- Seja capaz de efectuar a tradução em texto dinâmico, já que a presença de variáveis no interior da frase pode originar erros, em línguas onde os elementos sintáxicos ocupam posições diferentes das ocupadas no Inglês.
- Tenha pouco impacto na legibilidade do código, de modo a não prejudicar a manutenção e as acções posterior de desenvolvimento.
- Seja de simples utilização pois pretende-se reduzir a curva de aprendizagem de futuros engenheiros que venham a incorporar a BLOG.COM.
- Seja pensado para ser utilizado com tradutores externos à BLOG.COM. É necessário o envio do material a traduzir e como tal pretende-se que este seja facilmente portável. É ainda de assumir que os tradutores tenham poucos conhecimentos de informática.
- Utilize UTF-8 para permitir facilmente suportar conjuntos de caracteres diferentes do ocidental.

2.3 Análise tecnológica

Na análise tecnológica foram analisadas diferentes possibilidades de implementação com o intuito de encontrar um sistema que cumprisse com o conjunto de

requisitos recolhidos, e que em simultâneo oferecesse o maior conjunto de vantagens face aos seus concorrentes.

A procura foi efectuada, tendo como base a análise da proposta inicial para a utilização do SMARTYML. Desta análise resultaram algumas das soluções alternativas apresentadas. Outras soluções resultaram da consulta de sítios WEB e fóruns de discussão. Os temas de procura utilizados foram internacionalização, SMARTY e PHP.

2.3.1 SmartyML

O SMARTYML é uma classe PHP desenvolvida por ANDRÉ RABOLD, para adicionar suporte multilingue aos *templates* SMARTY. Foi sugerido inicialmente que esta fosse utilizada como base de partida para a análise de soluções do projecto de internacionalização.

Embora o SMARTYML não disponha de uma página de projecto própria, é possível encontrar o código e alguma documentação no sítio FRONT PAGE - SMARTYWIKI, em HTTP://SMARTY.INCUTIO.COM, e informação adicional no fórum de discussão PHP INSIDER, em HTTP://WWW.PHPINSIDER.COM/SMARTY-FORUM/. Apesar da existência destes locais e do código da classe encontrar-se comentado, começar a utilizar o SMARTYML pode revelar-se algo complicado. A não existência de um tutorial ou de exemplos de utilização pesa sobretudo para os utilizadores que estejam pouco familiarizados com o SMARTY e com a internacionalização de sítios WEB. Nesta situação o melhor recurso é a comunidade de utilizadores PHP e SMARTY.

Para internacionalizar um sítio WEB com o SMARTYML é necessário começar por identificar nos *templates* quais as frases a traduzir, e proceder à sua substituição por variáveis. A nomenclatura utilizada para as variáveis é opcional, sendo apenas necessário que estejam delimitadas por uma sequência de dois cardinais, como se pode ver no exemplo seguinte:

```
<html>
<body>
##NG_HELLO_WORLD##<br>
##NG_MULTILINE##<br>
<input type="button" value="##NG_OK##" onclick="alert('##NG_OK##')">
<input type="button" value="##NG_CANCEL##" onclick="alert('##NG_CANCEL##')">
</body>
</html>
```

Num ficheiro de texto, fazemos depois corresponder essas variáveis ao texto original. Utiliza-se o formato indicado no exemplo seguinte. Este constitui o primeiro ficheiro de tradução, que no caso concreto do BLOG.COM, corresponde ao ficheiro em Inglês:

```
// English language file
NG_OK=Ok
NG_ABORT=Abort
NG_CANCEL=Cancel
NG_MULTILINE=This entry spans 2 lines, this is the first
and this is the second.
NG_HELLO_WORLD=Hello World!<br>How do you do?
```

O nome do ficheiro e sua localização no sistema de ficheiros podem ser configurados. Recomenda-se no entanto, sobretudo nas primeiras experiências com o SMARTYML, que se utilize a estrutura apresentada no exemplo seguinte, pois esta já se encontra inicialmente na configuração:

```
+-languages
 +-eng/global.lng
 |-deu/global.lng
 |-fre/global.lng
```

Este primeiro ficheiro pode depois ser copiado para se proceder às traduções nas restantes línguas. Os tradutores recebem os ficheiros com as frases na língua original, a fim de serem substituídas pela tradução correspondente. O exemplo seguinte apresenta uma possível tradução para Alemão do ficheiro anterior em Inglês:

```
// German language file
NG_OK=Ok
NG_ABORT=Abbrechen
NG_CANCEL=Abbrechen
NG_MULTILINE=Dieser Eintrag umfasst 2 Zeile, diese ist die erste
und diese ist die zweite.
NG_HELLO_WORLD=Hallo Welt!<br>Wie geht es Dir?
```

No código PHP, o SMARTYML é utilizado da mesma forma que o SMARTY. Isto ocorre porque o SMATRYML estende a classe SMARTY, acrescentando-lhe apenas o suporte para a tradução nos *templates*. Torna-se apenas necessário substituir as instanciações da classe SMARTY pelo SMARTYML, o que pode ser feito facilmente com uma ferramenta de procura e substituição do editor de texto ou do sistema operativo.

```
<?php
$smarty = new smartyML();
// [...]
// $smarty->assign(...);
$smarty->display("myTemplate.tpl");
?>
```

Como se pode verificar, a utilização do SMARTYML é relativamente simples, sendo essa uma das suas principais vantagens. No entanto, este representa apenas uma solução parcial para a internacionalização de um sítio WEB, pois é normal que algum do conteúdo a traduzir não se encontre nos *templates*. Pode surgir texto proveniente de outros locais, como do código PHP ou da base de dados. Estas situações ocorrem no BLOG.COM, pelo que seria necessário recorrer a soluções complementares e específicas para resolver cada situação.

O SMARTYML apresenta ainda alguns aspectos que prejudicam a sua utilização em projectos como o BLOG.COM. Refere-se sobretudo o facto de ser necessário substituir o texto por variáveis. Esse processo necessita ser efectuado manualmente e é moroso, mas o aspecto mais nefasto advém do código nos templates perder muita da sua legibilidade. Veja-se um extracto do mesmo exemplo dado acima:

```
##NG_HELLO_WORLD##<br>
##NG_MULTILINE##<br>
```

O significado do texto na primeira linha pode ser mais ou menos óbvio, sobretudo para pessoas ligadas à informática, mas o mesmo não se pode dizer da segunda. Aqui é necessário recorrer ao ficheiro de tradução para verificar o que está a ser transmitido ao utilizador. Esta necessidade de comutar entre os dois ficheiros dificulta a compreensão, manutenção e desenvolvimento de sítios WEB, sobretudo naqueles que necessitem de actualizações frequentes.

Outro aspecto de utilização que ocorre com esta arquitectura é a dificuldade para actualizar os ficheiros com as traduções. Obter o primeiro conjunto de ficheiros é relativamente simples, pois podemos utilizar o ficheiro na língua original como base de tradução, mas a partir desse momento torna-se necessário estar permanentemente a colocar as frases, que entretanto são adicionadas ao sítio WEB, em cada um dos ficheiros das várias línguas, uma vez mais dificultando o desenvolvimento.

A tradução em texto dinâmico, caracterizado por incluir variáveis no corpo da frase, pode também ser efectuada com o SMARTYML, mas exige alguns cuidados suplementares. O problema que normalmente ocorre nestas situações é que a posição da variável na frase pode mudar conforme a língua seleccionada, e como tal é necessário incluí-la no ficheiro de tradução. Para a variável apresentar o valor correcto no resultado final, é necessária a utilização de pré filtros. O código facultado com o SMARTYML já contém um exemplo de como utilizar estes pré filtros.

Pode-se dizer assim que embora o SMARTYML ofereça de facto suporte de tradução ao SMARTY, é pouco prático para uma utilização frequente, e constitui apenas uma resposta incompleta para as necessidades do BLOG.COM.

2.3.2 Pear I18N

PEAR significa "PHP EXTENSION AND APPLICATION REPOSITORY". Como o nome indica é um repositório de aplicações reutilizáveis que estendem o PHP. Para além do repositório, podemos encontrar no PEAR um sistema de manutenção de pacotes, de classificação e distribuição de código, *mailing lists*, *download mirrors* e uma definição de regras e estilos para programação em PHP, numa tentativa para promover código homogéneo e mais compreensível. Na realidade representa toda uma plataforma de desenvolvimento e reutilização de código para apoio à comunidade de utilizadores do PHP.

Na lista de pacotes disponíveis encontramos uma categoria exclusivamente para internacionalização. Esta apresenta dois pacotes, I18N e TRANSLATION, cada um com duas versões distintas, a inicial e outra mais recente que resultou de um *refactoring* significativo da primeira. Embora este *refactoring* seja um sinal de evolução, o próprio autor não garante que a nova estrutura seja mais correcta do que a versão original, para além de que nenhuma das versões apresenta-se num estado de desenvolvimento estável, encontrando-se ambas em *beta*. Embora o termo seja comum, podemos verificar no próprio sítio WEB qual o significado exacto dado a esta classificação pelo PEAR:

alpha Fases iniciais do desenvolvimento, tudo é susceptível de ser ainda modificado, interface ou comportamento, e sem notificação, são esperados muitos erros.

beta A interface não deve já ser modificada. É já utilizável mas deve-se contar ainda com erros.

devel Versão disponibilizada para ser utilizada por programadores. Não faz assumpções quanto à estabilidade do código.

stable O pacote dispõem de uma interface fixa, foi testado e encontra-se documentado.

Esta classificação em conjunto com a informação recolhida do autor levaram a que esta classe não fosse mais considerada como uma hipótese para utilização no BLOG.COM. É aqui apresentada apenas para servir de futura referência, pois dada a reputação de alguns pacotes do PEAR, é normal que com o tempo este venha também a tornar-se um elemento importante na internacionalização de sítios WEB em PHP.

2.3.3 Flaimo's Little Package

Procurando no SOURCEFORGE, um repositório de *download* e desenvolvimento de código e aplicações *open source*, disponível em HTTP://SOURCEFORGE.NET/,

encontramos uma outra possibilidade para internacionalização de sítios em PHP. Denomina-se simplesmente I18N e é uma parte integrante do FLAIMO'S LITTLE PACKAGE. Este último consiste num conjunto de classes para utilização em sítios WEB, sendo apenas algumas destas, especificas para internacionalização. A análise foi efectuada especificamente ao conjunto I18N mas por o nome ser demasiado genérico, referimo-nos a ele pelo nome da colecção, FLAIMO'S LITTLE PACKAGE.

Uma das primeiras vantagens que se destacam ao iniciar a análise deste projecto é a qualidade da sua documentação. O código encontra-se documentado com PHPDOC, uma adaptação do JAVADOC para PHP que apresenta um bom resultado final ao leitor. Esta documentação funciona tanto para as pessoas envolvidas no desenvolvimento das classes como para os utilizadores finais. Para estes últimos existe ainda um pequeno tutorial que permite iniciar a utilização das classes rapidamente.

Outra vantagem é a flexibilidade oferecida quanto ao formato de armazenamento das traduções, sendo opcional a utilização de ficheiros de texto simples, do formato GETTEXT ou de uma base de dados MYSQL.

Das várias classes presentes, as principais a assinalar são as classes LANGUAGE, TRANSLATOR e o ficheiro I18N_SETTINGS.INI de configuração. A classe LANGUAGE é utilizada para determinar os parâmetros genéricos de operação, como o *locale* preferido do utilizador, indicado pelo *browser*, ou caso tal não seja possível, determinar os valores de operação por omissão. No exemplo seguinte apresentam-se de forma resumida os principais elementos para utilização desta classe.

```
include('class.Language.inc.php');
$lg = new Language();
//Metodos para obter os valores do browser do utilizador
$lg->getLocale();
$lg->getLang();
$lg->getCountry();
//Metodos para obter os valores por omissao
$lg->getDefaultLocale();
$lg->getDefaultLanguage();
$lg->getDefaultCountry();
```

A substituição das frases originais pelas traduções correspondentes é feita pela classe TRANSLATOR, que estende a classe LANGUAGE. Para instanciar um objecto desta classe é necessário indicar dois parâmetros. O primeiro parâmetro permite forçar a utilização de uma determinada língua, ou utilizar os valores previamente definidos caso não seja especificado nenhum valor. O segundo parâmetro indica qual a tabela da base de dados, ou ficheiro de tradução, a utilizar, possibilitando manter diferentes traduções para diferentes partes do sítio WEB. Isto

pode mostrar-se útil em situações em que a mesma palavra é utilizada na língua original com dois sentidos diferentes, que correspondem a duas palavras distintas na língua de tradução. Se a opção tomada para armazenamento das traduções for uma base de dados MYSQL, então o campo deve indicar o namespace da tabela, caso contrário colocamos o nome do ficheiro onde as traduções serão armazenadas sem a respectiva extensão.

Existem três métodos alternativos para efectuar a tradução. O método TRANSLATE() retorna simplesmente a tradução encontrada para o texto enquanto os métodos TRANSLATEENCODE() e TRANSLATEENCODEPLUS() retornam o texto já codificado em HTML. O método TRANSLATEENCODEPLUS() converte um conjunto de caracteres mais extenso que o TRANSLATEENCODE(). Para cada um dos três métodos de tradução, são disponibilizados métodos com nomenclatura simplificada para facilitar a sua utilização e melhorar a legibilidade do código. Estes são constituídos por sequências de um, dois ou três traços sublinhados.

```
include('class.Translator.inc.php');
$translation = new Translator('','frontpage, otherpages');
//metodos de traducao
echo $translation->translate('Welcome');
echo $translation->translateEncode('Welcome');
echo $translation->translateEncodePlus('Welcome');
//alias para os metodos de traducao
echo $translation->_('Welcome'); //translate
echo $translation->__('Welcome'); //translateEncode
echo $translation->___('Welcome'); //translateEncodePlus
```

Para utilizar conjuntamente com templates SMARTY, é defendido que a tradução seja feita na camada do PHP e só depois passada como parâmetro ao SMARTY.

```
$smarty->assign('LANG_sidebar', $translator->__('sidebar'));
$smarty->assign('LANG_block', $translator->__('block'));
$smarty->display('template.tpl', $translator->getLocale());
```

No template basta aceder aos parâmetros passados para obter o texto já traduzido.

```
<html><body><p>{$LANG_sidebar} {$LANG_block}</p></body></html>
```

Apesar desta abordagem promova a separação entre camadas lógica e a de apresentação, verifica-se na prática não ser a melhor, sobretudo em sítios que necessitem de actualizar frequente a interface com o utilizador pois a criação das páginas fica bastante dificultada. É necessário ao programador estar constantemente a parametrizar o texto e a colocar código adicional nos ficheiros PHP. O código parametrizado tem ainda como inconveniente a perda de legibilidade, conforme já havia sido mencionado em 2.3.1.

Este conjunto de classes oferece para além do suporte plurilinguístico, a possibilidade de conversão de datas, formatos numéricos e unidades de medição para cada um dos locais configurados. Cada tipo de conversão encontra-se numa classe própria simplificando bastante a sua identificação e utilização.

```
include('class.FormatDate.inc.php');
include('class.FormatNumber.inc.php');
include('class.FormatMeasure.inc.php');
```

O ficheiro de configuração encontra-se bem estruturado e é de fácil utilização. Lá, podemos indicar, entre outros, os locais suportados com a respectiva informação sobre os formatos de datas e unidades, a sua localização no sistema de ficheiros e o método utilizado para armazenar as traduções: texto; GETTEXT ou MYSQL.

```
[Locale]
default_locale = "en"
default_language = "en"
default_country = "us"
locales_path = "../locale"
[Translator]
mode = "Gettext"
default_namespaces = "lang_main"
[FormatDate]
; 0 = standard format, 1 = iso date, 2 = swatch date
default_timeset = 0
[FormatNumber]
default_minor_unit = 2
default_decimal_point = "."
default_thousands_sep = ""
[Measure]
default_input_system = "si"
```

Como resultado final temos um conjunto de classes bem construído, compreensível e de simples utilização. Os aspectos negativos são a falta de suporte para outras bases de dados diferentes do MYSQL. É mesmo indicado que caso seja pretendido, esse suporte terá que ser desenvolvido pelo utilizador, pois não está prevista nem a sua implementação, nem a utilização de uma camada de abstracção no acesso à base de dados que simplifique o desenvolvimento.

As traduções são feitas apenas no código PHP, não sendo oferecido suporte para tradução directa em *templates* SMARTY. A tradução de texto dinâmico também não é facultada, mas podemos simplesmente utilizar a função SPRINTF() do PHP para parametrizar o texto variável e associá-lo à frase após a tradução. De forma semelhante, também não é facultado suporte directo para frases em plural, de modo que se for necessário, teremos que implementar essa funcionalidade.

2.3.4 Aztek

AZTEK não se refere a uma classe ou biblioteca de funções para internacionalização, mas antes ao nome de utilizador de um dos membros participantes nos fóruns de discussão do SMARTY. Um dos temas discutidos é a escolha da melhor forma de construir um sítio plurilinguístico com o SMARTY. Naturalmente, este tema suscitou interesse, dada a relevância que tem para o projecto de internacionalização do BLOG.COM, e levou a uma leitura mais profunda dessa discussão. Uma das primeiras ideias propostas foi feita por este membro, AZTEK, e por ser relativamente simples e versátil foi aqui considerada.

No seu comentário, AZTEK sugere utilizar um sistema de marcação do texto nos templates, implementando para tal um bloco {translate}{/translate}. O texto contido nos blocos é procurado num ficheiro XML e substituído com a respectiva tradução. Esta é uma aproximação minimalista mas constitui a base do que é necessário para a tradução. O bloco permite manter a legibilidade do código do template podendo mesmo ser melhorado, se modificado para {t}{/t}. O XML, por ser um formato que segue uma estrutura de fácil compreensão, e por dispor de um leque variado de ferramentas que operam sobre ele, como por exemplo o XPATH, torna-se uma boa escolha, pois sabemos que caso seja necessário construir alguma ferramenta adicional, o acesso será sempre efectuado de uma forma simples e eficiente. É também um formato que sabemos não apresentar problemas a lidar com UTF-8, que constitui um factor crítico para o projecto.

Esta aproximação é tentadora pois representa uma solução à medida para as necessidades do BLOG.COM. Como todas as soluções *custom made,* a desvantagem é ser necessário todo o seu desenvolvimento desde a raiz, que pode tornar-se mais moroso e eventualmente perder versatilidade face a algumas das soluções já existentes. É previsível que surja a necessidade de criar algumas ferramentas complementares. É o caso de ferramentas que recolham o texto dos templates para os ficheiros XML com as traduções, e que sejam capazes de os manter actualizados. Ferramentas que permitam converter o formato num outro mais simples para os tradutores, de modo a evitar que estes modifiquem elementos que não devem, ou mesmo a construção de um editor próprio para o efeito. Será como tal preferível utilizar uma solução que já disponibilize estas ferramentas sem implicar todo o custo do desenvolvimento.

2.3.5 Smartest Smarty Multilanguage Support

O SMARTEST SMARTY MULTILANGUAGE SUPPORT é um conjunto de funções construídas por TOM ANDERSON que podem ser utilizadas como plugin para facultar suporte plurilinguístico a páginas construídas utilizando o SMARTY. O mesmo autor escreveu um outro artigo denominado SMARTEST SMARTY PRAC-

TICES, onde aconselha um conjunto de práticas para a organização física e configuração global de um projecto PHP utilizando, como no BLOG.COM, templates SMARTY e classes PEAR.

Um dos pontos a favor desta abordagem é a utilização de um bloco SMARTY para a marcação do texto a traduzir. Os blocos SMARTY são etiquetas de marcação semelhantes ás encontradas nas MARKUP LANGUAGES, como o HTML e o XML, mas delimitadas por chavetas em vez dos sinais "maior que" e "menor que".

```
{t}Welcome to Blog.com{/t}
```

O bloco é identificado pela letra t, inicial de *translation*, e representa uma vantagem por praticamente não interferir com a legibilidade do código do template. Este bloco assim denominado passa praticamente desapercebido no código. Outras propostas promovem a utilização do termo por extenso, eventualmente por ser mais compreensível, mas essa abordagem acarreta um maior destaque do bloco no código e retira a atenção dos restantes elementos, que contêm a lógica ou a estrutura da página.

A utilização do bloco de marcação, independente do seu nome ser por extenso ou em inicial, representa uma melhor solução do que a apresentada pelo SMARTYML, pois mantém as frases no template facilitando bastante a sua leitura e compreensão.

Outro aspecto interessante desta solução é a utilização do GETTEXT, já que este é uma ferramenta de tradução robusta, versátil e bem documentada, desenvolvida pela GNU. Esta utilização é feita recorrendo ao suporte para GETTEX nativo do PHP. O facto de os ficheiros guardados serem no formato GETTEXT, é também vantajoso, pois estes foram desenvolvidos considerando as necessidades dos intervenientes na tradução e outros factores relativos ao desempenho. Por o GETTEXT ser amplamente utilizado, existem já desenvolvidas inúmeras ferramentas para edição e manipulação dos seus ficheiros.

Uma das características desta solução é a possibilidade de tradução automática do texto utilizando o sistema de tradução gratuito e online BABELFISH. Para tal, o código implementa dois modos de execução, produção e desenvolvimento, seleccionados numa variável booleana como se ilustra de seguida:

```
$development = true;
```

O modo de desenvolvimento é utilizado durante a tradução. As frases não traduzidas no ficheiro de tradução são verificadas no BABELFISH e se uma tradução existir são adicionadas. Quando a tradução encontra-se completa, o ficheiro é formatado no formato GETTEXT.

A tradução automática pode ser interessante quando utilizada em página pessoais, contudo, a qualidade do texto traduzido pelo BABELFISH está, na presente

data, longe de ser suficiente para a utilização com o BLOG.COM. O autor defende que utiliza essa funcionalidade para criar as primeiras tabelas de tradução para cada língua, corrigindo depois as entradas incorrectas, e assim ganhando tempo. Por ser no entanto previsto que a tradução seja efectuada por profissionais e não pelos elementos do BLOG.COM, este argumento também não justifica a sua utilização, podendo mesmo levar à introdução de erros na tradução.

A criação de tabelas de tradução automaticamente pelo BABELFISH não é no entanto obrigatória e pode ser desactivada através da negação de uma variável booleana própria para o efeito,

```
$use_babelfish = false;
```

ou por manter sempre a função em modo de produção,

```
$development = false;
```

mas isto implica sempre código adicional desnecessário para o que é pretendido no BLOG.COM. Isso não seria um problema se esta fosse considerada uma biblioteca externa não modificável pela equipa do BLOG.COM, mas como é indicado pelo autor, esta encontra-se ainda numa fase inicial de desenvolvimento, sendo de esperar que necessite ainda de várias intervenções para corrigir possíveis problemas que possam surgir.

2.3.6 Smarty Gettext

Esta é uma biblioteca desenvolvida com o intuito de facultar ao SMARTY suporte para a utilização conjunta com o GETTEXT. Pode ser encontrada no SORCEFOURGE, em HTTP://SOURCEFORGE.NET/PROJECTS/SMARTY-GETTEXT/ e pode ser utilizada segundo os termos publicados na licença GNU LESSER GENERAL PUBLIC LICENCE pela FREE SOFTWARE FOUNDATION.

O SMARTY GETTEXT utiliza para marcação do texto a traduzir nos *templates* um bloco SMARTY denominado bloco t, numa abordagem semelhante à utilizada pelo SMARTEST SMARTY, indicada em 2.3.5.

```
{t}Welcome to Blog.com{/t}
```

O bloco t aqui apresentado permite a utilização de parâmetros dinâmicos dentro da frase a traduzir, podendo os seus valores serem depois passados como argumento. Veja-se um exemplo:

```
{t name='John'}Hello %1, welcome back to Blog.com{/t}
```

O nome da variável inserida no texto tem que ser um inteiro, como %1, %2, %3, com um único algarismo. O nome do parâmetro é ignorado, exceptuando se for um dos nomes reservados, conforme iremos ver mais à frente. O que determina a atribuição do valor à variável é a sua posição no texto. O valor do primeiro parâmetro é atribuído a %1, o segundo a %2, prosseguindo consecutivamente até %9.

```
{t 1='Um' 2='Dois' 3='Três'}
O primeiro parâmetro é %1, o segundo %2 e o terceiro %3
{/t}
```

Este formato foi decidido porque a sua sintaxe é mais simples e simplifica a tarefa dos tradutores, limitando a possibilidade destes cometerem erros com as variáveis. O motivo é que a posição dos parâmetros no texto pode variar conforme a língua, e manipular um valor numérico é mais simples e menos propenso a erros do que um nome. O uso de nomes podia também levar a que por algum motivo este fosse traduzido. Por exemplo, mudar %name para %nome na tradução iria introduzir um erro.

O bloco t do SMARTY GETTEXT também permite a utilização com código HTML, o bloco seguinte apresenta ao utilizador uma hiperligação para o sítio WEB do BLOG.COM.

```
{t escape=no url='http://www.blog.com' name='Blog.com'}
<a href="%1">%2</a>
{/t}
```

O parâmetro *escape* evita que os caracteres especiais do HTML sejam anulados, de outra forma o utilizador veria:

```
<a href="http://www.blog.com">Blog.com</a>
```

Outros parâmetros reservados são indicados de seguida:

html permite anular o significado especial de caracteres em HTML, resultando na apresentação do código ao utilizador conforme o exemplo anterior. Este é o valor por omissão.

js anula significado especial de caracteres em JAVASCRIPT.

url anula significado especial de caracteres presentes em endereços URL.

no/off/0 desactiva todas as formas de anulação de caracteres especiais em simultâneo.

plural indica a forma plural da texto a traduzir.

count utilizado determinar o número de elementos presentes e associar a forma plural se necessário.

Para além da tradução em ficheiros SMARTY, é também disponibilizada uma forma de efectuar a tradução em ficheiros PHP, utilizando a função SMARTY_GETTEXT_STRARG(), e uma ferramenta que permite extrair as frases traduzidas e convertê-las em chamadas GETTEXT em C, para construir os ficheiros portáveis do GETTEXT. Estes podem depois ser utilizados com as restantes ferramentas disponibilizadas pelo GETTEXT.

2.3.7 A ferramenta escolhida

Um factor determinante na decisão final, foi a possibilidade de se efectuar a tradução directamente em *templates* SMARTY. Foram ainda privilegiadas as soluções que permitam a utilização do GETTEXT, pois representa uma solução estável, funcional e bem documentada, com o apoio de uma comunidade significativa de utilizadores e diversas ferramentas compatíveis, frequentemente gratuitas.

Neste âmbito, o SMARTY GETTEXT apresentou-se como sendo a solução mais completa. Oferece em simultâneo um método para tradução no código PHP e no SMARTY, incluindo ainda uma ferramenta para automatizar a extracção do texto marcado. A flexibilidade oferecida pela implementação do seu bloco t, suportando facilmente texto dinâmico e formas em plural, foi um dos aspectos de distinção desta solução face às restantes propostas que continham uma abordagem em bloco semelhante.

2.4 PoEdit

Para além das ferramentas centrais ao projecto, discutidas na secção anterior, é comum ser necessário recorrer a algumas ferramentas complementares, de forma a resolver necessidades específicas que por vezes surgem. Neste caso, trata-se de uma necessidade originada pelo GETTEXT, que é a edição dos ficheiros no seu formato portável por pessoas não especializadas em informática, como por exemplo os tradutores. Os ficheiros no formato PO podem ser manipulados com um editor de texto comum, contudo é fortemente aconselhada a utilização de um editor próprio. Estes ficheiros seguem um formato específico e estrito, de modo que o tradutor, se utilizar um editor de texto comum, vai necessitar saltar várias linhas entre cada elemento a traduzir, e ter que utilizar obrigatoriamente a codificação

correcta, que no caso do BLOG.COM será UTF-8. Isto pode facilmente dar origem a erros.

A complexidade do formato PO, comparativamente com os formatos básicos de algumas das restantes soluções analisadas, é na realidade uma vantagem, desde que seja utilizada correctamente, pois incorpora informação adicional que pode revelar-se muito útil, como por exemplo o número de vezes e posições exactas em que cada frase surge no código. Percebemos melhor o potencial da informação contida no ficheiro PO quando utilizamos um dos vários editores específicos para este formato, pois verificamos que a edição torna-se extremamente simples e passamos a usufruir de vantagens que de outra forma não seriam possíveis.

Um dos editores testados, que responde particularmente bem a todas as questões de utilização colocadas, é o POEDIT. Este é um editor de tamanho reduzido, disponibilizado para várias plataformas incluindo WINDOWS e LINUX. Encontra-se disponível para download em HTTP://WWW.POEDIT.ORG sem qualquer necessidade de registo. A utilização do POEDIT mostrou-se simples, com uma curva de aprendizagem bastante rápida, sobretudo porque a interface funciona frequentemente da forma que esperaríamos, resultando numa utilização muito intuitiva.

Normalmente, para a utilização mais simples, não é necessária a consulta do manual, ainda assim, a documentação encontra-se disponível e incorporada na própria aplicação. Nesta, para além da informação relativa ao POEDIT, podemos igualmente consultar documentação sobre o GETTEXT. As informações básicas encontram-se agrupadas numa secção sobre os elementos essenciais de cada um, encontrando nas restantes secções os tópicos mais avançados.

Pelas implicações que podem resultar da não utilização de um editor específico para ficheiros PO, decidiu-se incluir no próprio processo de tradução, uma recomendação aos tradutores para a utilização do POEDIT.

Lista-se de seguida, de uma forma resumida, as vantagens encontradas neste editor:

- Gratuito.
- *Download* sem registo.
- Tamanho reduzido.
- Disponível para várias plataformas incluindo WINDOWS e LINUX.
- Apresenta uma utilização simples e com uma curva de aprendizagem reduzida.
- Bem documentado com informação adicional sobre o GETTEXT

- Permite a editar apenas o necessário e oculta as especificidades do ficheiro PO.
- Detecta automaticamente os problemas mais comuns de utilização, como escolha incorrecta da codificação.
- Organiza os elementos no ecrã de uma forma bastante legível, separando os elementos já traduzidos daqueles que resultaram numa tradução *fuzzy* e dos dos que falta traduzir, dando destaque a estes últimos.

2.5 O processo de tradução

2.5.1 Marcação

O processo de tradução inicia-se com a marcação do texto a traduzir, tanto nos *templates* como nos ficheiros PHP. Nos *templates* utiliza-se o bloco SMARTY t para marcar as frases, sendo recomendada a utilização sempre que possível de parâmetros para remover variáveis e código HTML da frase a traduzir. Com isto pretende-se eliminar a presença de código nos ficheiros enviados aos tradutores.

```
{t}Texto simples{/t}
{t variavel=$var}Texto com uma %1{/t}
{t escape=no 1='<b>' 2='</b>'}Texto com %1HTML%2{/t}
```

O mesmo processo tem que ser efectuado nos ficheiros PHP utilizando a função de tradução criada para o efeito, chamada com a utilização de duplo espaço sublinhado (*underscore*). A redução de código enviado para a tradução é aqui conseguida utilizando a função SPRINTF().

```
$smarty->assign('$var', __('variable'));
$smarty->assign('$note', sprintf(__('%s block selected'),$block_type));
```

2.5.2 Recolha

O texto marcado nos ficheiros SMARTY e PHP necessita agora ser recolhido para construir o ficheiro PO com todos os elementos a traduzir, a estes é necessário adicionar as frases presentes nos ficheiros XML de configuração. Este processo, se feito manualmente, pode ser demorado e levar facilmente a que, por lapso, algumas frases fiquem excluídas. Para simplificar esta tarefa, criou-se uma pequena biblioteca de funções, colocadas em RETRIEVE.PHP, que permitem percorrer ficheiros SMARTY ou PHP, e colocar as frases encontradas no *standard output,* no

formato C reconhecido pelo GETTEXT. As funções criadas baseiam-se no TSMARTY2C.PHP, disponibilizado com o SMARTY GETTEX, e licenciado de acordo com os termos publicados na GNU LESSER GENERAL PUBLIC LICENCE, de modo que é permitido efectuar a sua modificação. Este efectua já a procura nos *templates* SMARTY utilizando uma expressão regular para identificar os blocos t. Para expandir essa capacidade aos ficheiros PHP, bastou simplesmente adicionar a expressão regular que identifica o texto presente nas chamadas à função de tradução utilizada no PHP. Optou-se ainda por simplificar a expressão regular original que efectuava a procura nos *templates* SMARTY, de forma a aumentar a sua legibilidade. O código original,

```
global $ldq, $rdq, $cmd;
preg_match_all(
   "/{$ldq}\s*({$cmd})\s*([^{$rdq}]*){$rdq}
      ([^{$ldq}]*){$ldq}\/\\1{$rdq}/",
   $content,
   $matches
);
```

foi substituído pelo código seguinte

```
//tpl_file
$pattern = '/{t[^}]*}(.+){\/t}/mU';
preg_match_all($pattern, $content, $matches);
(...)
//php_file
$pattern = '/__\(\'([\000-\255]+)\'\)/mU';
preg_match_all($pattern, $content, $matches);
(...)
```

As funções de recolha foram denominadas DO_TPL() e DO_PHP(). A estas foi adicionada uma terceira DO_XML() para recolher as frases dos ficheiros de configuração. Nesta última função optou-se pelo uso do XPATH, que por ser específico para XML, permite uma simplicidade de código e legibilidade maior do que utilizando expressões regulares. O exemplo seguinte ilustra a forma como são obtidos os vários títulos de página colocados no ficheiro de configuração. A mesma técnica é utilizada para os restantes elementos a recolher, como as breadcrumbs.

```
foreach ($content->xpath('//page/title') as $node) {
    print 'gettext("'.fs((string) $node).'");'."\n";
  }
```

Se o ficheiro contendo as funções, RETRIEVE.PHP for executado, ele vai chamar cada uma das três funções sequencialmente, de modo a recolher frases de qualquer um dos três formatos.

Para completar a recolha, recorre-se aos utilitários do GETTEXT para construir o ficheiro PO, neste caso a ferramenta utilizada será o XGETTEXT. A sintaxe de utilização do XGETTEXT é a seguinte,

```
xgettext [option] [inputfile1] [inputfile2]...
```

Não existindo nenhuma opção que possamos activar para obter o texto do *stdout,* torna-se necessário previamente colocar as frases num ficheiro. Isto pode ser feito pela linha de comando, simplesmente direccionando o *output* para um ficheiro temporário, que optamos por denominar I18N.C

```
retrieve.php > /tmp/i18n.c
```

Este ficheiro serve agora como argumento para executar o XGETTEXT.

```
xgettext /tmp/i18n.c -o /tmp/includes.po --force-po --no-location
```

Como pode ser verificado no exemplo, são utilizados alguns parâmetros adicionais do XGETTEXT. Estes vão direccionar o *output* para o ficheiro PO pretendido, forçando a sua criação caso ele ainda não exista, e retirando a informação do local em que a frase a traduzir foi encontrada. Esta última opção não é de todo necessária e visa apenas simplificar o ficheiro PO. De outra forma a informação do local original das traduções, com as suas muitas entradas, tende a tornar-se predominante no ficheiro, aumentando significativamente o seu tamanho e retirando destaque as frases a traduzir.

O XGETTEXT tem várias outras opções que podem mostrar-se de grande utilidade noutras situações. A sua lista completa pode ser encontrada no manual do GETTEXT disponível no sítio Web da GNU[1].

2.5.3 Salvaguarda da informação

As ferramentas que vão ser introduzidas em 2.5.4 podem originar a perda de informação pelo que introduz-se este passo no processo de tradução, de forma a salvaguardar as traduções já efectuadas. A salvaguarda da informação consiste em duplicar os ficheiros PO de cada local encontrado na directoria LOCALE.[2].

```
+-locale/pt-PT/LC_MESSAGES/messages.po
+-locale/pt-BR/LC_MESSAGES/messages.po
|-locale/en-UK/LC_MESSAGES/messages.po
|-locale/en-US/LC_MESSAGES/messages.po
|-locale/fr-FR/LC_MESSAGES/messages.po
```

[1] http://www.gnu.org/software/gettext/manual/gettext.html
[2] http://www.gnu.org/software/gettext/manual/gettext.html

2.5.4 Junção de traduções

O ficheiro PO originado depois da recolha de texto é, do ponto de vista das traduções, um ficheiro PO vazio mas actualizado, ou seja, contém ainda por traduzir todas as frases actualmente no sítio que tenham sido marcadas para tradução, não apresentando já as frases que entretanto foram retiradas.

Por uma questão de redução dos custos de tradução, antes de enviar este novo ficheiro aos tradutores é conveniente aproveitar as traduções previamente efectuadas, de modo a ser apenas necessário traduzir o texto actualizado. De forma semelhante ao descrito em 2.5.2, também aqui podemos utilizar as ferramentas disponibilizadas pelo GETTEXT para obter o resultado pretendido. Temos duas opções, o MSGCAT ou o MSGMERGE, com resultados distintos. Podemos dizer que o MSGCAT executa a união dos dois ficheiros, mantendo todas as entradas presentes em qualquer um dos ficheiros, apenas eliminando os duplicados. O MSGMERGE mantém todas as entradas de apenas um dos ficheiros, enquanto as entradas do segundo ficheiro que não intersectem com o primeiro são descartadas.

```
msgmerge [option] def.po ref.pot
def.po
    Translations referring to old sources.
ref.pot
    References to the new sources.
```

O MSGCAT pode ser utilizado como forma de preservar todas as traduções, mesmo as mais antigas, para a eventualidade de voltarem a ser necessárias. A utilização com os ficheiros do BLOG.COM é feita da forma ilustrada de seguida, onde o ficheiro EMPTY.PO representa o ficheiro recentemente extraído, e o MESSAGES.PO o ficheiro com as mensagens já existentes.

```
msgcat empty.po messages.po -o messages.po
```

A utilização do MSGMERGE é adequada a uma utilização em que pretende-se apenas manter a informação actual, e por uma questão de organização, eliminar todas as entradas obsoletas. Pelas características da operação que efectua, a ordem dos argumentos é importante, podendo originar um resultado bastante diferente do esperado se não for respeitada e causar a perda de informação.

```
msgmerge messages.po empty.po -o messages.po
```

2.5.5 Tradução

As traduções do texto são normalmente efectuadas recorrendo a tradutores externos ao BLOG.COM. Da parte da BLOG.COM, este passo consiste apenas em enviar

os ficheiros PO respectivos e, posteriormente, colocar os mesmos nos directórios próprios para cada língua. Deve ser recomendado aos tradutores a utilização do POEDIT.

2.5.6 Formatação

Para finalizar o processo de tradução, é necessário converter as traduções já efectuadas e colocada nos locais devidos, do formato portável em que se encontram para o formato compilado e utilizável pelo GETTEXT no decorrer da sua execução. A compilação neste formato, denominado *Machine Object*, é feita utilizando o MSGFMT incluído nas ferramentas disponibilizadas no GETTEXT.

```
msgfmt [option] filename.po ...
```

A sua sintaxe é simples, bastando indicar o ficheiro PO a compilar. Dado que o ficheiro PO original não é modificado com esta acção não existe o risco de perda de dados. A única excepção ocorre se por algum motivo não tiverem sido feitas cópias de segurança, conforme descrito em 2.5.3, do ficheiro PO relativo ao MO que vai ser substituído. Neste caso, devemos antes recriar o ficheiro PO a partir da versão compilada actual, utilizando para isso o MSGUNFMT.

```
msgunfmt [option] [file]...
```

Estando em posse de ambos os ficheiros PO, relativos à versão actualmente em produção e à que a vai substituir, podemos prosseguir com a execução da formatação e substituição dos ficheiros.

2.6 Scripts de automatização

Como meio de facilitar e incentivar a correcta utilização do processo de tradução, descrito em 2.5, criaram-se *scripts* em linguagem de SHELL UNIX que o automatizam. O conjunto é constituído por quatro ficheiros:

```
retrieve.sh
append.sh
trim.sh
format.sh
```

Estes encontram-se disponíveis no directório:

```
admin\maintnance\i18n\
```

Os quatro scripts correspondem a várias etapas do processo de tradução, existindo scripts que executam em simultâneo mais do que uma etapa, e algumas etapas que necessitam ser efectuadas manualmente. A escolha dos nomes tenta reflectir a funcionalidade de cada ficheiro, tendo os nomes TRIM e APPEND sido escolhidos com particular cuidado. Dado constituírem duas alternativas para uma mesma operação, conforme irá ser explicado mais à frente, pretendeu-se que o utilizador efectuasse rapidamente a distinção entre eles pelo nome, de modo a escolher o *script* correcto para o que pretende. Tentou-se ainda manter uma sintaxe de utilização semelhante entre eles, sendo necessário normalmente indicar apenas um argumento e este, quando necessário, é sempre a indicação do locale sobre o qual a operação será efectuada.

2.6.1 Automatização da marcação de texto

A etapa inicial, relativa a marcação do texto a traduzir, não foi possível automatizar no período de tempo disponível para essa tarefa. Inicialmente colocou-se essa hipótese dado verificar-se alguns padrões na disposição do texto a traduzir, que aparentemente podiam ser explorados para automatizar esta fase. Os padrões baseavam-se nas etiquetas do HTML já que, ao conhecer a linguagem, é possível determinar à partida onde é que o texto irá ser colocado. A dificuldade surgiu por o HTML não ser uma linguagem estrita, ao contrário do XHTML, e permitir situações em que o código fica incompleto, podendo por exemplo faltar etiquetas de fecho dos elementos de uma tabela sem que isso origine uma falha crítica. Nessas situações, são os navegadores WEB que vão ditar, cada um à sua maneira, como é que os elementos serão apresentados. Por o código do BLOG.COM ter já sofrido várias modificações, ou simplesmente pelo uso do cortar e colar código para facilitar e acelerar o desenvolvimento, verificou-se em diversos locais a existência de código mal construído. Cada uma destas situações originava erros no processo de automatização e necessitavam ser verificadas e corrigidas individualmente. A possibilidade de ignorar este erros foi descartada pois considerou-se preferível aproveitar o processo para em simultâneo corrigir os erros e assim melhorar a estrutura, sendo que após as correcções iniciais a automatização deveria executar sem interrupções.

À medida que os erros do código HTML eram eliminados, começava-se a perceber que algumas das situações não eram originadas por erros de código mas pela utilização do sistema de templates. As principais dificuldades surgiam com o facto do sistema de templates permitir que o código de uma única página esteja disperso por vários ficheiros, e que alguns ficheiros sejam utilizados simultaneamente em várias páginas, assim como o facto de ser possível que o texto a traduzir não esteja

presente nas marcas onde era esperado mas sim parametrizado em variáveis. Outras dificuldades surgiam ainda quando na presença de código javascript embutido no HTML.

Os factores indicados levam a que a marcação automática, mesmo que seja possível ser efectuada, necessite do desenvolvimento de interpretadores próprios, que consigam reconhecer os padrões onde o texto se situa em páginas que utilizem as várias tecnologias em simultâneo. Esse seria sem dúvida um projecto interessante mas cujo tempo de desenvolvimento excedia aquele disponibilizado para a marcação. Isto, aliado ao facto de ser possível efectuar a marcação de todo o projecto em alguns dias apenas, levou a que a possibilidade inicial fosse abandonada e adoptada a marcação manual do texto.

2.6.2 retrieve.sh

O primeiro *script* a ser executado deverá ser o que permite recolher o texto marcado e coloca-lo no ficheiro portável do GETTEXT. Este ficheiro foi denominado RETREIVE.SH e utiliza o RETREIVE.PHP, baseado no TSMARTY2C.PHP. A função do RETRIEVE.PHP consiste em percorre um ficheiro que lhe seja passado como argumento e colocar no STDOUT as frases que encontra para tradução. O *script* vai percorrer todos os directórios onde é previsto encontrar frases a traduzir e passa os ficheiros lá encontrados ao RETRIEVE.PHP, direccionando em simultâneo o output para um ficheiro temporário. Terminada essa tarefa, o *script* formata as frases para o formato portável do GETTEXT, criando o ficheiro *empty.po* no directório LOCALE.

2.6.3 append.sh

Este *script* vai adicionar o conjunto de frases mais recentes encontradas para tradução, presentes em LOCALE/EMPTY.PO, a um dos conjuntos já existentes e previamente traduzidos para os vários idiomas. Para tal, o *script* utiliza o utilitário MSGCAT do GETTEXT. A adição é feita individualmente para cada *locale*, sendo necessário indicar qual o *locale* pretendido como argumento. A sintaxe de utilização encontra-se em comentário no início do código de cada *script*.

```
#usage: append [locale]
#Example append pt_PT
```

Na execução, é criado um novo ficheiro PO, contendo o conjunto completo das frases de ambos os ficheiros de origem. A escolha do nome do *script* reflecte esta característica, sugerindo a anexação dos dois ficheiros num só. O novo ficheiro, denominado MESSAGES.PO, será colocado no local onde se encontrava a versão

anterior seguindo a estrutura utilizada pelo GETTEXT. Antes de substituir os ficheiros, é feita uma cópia de segurança do ficheiro PO antigo. Este fica colocado no mesmo local mas renomeado utilizando a data actual do sistema. Tenta-se assim garantir a unicidade das cópias de segurança e em simultâneo assinalar a data em que a alteração foi efectuada, que poderá ser depois comparada com a data do ficheiro relativa à última modificação para verificar se ocorreram modificações.

2.6.4 trim.sh

Este *script* é uma alternativa ao APPEND.SH, já que utiliza o MSGMERGE em deterimento do MSGCAT, mas efectua todo o restante procedimento, incluindo as cópias de segurança. A diferença final entre os dois, consiste no facto de este descartar as frases presentes nas antigas versões traduzidas mas que já não se encontrem na recolha mais recente. A sintaxe de utilização também é idêntica substituindo apenas o nome do *script*.

```
#usage: trim [locale]
#Example trim pt_PT
```

A escolha do nome *trim*, verbo inglês que significa aparar, tenta ilustrar que algumas frases poderão ser eliminadas, e deste modo distinguir-se do seu *scrip* alternativo APPEND.SH.

2.6.5 format.sh

Tendo os ficheiros portáveis já traduzidos, falta apenas proceder à sua formatação. Para tal podemos recorrer a este *script*, executando-o com o *locale* de destino como argumento.

```
#usage: format [locale]
#Example format pt_PT
```

Esta operação vai criar o ficheiro MO e coloca-lo no directório correspondente ao seu *locale*, ficando pronto a ser utilizado.

Capítulo 3

Estrutura da árvore da aplicação

3.1 Motivação

Uma das decisões a tomar no desenvolvimento de uma aplicação, é a forma como esta vai estar distribuída no sistema de ficheiros. A escolha de uma disposição adequada tem implicações importantes, como por exemplo na segurança e na facilidade de manutenção, dependendo das características de cada aplicação. No caso concreto do BLOG.COM, alguns dos aspectos que tornam importante o planeamento da árvore da aplicação são:

- Questões de segurança, provenientes do facto de ser uma aplicação WEB com um número significativo de utilizadores.
- Ser constituída por um elevado número de ficheiros de vários tipos e função.
- A necessidade de actualizações frequentes, associadas ao dinamismo da INTERNET.
- Previsão de aumento no número de pessoas associadas ao projecto, dado o seu potencial de crescimento.

Ao analisar a estrutura existente, verificava-se que a disposição e nomenclaturas utilizadas para os ficheiros e directórios, não seguiam ainda um conjunto de regras pensado e definido para o efeito. Por exemplo, ao analisar o directório DOCS, verificava-se ser este o directório onde estariam colocados os documentos a servir pelo servidor WEB e, em simultâneo, representava a raiz da árvore da aplicação, onde seriam colocados todos os restantes directórios. Como resultado, obtém-se um directório sobrecarregado, com dezenas de subdirectórios contendo diferentes tipos de ficheiros, dificultando a sua análise e a navegação.

Figura 3.1: Estado inicial da raiz da aplicação

Uma consequência desta organização, era a existência de ficheiros e directórios obsoletos que permaneciam esquecidos no sistema. A sua identificação seria praticamente imediata num directório com poucos ficheiros, mas na estrutura actual, estes permaneciam como que ocultos pelos restantes.

Também verificavam-se questões de segurança, pelo facto de grande parte da aplicação, incluindo ficheiros sensíveis, estar dentro do directório de documentos acessíveis pelo servidor HTTP. Isto vai obrigar a uma maior atenção na configuração, propiciando possíveis falhas na segurança. Era preferível que esses ficheiros fossem isolados fora dos documentos a partilhar, facilitando a tarefa de manter o servidor seguro.

Para além das questões levantadas, a árvore de aplicação existente era partilhada em simultâneo por dois componentes distintos do BLOG.COM, o portal e o editor, e existia já a vontade de promover uma maior separação entre os dois. O portal representa a área de acesso público do BLOG.COM, como a página inicial, FAQs e processo de inscrição. O editor, ou QUILL, representa a parte privada, cujo acesso necessita de autenticação, e onde os utilizadores podem editar os seus *blogs* e artigos. Apesar de ambos apresentarem várias semelhanças, contêm características próprias que justificam a sua separação. Uma destas características é o facto de o portal necessitar de actualizar os seus elementos gráficos com maior frequência do que o editor, que a este nível é mais estável. Por outro lado, a modificação do portal é mais simples do que no editor, pois é neste último que se encontra a quase totalidade das funcionalidades, tornando-o mais propenso à introdução de erros críticos.

Foi assim proposta a reestruturação da árvore de aplicação.

3.2 Descrição do trabalho

Para melhorar a estrutura da aplicação, decidiu-se manter no directório DOCS apenas os ficheiros destinados a serem servidos por HTTP. Os restantes ficheiros foram classificados nas seguintes categorias:

Documentos HTTP Estes são os ficheiros destinados a ser servidos pelo servidor WEB. Podem ser ficheiros PHP, CSS, JAVASCRIPT ou outros. Apenas estes ficheiros devem poder ser acedidos directamente pelos utilizadores, por uma questão de segurança, todos os restantes devem ser excluídos desta categoria.

Ficheiros Smarty Os sistemas de *templates* são utilizados frequentemente para promover a reutilização de código mas também para efectuar a separação entre a camada lógica e a camada de apresentação da aplicação. O sistema de *templates* utilizado no BLOG.COM é o SMARTY. Os seus *templates* são identificados pela extensão TPL.

Por uma questão de *performance,* o SMARTY utiliza adicionalmente ficheiros pré-compilados e ficheiros de cache. A disposição destes ficheiros é facilmente configurável, mas segue normalmente uma estrutura à qual os utilizadores do SMARTY já se encontram familiarizados. Optou-se por adoptar esta disposição por ela constituir um padrão facilmente identificável a quem já tenha utilizado o SMARTY e por ser aquela presente nos tutoriais de utilização.

Ficheiros Gettext O sistema de tradução escolhido, o GETTEXT, utiliza ficheiros com a tradução da língua original para uma língua em particular. A disposição destes ficheiros, identificados pelas extensões .PO e .MO, segue uma estrutura já previamente definida, pelo que apenas localização da raiz dessa estrutura é opcional.

Bibliotecas internas Foram classificados como sendo bibliotecas internas os ficheiros cujo código é destinado a ser incluído ou chamado por outro ficheiro e não pedido directamente pelo servidor WEB. O ficheiro pode conter classes, funções ou código sequencial, normalmente destinado a ser incluído no código de outro ficheiro de forma a promover a sua reutilização em vários locais e evitar duplicações.

Os ficheiros aqui colocados contêm apenas código que tenha sido desenvolvido especificamente para o projecto BLOG.COM. Inclui-se também aqui o

Figura 3.2: Raiz da aplicação reorganizada

código baseado em código externo que tenha sido alterado pela BLOG.COM para adequar-se ao projecto.

Bibliotecas externas Esta categoria difere das bibliotecas internas na medida em que o código não foi desenvolvido nem se destina a ser manipulado pela BLOG.COM. Como exemplos deste tipo de biblioteca temos o JPGRAPH, SMARTY-GETTEXT ou a classe de procura de artigos disponibilizada pela AMAZON.

Em situações em que as aplicações externas possam ser utilizadas simultaneamente em mais do que um projecto no mesmo servidor, ou quando verificar-se que simplifica a sua manutenção, classes como o SMARTY, O GETTEXT ou as classes PEAR, podem ser instaladas fora do directório da aplicação, nos locais onde normalmente estas se encontram. Esta excepção deve ser ponderada individualmente para cada caso.

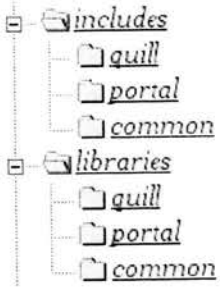
Ficheiros de administração Ficam agrupados nesta categoria os ficheiros que se destinem à administração e manutenção do BLOG.COM. Podem ser ficheiros que necessitem ser executados numa base regular ou esporádica. Um exemplo são os *scripts* criados para automatizar o processo de tradução, que necessitam ser executados apenas quando há necessidade de actualizar as traduções anteriores ou adicionar novas.

Ficheiros de configuração Estes são os ficheiros que vão conter configuração dos parâmetros necessários para o funcionamento da aplicação.

O directório DOCS deixa de ser a raiz da aplicação, tornando-se um subdirectório desta. A estrutura resultante pode ser vista na figura 3.2.

Para a separação do portal e do editor, considerou-se colocar cada aplicação numa árvore distinta, ambas baseadas na nova estrutura encontrada. Essa ideia foi no entanto abandonada pois iria obrigar à duplicação dos ficheiros partilhados por ambas, dificultando as acções de manutenção. Em substituição, estruturou-se

Figura 3.3: Separação dos componentes



cada directório com três subdirectórios, como ilustrado na figura 3.3para conter os ficheiros específicos do portal, do editor e aqueles que são comuns.

Isto permite identificar facilmente, para uma determinada modificação, quais os componentes susceptíveis de serem afectados. Permite ainda identificar quais os ficheiros necessários duplicar caso se torne necessário separar o portal do editor em máquinas distintas.

Capítulo 4

Refactoring

4.1 O que é Refactoring?

REFACTORING é um processo utilizado no desenvolvimento de software com o intuito de melhorar o código já existente, sem que as suas funcionalidades externas, visíveis aos utilizadores, sejam alteradas. As melhorias obtidas não são de carácter funcional, como novas funcionalidades ou melhor desempenho, mas antes qualitativas. O que se pretende com o REFACTORING é tornar o código existente melhor estruturado e compreensível.

O REFACTORING é por vezes utilizado nos processos de engenharia de software, integrado no ciclo de desenvolvimento ou isoladamente e de forma esporádica e espontânea. Integra-se perfeitamente com os métodos de programação ágil, como o EXTREME PROGRAMMING. Este é conhecido por ter como princípio manter ao máximo a simplicidade do código, sendo defendido pelos seus autores que deve-se desenvolver apenas o estritamente necessário para as necessidades actuais. Se a dada altura no futuro for necessário adicionar complexidade, é então possível utilizar o REFACTORING para efectuar as alterações necessárias.

O REFACTORING é também considerado um complemento na utilização de DESIGN PATTERNS. Estes representam possíveis soluções para problemas que surgem de forma recorrente, pretendendo tornar o código mais flexível e reutilizável. A informação para cada padrão detectado é agrupada num conjunto indicando, entre outros, o problema a que se dirige, a alteração a efectuar e as possíveis desvantagens da sua utilização. Um pormenor considerado importante, é a tentativa de se identificar o padrão com um nome sugestivo e de fácil memorização, de modo a que seja criada uma nomenclatura apropriada para discussões que envolvam desenho de software. Os principais autores dos DESIGN PATTERNS são frequentemente eles próprios promotores do REFACTORING, afirmando que os DESIGN PATTERNS são uma ferramenta que ajuda a identificar as

alterações necessárias, e que posteriormente o REFACTORING ajuda a efectuar essas transformações.

O REFACTORING teve a sua origem nas comunidades de programação orientada a objectos (OO), nomeadamente o SMALLTALK, mas rapidamente expandiu-se para as restantes e mais comuns linguagens OO, como o JAVA, C++ e C#. Muito recentemente têm surgido algumas tentativas interessantes para adaptar o REFACTORING às restantes linguagens não OO, sendo talvez a mais avançada a utilização de REFACTORING em C.

É considerado uma parte integral no desenvolvimento de *frameworks*, conjuntos de classes que formam um *design* reutilizável para um determinado domínio de software, capturando-se nelas a lógica do domínio em causa. Nesta situação, é o próprio desenho da aplicação que é reutilizado, e que será comum às várias aplicações assim desenvolvidas, em vez da reutilização de código, que pode ser visto como a abordagem contrária. As decisões relativas à estrutura principal da aplicação deixam de ser da responsabilidade do arquitecto/programador, centrando-se este nas especificidades do seu software. Na reutilização de código, o arquitecto/programador centra-se em determinar arquitectura principal da aplicação, e reutiliza código anterior para as secções mais específicas.

O principal promotor do REFACTORING tem sido MARTIN FOWLER, que ocupa o cargo de cientista chefe na empresa THOUGHT WORKS, especializada em desenvolvimento de software empresarial e consultadoria. Foi pioneiro na tecnologia orientada a objectos, DESIGN PATTERNS, AGIL DEVELOPMENT, UNIFIED MODEL LANGUAGE (UML) e EXTREME PROGRAMMING. O seu livro "Refactoring. Improving the design of existing code" é a obra de referência nesta matéria.

De seguida apresentam-se as definições mais comuns sobre REFACTORING:

Martin Fowler "Uma mudança feita na estrutura interna do software para o tornar mais fácil de perceber e económico de modificar sem alterar o seu comportamento observável".

Kent Beck "Uma mudança no sistema que deixa o seu comportamento inalterado, mas melhora algumas qualidades não funcionais, como a simplicidade, flexibilidade e compreensibilidade".

Tom Mens "Uma transformação de software que melhora a sua estrutura interna sem alterar o comportamento visível".

Wikipedia "O processo de modificar um sistema de software para melhorar a estrutura interna do código sem alterar o seu comportamento externo".

4.2 Porque é necessário?

Podemos dizer que ao desenvolver um projecto de software desde a raiz, é possível obter software bem desenhado e compreensível e assim não ser necessário a utilização do REFACTORING. Se bem que verdadeiro, este facto nem sempre se verifica. Podem-se enumerar vários motivos para tal, sendo talvez os problemas mais frequentes derivados de análises de requisitos insuficientes, prazos de desenvolvimento limitados, ou mesmo a mudança posterior dos requisitos.

Mesmo o software cujo código inicial apresenta um bom desenho, é sujeito a alterações com o decorrer do tempo, seja pelo acrescento de novas funcionalidades ou pela correcção de erros entretanto detectados nas funcionalidades já existentes. Estas intervenções constantes, nem sempre efectuadas pela mesma pessoa, degradam o código e eventualmente a sua estrutura.

MARTIN FOWLER enumera quatro principais razões para o REFACTORING, sendo a principal a melhoria do desenho do software, tendo sido esta a escolhida para título do seu livro. As melhorias obtidas com uma utilização regular da técnica do REFACTORING[1], contrariam o efeito de envelhecimento e degradação naturais do código, que é causado por sucessivas intervenções. O código degradado torna-se menos legível, e mais difícil para perceber-se a estrutura adoptada inicialmente. Isto aumenta a possibilidade de que futuras intervenções venham a degradar ainda mais o desenho, fechando-se assim um ciclo.

Um dos efeitos mais visíveis do REFACTORING é a redução da quantidade de código, vulgarmente pela eliminação de código duplicado. ERICH GAMMA, co-autor do livro DESIGN PATTERNS, indica ser vulgar no meio das discussões das equipas de desenvolvimento de *frameworks*, ouvir os intervenientes a contar histórias de quantas linhas de código duplicado conseguiram remover do código. Esta redução não leva normalmente a um aumento do desempenho, mas tem um impacto significativo para a sua compreensão e manutenção. Quanto mais código existir, mais difícil de morosa é a sua compreensão. Obter código facilmente compreensível é a segunda razão indicada por MARIN FOWLER para o REFACTORING. Como ele próprio indica:

> Qualquer tolo consegue escrever código que o computador compreenda. Bons programadores escrevem código que os humanos podem compreender.

Por sua vez, uma boa compreensão do código torna as acções de eliminação de erros mais simples e eficazes, outra das razões indicadas para utilizar o REFACTORING. Encontrar um erro entre código de difícil compreensão pode ser moroso

[1]Note-se que uma utilização regular não implica necessariamente uma utilização periódica. A utilização do refactoring numa base periódica está longe de ser consensual.

e mesmo frustrante. Efectuar um pouco de REFACTORING ajuda a tornar o código mais simples tornando mais óbvios os erros subjacentes.

Como último ponto, verifica-se que esta prática torna a programação mais eficaz e reduz os tempos de desenvolvimento. Este ponto pode ser estranho à partida, já que estamos a adicionar ao tempo de desenvolvimento o tempo utilizado para REFACTORING mas na realidade é verificado que as perdas de tempo causadas por um mau desenho de código, seja no desenvolvimento de novas funcionalidades ou correcção de erros, não só é superior como pode ser determinante para o sucesso ou fracasso de um projecto.

Resumindo-se o que foi dito, através do REFACTORING podem-se obter principalmente as seguintes vantagens:

- Melhorar o desenho do software.
- Tornar o software mais legível e fácil de compreender.
- Auxiliar à identificação de erros.
- Reduzir o período necessário ao desenvolvimento.

Procurando em artigos de outros autores, podemos facilmente encontrar citações sobre outras vantagens para além das anteriores. A relevância de cada vantagem também irá variar conforme o autor. KENT BECK, um promotor de AGIL DEVELOPMENT e de EXTREME PROGRAMING, valoriza o facto de o REFACTORING reduzir a complexidade do código, enquanto noutro artigo podemos encontrar um determinado gestor de projecto a enunciar mais veemente a redução nos custos de manutenção de software. Tudo depende da perspectiva do utilizador, mas independente disso, encontramos um ponto consensual com a frase:

> Refactoring melhora o design de código existente.

4.3 Quando deve ser utilizado?

O REFACTORING pode ser utilizado numa base periódica, agendando o ciclo de desenvolvimento de modo a alternar-se as fases de desenvolvimento de novas funcionalidades com uma fase de consolidação do código existente. Esta não é contudo a doutrina defendida pelos seus maiores promotores, que propõem antes a utilização do REFACTORING simplesmente quando se verificar a sua necessidade.

Uma das alturas mais propícias para se fazer REFACTORING, é ao tentar compreender o código anterior. Mesmo os melhores programadores reconhecem que essa é frequentemente uma tarefa árdua, em particular quando não se trata do

nosso código. A qualidade e eficácia das acções de desenvolvimento em código já existente, passa por uma boa compreensão desse código, e da correcta percepção da estrutura em que ele se insere. Tentar iniciar o desenvolvimento sem essa compreensão, leva frequentemente à degradação da estrutura anterior. Efectuar REFACTORING durante a análise anterior ao desenvolvimento, com o propósito de tornar o código mais compreensível, torna esse desenvolvimento mais eficiente. Em simultâneo, estamos a introduzir o conhecimento adquirido por quem efectuou a análise no próprio código, facilitando acções posteriores.

Outra boa ocasião para se efectuar REFACTORING ocorre durante o desenvolvimento, quando se tenta introduzir uma funcionalidade e esta parece não se enquadrar bem com o código existente. É mesmo considerado como sendo uma boa prática utilizar o REFACTORING em conjunto com DESIGN PATTERNS, para mudar o código existente de modo que a nova funcionalidade se adapte naturalmente.

A localização de erros pode ser também um bom momento para fazer REFACTORING no código, sobretudo se o erro em causa mostra-se difícil de encontrar. Nesse momento o REFACTORING pode evidenciar o problema, seja por tornar o código mais legível, ou pelas próprias alterações à estrutura. É frequentemente surgirem erros cuja origem provém da duplicação de código. A eliminação desse código duplicado pode por si só resolver o problema ou pelo menos evidencia-lo. Também auxilia na resolução de erros encadeados, em que a resolução de um problema origina um erro imprevisto noutra funcionalidade. Ao melhorar a estrutura global do código, todo ele passa a interligar-se mais naturalmente, reduzindo a ocorrência deste tipo de situações.

MARTIN FOWLER refere também as revisões de código como uma altura propícia para utilizar o REFACTORING. Estando acostumado a utilizar essa técnica nas suas equipas de desenvolvimento, como forma de partilhar conhecimento e criar homogeneidade dentro das equipas, refere que ao utilizar o REFACTORING durante a revisão não só consegue obter um nível mais profundo nas suas sugestões de melhoria, como permite-lhe facultar informação sobre a dificuldade em implementar as mudanças propostas.

Resumindo, listam-se de seguida as alturas em que é sugerida a utilização de REFACTORING:

- Ao tentar compreender código existente.
- Antes de adicionar uma nova funcionalidade.
- Durante acções de *debugging*.
- Durante inspecções formais ao código.

Apesar desta indicação, a regra mais indicada é utilizar o REFACTORING sempre que se sente que há essa necessidade.

4.4 Onde fazer *refactoring*?

Como vimos até agora, a utilização do REFACTORING pode trazer vários benefícios. Vimos também que o REFACTORING não necessita ser utilizado com uma base periódica ou inserido num ciclo de desenvolvimento para ser eficaz. Pelo contrário, os promotores do REFACTORING preferem a noção de utilização oportuna, indicando alguns dos momentos onde essa oportunidade costuma se revelar. No entanto, quando nos encontramos numa dessas situações examinando o código existente, a questão que se coloca normalmente é, por onde devo começar?

O REFACTORING não abrange a totalidade do código, mas apenas os segmentos que necessitam ser modificados. Como é possível identificar esses segmentos? KENT BECK definiu uma regra heurística para este processo. Ele refere a identificação de possíveis locais a utilizar REFACTORING como sendo segmentos de código com "mau cheiro". Alguns destes cheiros podem ser tão subtis que apenas um nariz bem treinado os identificam. Outros podem constituir apenas um mau aroma, não resultando da melhor forma, quando inserido no restante conjunto, ou *bouquet* de aromas, que constitui o programa, mas também não sendo realmente um cheiro incomodativo ou problemático. Outros ainda são cheiros tão nauseabundos que a sua remoção deve ser feita de imediato. Estes últimos, constituem de facto um problema no código. Os maus cheiros não são mais do que padrões que encontramos no código, e que por experiência ou intuição, nos transmitem aquela sensação que algo não está bem e devia ser mudado.

Nesta secção vamos listar e facultar uma pequena descrição dos maus cheiros considerados mais frequentes. A mesma lista, contendo uma descrição mais detalhada, pode ser encontrada no livro REFACTORING de MARTIN FOWLER.

Código duplicado Este cheiro ocorre quando encontramos a mesma estrutura de código em dois ou mais locais.

Método longo Um outro cheiro comum é a presença de métodos longos. Recomenda-se a decomposição destes métodos noutros de menor dimensão, visto aumentar a legibilidade do código e promover a sua reutilização.

Classe grande À semelhança dos métodos longos, também pode-se encontrar classes demasiado grandes. Em vez de atribuir a totalidade do processamento a uma só classe, é preferível decompô-la em classes mais simples e pequenas. Também aqui estaremos a aumentar a compreensão do código e aumentar a sua reutilização.

Longa lista de parâmetros Outra boa oportunidade de melhorar o código, surge frequentemente quando encontramos métodos que recebem muitos parâmetros. Os métodos assim concebidos dificultam a sua utilização e compreensão. As chamadas a estes métodos são extensas e são mais propensas a necessitarem ser actualizadas se o método for alterado. Em linguagens OO é preferível, em vez de passar ao método a totalidade de parâmetros, facultar-lhe apenas o necessário para este poder aceder à informação.

Mudança divergente É chamado mudança divergente quando verificamos a necessário de alterar uma mesma classe duas ou mais vezes por motivos diferentes. É preferível que cada alteração afecte um único e distinto local do código. Ao encontrarmos uma classe que necessita ser alterada frequentemente por motivos diferentes o melhor é criar duas classes, cada uma para responder especificamente à funcionalidade apontada pelas mudanças.

Cirurgia de caçadeira Por oposição á mudança divergente, este cheiro ocorre quando verificamos que um determinado tipo de mudança leva frequentemente a que um grande número de classes sofra pequenas alterações. Para além de tornar a alteração mais difícil e morosa, pode levar facilmente à introdução de erros se alguma das pequenas alterações for esquecida.

Hierarquias de herança paralelas Em determinadas situações, verificamos que sempre que criamos uma subclasse para uma determinada classe, temos também que criar uma subclasse para outra classe paralela. Uma outra forma de identificar esta situação é pelo sufixo[2] dado às subclasses, que tende a ser idêntico.

Inveja de funcionalidades Quando um método parece mais interessado nos dados de outra classe, do que naqueles da classe onde se encontra, então o mais provável é o método estar na classe errada. Nessa altura podemos aplicar MOVE METHOD para colocá-lo na classe mais apropriada.

Moitas de dados Este cheiro surge quando verificamos grupos de dados com tendência a aparecer em conjunto. Nesta situação deve-se considerar o seu agrupamento num objecto próprio.

Obsessão primitiva Refere a relutância que ainda persiste em utilizar pequenos objectos para pequenas tarefas, como o fazem os tipos primitivos. Essa relutância tem origem normalmente na utilização anterior de linguagens mais antigas, onde usar estruturas acarretava sempre algum *overhead.* As linguagens OO praticamente eliminaram esse problema pelo que é recomendada

[2]Ou o prefixo se a nomenclatura utilizada for em inglês

a utilização de pequenas classes em substituição dos tipos primitivos.
Encontramos um bom exemplo desta lógica no JAVA, onde encontramos classes que substituem alguns dos tipos primitivos normalmente utilizados noutras linguagens.

Estrutura Switch A estrutura SWITCH/CASE providencia outro sinal de um potencial *refactoring*. A repetição de um SWITCH em vários locais de código leva ao mesmo problema causado pela duplicação. Quando observamos uma destas estruturas devemos considerar a possibilidade de a substituir por polimorfismo.

Classe preguiçosa Uma classe preguiçosa é uma classe que não faz o suficiente para justificar a sua existência. Essa justificação deveria existir já que cada classe adicionada tem um custo, seja em manutenção ou na compreensão do código. Este tipo de classe pode ser eliminado com COLLAPSE HIERARCHY ou INLINE CLASS para casos de inutilidade extrema.

Legado recusado Verifica-se quando temos classes que não pretendem suportar a interface da sua super classe.

Generalidade especulativa Generalidade especulativa refere o tipo de código gerado quando são acrescentadas funcionalidades para necessidades ainda não existentes. Torna-se o desenho do código mais genérico do que é necessário, baseado nas especulações do que virá a ser necessário no futuro. Isto resulta em classes abstractas que não fazem o suficiente, em delegação desnecessária e em métodos com parâmetros não utilizados.
As situações originadas pela generalidade especulativa devem ser analisadas individualmente sendo os *refactorings* mais comuns COLLAPSE HIERARCHY, INLINE CLASS, REMOVE PARAMETER e RENAME METHOD.

Campo temporário Campo temporário refere uma variável de uma instância que apenas é utilizada em determinadas situações, enquanto no restante tempo aparenta não estar a fazer nada. Usando EXTRACT CLASS podemos organizar estas variáveis numa classe própria para a tarefa que executam melhorando a compreensão do código.

Cadeias de mensagens O tipo de situação em que um cliente pede um objecto a outro objecto, pedindo depois ao objecto que recebeu ainda um outro objecto, é o que se designa por cadeia de mensagem. Estas cadeias podem percorrer vários objectos até atingir o método pretendido.

Classe de dados São classes que apenas têm campos e métodos assessores para esses campos.

Intermediário Designa-se como intermediário o objecto que, para promover a encapsulação, delega o pedido de um cliente noutro objecto. Esta delegação por vezes é efectuada em tal ordem que passa a ser preferível simplificar, e efectuar os pedidos directamente ao objecto seguinte. Nesta situação podemos utilizar REMOVE MIDDLE MAN para remover o intermediário.
Outras situações particulares deste cheiro podem ser solucionadas usando INLINE METHOD para mover os métodos necessários para o cliente ou REPLACE INHERITANCE WITH DELEGATION que transforma o intermediário numa subclasse do objecto.

Intimidade inapropriada Intimidade inapropriada ocorre quando temos duas ou mais classes que usam reciprocamente os métodos da outra em demasia. Nesta situação é conveniente reduzir a dependência mútua. Se as classes estiverem verdadeiramente inter-relacionadas, podemos recorrer a EXTRACT CLASS para colocar os métodos e dados comuns numa classe própria. Outras situações mais simples, podem ser solucionadas utilizando MOVE METHOD e MOVE FIELD, para transferir os dados para a classe mais necessitada.

Biblioteca incompleta Este tipo de situação ocorre quando utilizamos uma biblioteca que faculta apenas parcialmente as funcionalidades necessárias. Nesta situação podemos utilizar INTRODUCE FOREIGN METHOD para completar a funcionalidade que nos falta ou INTRODUCE LOCAL EXTENSION para casos mais abrangentes.

Comentários Aqui, refere-se a necessidade de utilização de comentários, quando provavelmente a criação de métodos com nomes apropriados seria suficiente para tornar o código legível.

4.5 Automatização

Uma característica interessante do REFACTORING é a possibilidade de ser automatizado. Essa possibilidade tem vindo a ser explorada pelos fabricantes de ambientes de desenvolvimento integrados (IDEs) a uma cadência cada vez maior. Estes IDEs introduzem progressivamente uma maior profundidade nos refactorings suportados e obtendo melhores resultados. Este tipo de ferramentas mostra-se extremamente útil, ainda que a utilização manual do REFACTORING seja simples. A automatização permite efectuar os *refactorings* de forma mais eficiente.

A primeira ferramenta de *refactoring* foi o SMALLTALK REFACTORING BROWSER, o que faz todo o sentido já que foi entre os programadores de SMALLTALK

que surgiu o REFACTORING. Desde então já foram desenvolvidos IDEs com suporte de *refactoring* para mais de 10 linguagens diferentes, incluindo o JAVA, C# e C++.

No caso concreto do JAVA, o suporte para *refactoring* é já praticamente um padrão no desenvolvimento de IDEs e *plug-ins* para editores de texto, existindo já mais de um dezena de possibilidades disponíveis, desde soluções gratuitas até IDEs profissionais. Menciona-se de seguida algumas destas soluções:

IntelliJ Idea Um IDE comercial.

Eclipse Um IDE gratuito, apoiado por várias empresas de referência como a IBM ou a NOKIA.

XRefactory Um *plug-in* para EMACS.

JavaRefactor Um *plug-in* para o JEDIT.

RefactorIt Uma ferramenta de *refactoring* que pode ser utilizada individualmente ou como *plug-in* para o NETBEANS, JBUILDER e JDEVELOPER entre vários outros.

Esta lista pretende apenas ilustrar a variedade de ofertas existentes, não pretendendo ser um reflexo das melhores, mais populares ou mais completas soluções existentes.

Apesar de automatizar o processo de transformação do código, a sua utilização continua a necessitar que o utilizador tenha um conhecimento detalhado do processo do refactoring e das suas implicações. As ferramentas não podem garantir que as alterações efectuadas não tenham efeitos inesperados e assim continua a ser necessário elaborar um bom conjunto de testes de validação.

Uma parte do desenvolvimento actual destas ferramentas, que já surge em algumas das versões mais recentes, mas que prevê-se que venha a ser mais explorado futuramente, é a automatização da detecção de maus cheiros, alertando o utilizador para uma possível necessidade de refactoring.

A automatização do REFACTORING é considerada por alguns como o maior passo dado na evolução do REFACTORING desde a sua origem. Não sendo essa uma opinião geral, é no entanto praticamente consensual que este foi um passo lógico da sua evolução.

4.6 Utilização no projecto

Tendo o conhecimento do que é o REFACTORING, podemos agora perceber que ele não constitui um objectivo em si, mas um meio de os atingir. Seria assim normal que a utilização do REFACTORING fosse apresentada incorporada na

descrição das intervenções efectuadas, e não o oposto, contudo, pela importância que o REFACTORING teve na reengenharia do BLOG.COM, optou-se por descrever aqui algumas das intervenções que beneficiaram do REFACTORING, mantendo-se em capítulos próprios, apenas as intervenções de maior relevância para o projecto, ou aquelas não relacionadas com esta técnica.

As acções de *refactoring* são vulgarmente caracterizadas por serem pequenas e ocorrerem em grande número. Por esse motivo, quando utilizado o REFACTORING manualmente, verifica-se não ser eficiente manter um registo detalhado das várias intervenções. Se tal for necessário, é preferível recorrer a software de *refactoring*, como descrito em 4.5. Assim, as intervenções que se descrevem de seguida, não representam a totalidade das situações onde foi utilizado o REFACTORING, sendo antes um conjunto seleccionado para caracterizar os benefícios obtidos pelo REFACTORING e exemplificar a sua utilização.

4.6.1 Eliminação de código duplicado

Durante a análise inicial do código do BLOG.COM, sobretudo para familiarização com a arquitectura utilizada e estilos de programação, verificaram-se várias situações em que o mesmo segmento de código surgia duplicado, que como já mencionado, constítui o principal dos "maus cheiros" indicadores da necessidade de *refactoring*.

Verificou-se que a arquitectura mantinha uma componente de código PHP-OO, a API do BLOG.COM, e outra de código PHP não OO, para alguns ficheiros de inclusão e as própria páginas do sítio WEB. O código duplicado não se encontrava no secção de código OO, situação em que poderíamos aplicar directamente os *refactorings* encontrados na literatura. No entanto, os problemas causados pela duplicação eram aqui igualmente verdadeiros. Numa breve análise, verificou-se ser possível utilizar os *refactorings*, ainda que adaptando-se à situação em causa. Manteve-se as características essenciais do REFACTORING, modificando o código em pequenos passos, seguidos de testes unitários para assegurar que o código mantinha a sua funcionalidade externa intacta e que não eram introduzidos erros.

A resolução descrita para este "mau cheiro" implica extrair o código duplicado para um método de uma determinada classe. Se o código se encontrar dentro de uma mesma classe basta extrair para um método próprio, caso contrário pode ser necessário movimentá-lo para outras classes da hierarquia ou mesmo construir uma nova classe.

Um exemplo desta utilização resultou no método GETBLOGLIST() da classe AUTHOR. Verificou-se existir um segmento de código semelhante em vários ficheiros INDEX.PHP, os ficheiros que representam as páginas do sítio WEB, que através de um ciclo construía um *array* com os nomes e identificador de todos os *blogs* do autor actual e obtinha em simultâneo o nome do *blog* actual.

Iniciou-se a transformação aplicando o *refactoring* EXTRACT METHOD para separar as funcionalidades passando o nome do *blog* actual a ser obtido directamente com a função FETCHBYID() da classe BLOG.

Foi criada uma função em Author denominada GETBLOGLIST() com a funcionalidade do código a substituir. A nova função reutiliza o código de GETBLOGS(), já existente na classe.

Substituiu-se finalmente as várias ocorrências do código duplicado com uma chamada ao novo método e efectuaram-se os testes de verificação.

As alterações ocorreram nos seguintes ficheiros:

1. INCLUDES/API/AUTHOR.PHP
2. DOCS/POSTS/INDEX.PHP
3. DOCS/SIDEBAR/INDEX.PHP
4. DOCS/BLOGROLLS/INDEX.PHP
5. DOCS/BOOKS/INDEX.PHP
6. DOCS/ALBUMS/INDEX.PHP
7. DOCS/ALBUMS/UPLOAD/INDEX.PHP
8. DOCS/CATEGORIES/INDEX.PHP

Neste caso o código encontrava-se duplicado em sete locais distintos, sendo representativo da maioria das situações encontradas. Em algumas ocorrências mais excepcionais, o segmento duplicado podia aparecer em mais de vinte ficheiros distintos, sobretudo nos INDEX.PHP, reflexo da arquitectura adoptada.

4.6.2 Breadcrumbs e Header

A necessidade para outro tipo de *refactoring* surgiu ao verificar-se que em vários templates, o valor de determinados elementos da página, como o seu título, apareciam escritos directamente no código.

O título de uma página é irrelevante para a funcionalidade do sítio WEB, mas constitui um importante elemento de navegação, pois é este que o navegador WEB vai apresentar no histórico das páginas visitadas. Ao escrever o valor do título da página no *template*, todas as páginas passam a apresentar o mesmo título e consequentemente, numa sessão de navegação dentro do BLOG.COM, o histórico das páginas visitadas vai tornar-se inútil, por deixar de ser possível a sua distinção.

Outro elemento de navegação encontrado que mantinha o valor escrito directamente nos *templates* eram as *breadcrumbs*. Estas constituem hiperligações que

Figura 4.1: Histórico de navegação corrigido

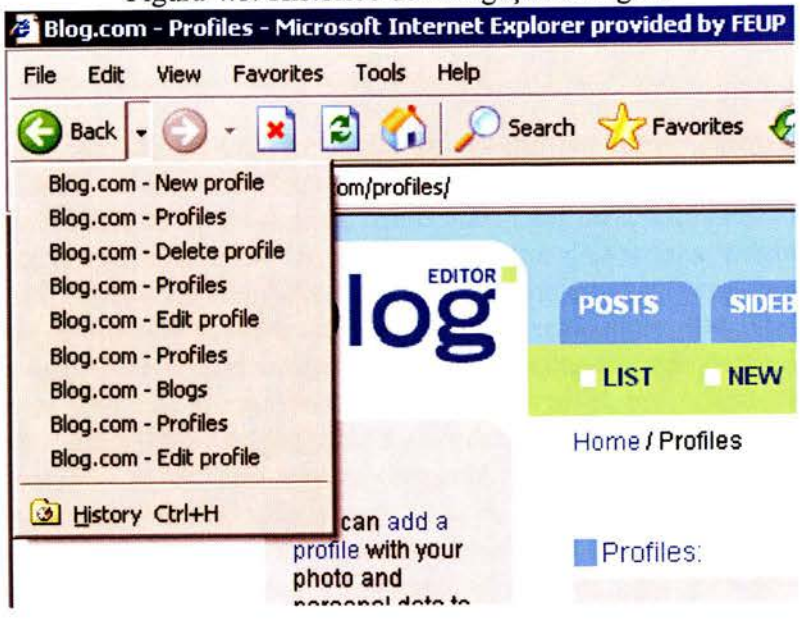


Figura 4.2: Utilização de breadcrumbs

apontam para as páginas anteriores aquela em que o utilizador se encontra actualmente. É importante verificar que esta referência é feita de um ponto de vista lógico, e não histórico, significando que não será apontada a página anterior visitada mas sim a antecessora hierárquica. Por este motivo, a funcionalidade das *breadcrumbs* é distinta e complementar daquela oferecida pelo histórico de navegação dos navegadores WEB.

Pode-se exemplificar a sua utilização ao imaginar uma determinada notícia de âmbito internacional num jornal *online*. Neste exemplo, a composição das *breadcrumbs* poderia ser a seguinte:

```
Nome do jornal->Secção Internacional->Notícia internacional
```

Neste exemplo temos três *breadcrumbs*. A página apresentada ao seleccionar o nome do jornal irá provavelmente permitir navegar pelas várias secções e a Secção Internacional pelas várias notícias internacionais desse dia/semana. Se o leitor saltar directamente para uma notícia de outra secção, digamos economia, o que

ficará nas *breadcrumbs* não será uma referência à notícia internacional lida antes, mas sim à secção Economia.

```
Jornal->Secção Economia->Notícia de economia
```

O problema que surge ao escrever o valor das *breadcrumbs* directamente no código do *template*, é que estamos a fixar a estrutura de navegação. Se por algum motivo for necessário modificar a estrutura, temos que modificar a informação em todas as páginas propiciando erros. Algumas ocorrências desses erros surgiam já em páginas que haviam sido movidas na estrutura mas mantinham *breadcrumbs* para as páginas da estrutura anterior, resultando no conhecido ERRO 404. Verificavam-se ainda *breadcrumbs* com valores errados que resultavam, não de mudanças na estrutura do sítio mas por ser um elemento pouco visível, propiciando a que seja esquecida a actualização do seu valor depois de duplicar o código de uma página para desenvolver as restantes mais rapidamente.

Optou-se por modificar a implementação de modo que estes elementos passassem a ser gerados dinamicamente a partir de um mapa de URLs central, que guardaria a informação relativa a cada página. Para tal, modelou-se as *breadcrumbs* numa classe com o mesmo nome, e o cabeçalho da página numa classe que se denominou HEADERDATA.

Cada objecto BREADCRUMB guarda apenas a informação relativa a uma das hiperligações, e a referência para a *breadcrumb* que a antecede, no caso de não ser a primeira. Estas passam assim a ser representadas numa estrutura em lista ligada de objectos.

Um objecto HEADERDATA guarda nele a informação a associar ao cabeçalho de uma determinada página e faculta métodos para se aceder a essa informação.

A informação passava a ficar fisicamente armazenada no ficheiro XML de configuração, também ele criado no decurso deste projecto de reengenharia para guardar as várias configurações necessárias, como a do SMARTY e a do GETTEXT. A classe CONFIGURATION, já responsável pela leitura do ficheiro, passa a efectuar a instanciação dos objectos necessários.

O tipo de erros antes verificado, como a inexistência de informação pertencente ao cabeçalho da página ou *breadcrumbs* mal associadas, passa a ser notificado na forma de excepções, que ao serem lançadas bloqueiam a navegação, obrigando a que estes dados sejam corrigidos.

4.6.3 Environment

Descreve-se por último a operação de *refactoring* operada sobre o processo de *login* e sobre a manutenção da sessão de navegação que resultaram no objecto ENVIRONMENT. Este teve origem em duas propostas de reengenharia distintas.

Havia sido proposto que o processo de *login*, cujo código encontrava-se distribuído por vários ficheiros, fosse modelado em objectos e encapsulado numa única classe LOGIN, por considerar-se que assim o código torna-se mais legível e mais fácil de modificar. Dado que o *login* era necessário em várias situações distintas, por exemplo, o *login* feito pelo portal e o *login* por utilização de *bookmarklets*, isto iria também promover a reutilização do código e limitar a sua duplicação.

Outra proposta consistia em modelar num objecto denominado SESSION, a sessão de navegação no editor. Seria delegado a esse novo objecto a responsabilidade de restaurar as variáveis e estado da navegação de um determinado utilizador quando esse retornasse ao editor. As propostas foram aceites tendo sido sugerido que ambas as funcionalidades fossem no entanto modeladas num único objecto ENVIRONMENT.

Para implementação utilizou-se o *refactoring* EXTRACT METHOD. Este é vulgarmente utilizado quando o método onde o segmento de código em causa está inserido é demasiado longo, ou quando sente-se necessidade de comentar o código para perceber o que ele faz, sendo que neste caso verificavam-se ambas as situações. Transferiu-se assim o código relativo ao processo de login para a classe respectiva e substituiu-se o código original por chamadas ao novo método.

De seguida efectuou-se o mesmo refactoring para a sessão. Sempre que no decorrer da navegação fosse necessário aceder a informação pertencente à sessão, como o autor seleccionado, o blog actual ou informação da configuração, esses dados seriam obtidos através de ENVIRONMENT, que passava assim a desempenhar um papel de intermediário.

Capítulo 5

Redesenho da barra lateral

5.1 Descrição da barra lateral

Uma das funcionalidades disponibilizadas para a edição dos *blogs* no BLOG.COM, é a possibilidade de personalizar a barra que surge lateralmente aos artigos escritos pelo utilizador. Esta consiste num agregado de blocos sobrepostos verticalmente que permitem apresentar de uma forma organizada, alguma informação adicional aos seus artigos.

O tipo de informação que o utilizador pode colocar na barra lateral, encontra-se limitado pelos blocos disponíveis, existindo no entanto uma variedade de blocos suficiente de modo a garantir um nível de personalização satisfatório para a maioria dos utilizadores. A variedade proposta inclui, entre outros, elementos de procura e navegação no *blog*, referências para outros *blogs* e sítios WEB, publicidade, comentários a livros e álbuns de fotografias.

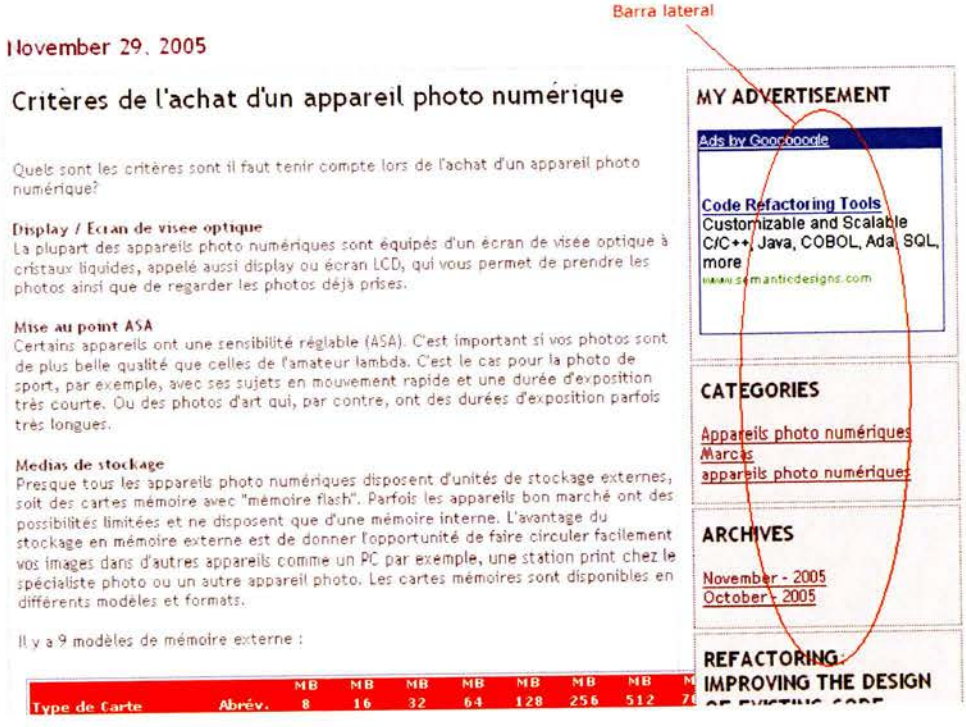
A selecção de blocos é feita no editor do BLOG.COM, na secção SIDEBAR, escolhendo a subsecção LIST. Aqui encontramos uma lista de dez tipos de blocos disponíveis, seguidos de uma caixa de validação. Os blocos validados são os seleccionados para serem apresentados na barra lateral. A edição dos blocos é feita em interfaces distintas, adaptadas ao tipo de informação apresentada.

Inicia-se esta secção com uma apresentação dos diferentes tipos de blocos, mencionando as suas características específicas e formas de utilização.

5.1.1 Books

Este bloco permite apresentar e comentar no *blog*, uma lista de livros seleccionados pelo utilizador. Estes serão dispostos verticalmente num único bloco com o título READING. Nele são apresentados o título e autores de cada livro, seguidos da imagem da capa, terminando com o comentário do utilizador sobre o livro.

Figura 5.1: Barra lateral do Blog.com.

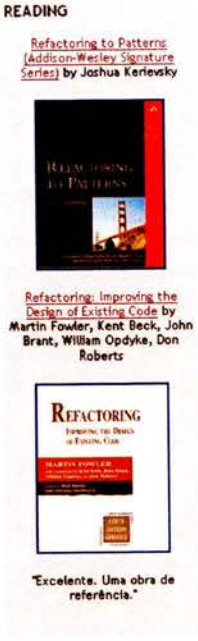


Este comentário é opcional, não surgindo nenhuma referência se o utilizador optar por não o colocar.

Este tipo de bloco só pode aparecer uma vez na barra lateral, ao contrário de outros tipos de blocos que, como veremos adiante, podem ser adicionados múltiplas vezes à barra lateral. Tal característica aqui iria permitir aos utilizadores agruparem os livros da forma que pretendessem, fosse por tipo, assunto, autores do livro, ou outras classificações possíveis, e não ter que apresentar todos em conjunto.

A edição do bloco é feita a partir de SIDEBAR ¿ BOOKS. Para adicionar um livro devemos indicá-lo no campo com a descrição ADD BOOK e carregar em SEARCH. É feita automaticamente uma procura na AMAZON e os títulos que coincidirem com o termo de procura serão listados. A procura é versátil, e pode ser efectuada utilizando apenas uma parte do título do livro, um autor, o código ISBN ou o ASIN. Se o livro pretendido se encontrar na lista, o utilizador deve seleccionar a opção ADD que surge em frente do código ASIN para adicioná-lo ao seu bloco. Apenas 30 resultados serão apresentados. Se a procura originar mais do que este número de resultados e o livro não se encontrar na listagem, será

Figura 5.2: Bloco de livros inicial.



necessário efectuar uma nova procura, tentando ser mais específico sobre o livro pretendido. A procura pelo ISBN é uma forma de evitar a situação descrita. Esta informação não é no entanto transmitida ao utilizador, que pode facilmente ser induzido a pensar não existirem mais resultados. Existe ainda a possibilidade de o livro não constar na base de dados da AMAZON, sendo que nessa situação não pode ser adicionado.

Ao seleccionar ADD, o utilizador é reencaminhado para a página de edição do livro, podendo modificar o título, autor e adicionar um comentário. Uma última opção denominada ON BLOGROLL surge numa caixa de validação. Ao retirar o visto desta caixa o livro deixará de ser apresentado no bloco de livros.[1] Caso seja necessário editar novamente os dados deste livro, podemos seleccionar EDIT na listagem ou simplesmente seleccionar o título do livro.

Para retirar um livro da selecção basta seleccionar a opção DELETE que surge à direita de EDIT. É importante verificar que não é pedida confirmação sobre se o utilizador pretende de facto efectuar essa operação e não existe qualquer forma de recuperar a informação depois de apagada, pelo que é necessário exercer algum cuidado na sua utilização.

[1] O nome do campo pode induzir em erro ao levar a crer que o livro vai surgir listado num Blogroll quando na realidade refere-se ao bloco de livros.

5.1.2 Syndication

A sindicação de artigos teve origem na imprensa dos E.U.A, onde os autores entregavam os seus artigos, ou tiras de banda desenhada, a empresas especializadas em apresentar o seu conteúdo no maior número de publicações possíveis. Na INTERNET, o termo também pode ser utilizado para referir o acto de se disponibilizar um determinado conteúdo, por vezes licenciado, para apresentação noutros sítios WEB. Mais frequentemente, refere a utilização de WEB FEEDS num sítio WEB, para disponibilizar um método de listar o seu conteúdo fora do sítio WEB e automaticamente mantê-lo actualizado. É vulgarmente utilizado em sítios de agências noticiosas, fóruns de discussão e *blogs*.

Esta opção seleccionada vai apresentar um bloco na barra lateral com hiperligações para os protocolos de sindicação suportados, RSS e ATOM.

5.1.3 Recent Comments

No BLOG.COM, é dada aos visitantes dos *blogs*, a possibilidade de exprimirem a sua opinião sobre os conteúdos visitados nos próprios *blogs*. Para tal surge debaixo de cada artigo uma hiperligação denominada COMMENTS, e junto dele a informação de quantos comentários existem para aquele artigo. Essa opção só existe se o autor do *blog* assim o desejar, podendo ser desactivada para todo o *blog* ou individualmente para cada artigo.

Num *blog* com um número razoável de artigos e frequentemente visitado pode tornar-se difícil, para o autor e leitores interessados, manterem-se actualizado sobre os comentários colocados. Foi assim dada a possibilidade ao utilizador de colocar um bloco na sua barra lateral com uma listagem dos comentários mais recentes. Esta listagem é feita automaticamente ao carregar a página, bastando activar o bloco para este ser desde logo apresentado. Uma excepção surge quando o blog não tem comentários, pois nesta situação mesmo activo o bloco não será apresentado. Esta foi a forma escolhida para evitar colocar blocos vazios na barra lateral. Não existe a possibilidade de qualquer configuração adicional[2], e como tal, o bloco não dispõem de uma subsecção própria.

5.1.4 Search

O bloco SEARCH introduz na barra lateral um campo de pesquisa, onde é possível procurar a ocorrência de um determinado texto nos artigos do *blog*. Os artigos resultantes da procura são apresentados na sua forma integral e não apenas em forma de lista com as ocorrências. Também não é feita paginação quando a procura retorna um número significativo de resultados, o que torna a apresentação

[2]Como seria por exemplo escolher o número de comentários recentes apresentados.

dos resultados bastante extensa, especialmente se os artigos apresentarem tamanhos significativos.

Este bloco não apresenta qualquer opção de configuração adicional.

5.1.5 Blogrolls

Um *blogroll* é um conjunto de hiperligações para outros *blogs*. O autor de cada *blog* tem a possibilidade de manter um ou mais *blogrolls* na sua barra lateral, com uma selecção dos seus *blogs* favoritos.

A forma actual de edição dos *blogrolls* foi simplificada relativamente a versões anteriores, como resultado da redução de funcionalidades. Para adicionar um *blogroll*, o utilizador necessita activar o tipo de bloco na subsecção de listagem de blocos, mudar depois para a subsecção BLOGROLLS e, para cada *blogroll* pretendido, adicionar o seu título na opção NEW BLOGROLL. Os *blogrolls* criados surgem listados na página, seguidos de opções para renomear, gerir e apagar cada *blogroll*. Por último, é necessário aceder à opção de gestão e adicionar endereços, já que um *blogroll* vazio não surge na barra lateral.

Ao aceder à gestão de um *blogroll* temos a possibilidade de adicionar endereços WEB individualmente ou importar directamente uma colecção destes a partir de um ficheiro no formato OMPL (OUTLINE PROCESSOR MARKUP LANGUAGE) baseado no XML. Para introduzir-se os endereços individualmente, é necessário preencher o título e o endereço do *blog* ou sítio WEB, e premir ADD BLOG para terminar a operação. Para utilizar um ficheiro OPML indicamos a sua localização em OPML URL e carregamos IMPORT.

5.1.6 Monthly Archives

Este bloco, quando activado, permite aos visitantes de um *blog* aceder directamente aos artigos escritos num determinado mês. Os meses que contêm artigos são listados verticalmente, do mais recente, colocado no topo, até ao mais antigo. Esta é a única apresentação possível pelo que a listagem pode tornar-se algo comprida, em particular no caso de *blogs* que permaneçam activos por vários anos, tornando o bloco algo desproporcional relativamente aos restantes. No entanto, pode ser argumentado que isto verifica-se apenas num número reduzido de *blogs* e que a não existência de configurações adicionais facilita a utilização nos restantes, sendo a simplicidade de utilização um aspecto chave para o sucesso da BLOG.COM.

5.1.7 Powered

Um bloco com publicidade ao BLOG.COM. A sua apresentação é obrigatória nos *blogs* gratuitos e opcional nos restantes. Os visitantes de um *blog* podem premir no logótipo do BLOG.COM para acederem à página de subscrição de novos utilizadores ou à opção RANDOM BLOG para continuar a navegar pelos *blogs*. Esta última funcionalidade utiliza internamente um conjunto de regras para tentar apresentar apenas *blogs* que estejam activos, de modo a tornar a navegação aleatória mais agradável.

5.1.8 Advertisement

Um bloco de publicidade externa. À semelhança do bloco POWERED, a sua apresentação é obrigatória nos *blogs* gratuitos, sendo uma forma de rentabilizar a sua utilização. Os *blogs* pertencentes a contas não gratuitas, por considerar-se já terem pago o valor do serviço prestado, têm a possibilidade de removê-lo. Esta possibilidade funciona como um incentivo para passar de uma conta gratuita para uma conta paga.

5.1.9 Category List

Os autores de artigos no BLOG.COM têm a possibilidade de os organizar em categorias. Esta categorização é feita na criação ou edição de um artigo, seleccionando a categoria pretendida das que surgem listadas na opção CATEGORY ou escolhendo da mesma lista a opção para criar nova categoria e preenchendo o campo seguinte, denominado NEW CATEGORY com o nome da categoria a criar.

Enquanto que a criação de novas categorias pode ser feita a partir da secção de artigos, toda a restante gestão, incluindo renomear e apagar categorias, encontra-se na secção da barra lateral. Esta escolha, apesar de aparentemente não ser a mais intuitiva, justifica-se pelo facto de a categorização surgir apenas no bloco CATEGORY LIST, e por ter-se tentado agrupar nessa secção toda a gestão da barra lateral. As opções dadas para gestão de categorias limitam-se, no entanto, à criação, remoção e renomeação das categorias, e não incluem qualquer funcionalidade que permita gerir a forma e ordem em que elas vão ser apresentadas na barra lateral.

Não é possível gerir cada categoria individualmente, como por exemplo, para verificar quais os artigos que lhe estão associados. Para tal é necessário percorrer cada um dos artigos e consultar qual a categoria associada. Se esta funcionalidade estivesse presente, a edição deste bloco poderia ser feita de forma semelhante à gestão dos *blogrolls*, sendo os artigos nas categorias o equivalente aos hiperligações nos *blogrolls*.

5.1.10 Album List

Este bloco é semelhante aos blocos dos *blogrolls* e das categorias, sendo que os elementos aqui listados são os álbuns de fotografias. Os álbuns são conjuntos de fotografias, da mesma forma como uma categoria é um conjunto de artigos.

A principal diferença encontrada é na utilização. Estranhamente, e contrariando a escolha feita para as categorias e *blogrolls*, a gestão dos álbuns não é feita na secção da barra lateral mas sim na secção PHOTOS, o que leva esta secção a também apresentar algumas características invulgares. Enquanto na secção POSTS são listados os artigos, que constituem o elemento mais pequeno, na secção PHOTOS o elemento listado é o álbum de fotografias. São abordagens opostas e que podem dificultar a utilização inicial do editor de *blogs*.

Assim como o bloco de categorias, também aqui não é dado ao utilizador qualquer controlo sobre a forma ou ordem em que os álbuns serão apresentados no bloco.

5.1.11 Profile

O bloco PROFILE permite colocar na barra lateral uma pequena apresentação do autor do *blog* aos seus leitores, podendo conter uma fotografia, alguns dados pessoais, como o nome, idade e localização, e ainda campos opcionais personalizáveis, para colocar dados adicionais que pretenda partilhar.

Os perfis mantêm uma utilização distinta dos restantes blocos. A diferença mais evidente é a ausência deste bloco na listagem que surge na subsecção MANAGE. Isto ocorre porque, ao contrário dos restantes blocos, existe a possibilidade de colocar o mesmo perfil simultaneamente em vários *blogs*. Assim havias sido decidido que um perfil seria associado aos *blogs* dentro da edição do próprio perfil, onde seriam também listados os vários *blogs* ao qual actualmente estivesse associado.

5.2 Motivação

Da análise efectuada à interface de edição do BLOG.COM, resultou um conjunto de referências a situações de utilização incoerentes. Muitas destas situações eram originadas pela evolução natural da interface à medida que as funcionalidades subjacentes vão sendo alteradas. Este processo de degradação progressiva apresenta semelhanças com aquele sofrido ao nível do código, e que constitui normalmente a principal motivação para as acções de refactoring. A questão aqui levantada é o custo adicional de aprendizagem que o utilizador vai ser sujeito ao utilizar uma interface com comportamentos diferentes nas várias secções, contrapondo com o

tempo que levaria se estas fossem mais homogéneas, considerando o facto de o modelo de negócio da BLOG.COM é o de tornar simples a publicação *online*.

Notava-se ainda que as várias secções apresentavam-se heterogéneas quanto à utilização das subsecções. Na secção BLOGS e PHOTOS encontravam-se apenas duas subsecções, nomeadamente para criar e listar os elementos nelas contidos. A secção de POSTS mantinha esta estrutura, mas tinha sido acrescentada uma funcionalidade para adicionar ao navegador WEB um *bookmark* para criação rápida de artigos e um campo para procurar texto nos artigos. Estas funcionalidades não eram na realidade subsecções e sugeria-se a sua remoção para um local mais apropriado. A estrutura lógica verificada nas secções mencionadas simplesmente não se aplicava à secção da SIDEBAR, cujo comportamento era distinto, apesar de se referir também à edição de elementos do *blog*. Em vez disso eram apresentados ao utilizador cinco subsecções:

- MANAGE
- PROFILES
- BLOGROLLS
- BOOKS
- CATEGORIES

MANAGE era a subsecção aberta por omissão, e permitia gerir os blocos que iam aparecer na sidebar, através de uma lista de caixas de validação. As caixas de validação seleccionadas eram apresentadas com um visto no interior significando que o bloco correspondente seria apresentado na barra lateral. As várias opções não estavam dispostas em nenhuma ordem particular. Assim o facto de a primeira opção ser BOOKS não significava que este fosse o primeiro bloco apresentado na barra lateral. As restantes quatro subsecções destinavam-se a blocos personalizáveis. Os restantes blocos cujo conteúdo não é definido pelo utilizador, por não necessitarem de configuração, não apresentam qualquer subsecção.

Esta disposição leva a que, para efectuar certas acções, o utilizador necessite aceder a secções distintas. Isto ocorre, por exemplo, para a acção de colocar um dos blocos configuráveis visíveis. O utilizador necessita para tal assinalar o bloco para visualização e configurá-lo em locais distintos. Seria preferível poder-se efectuar tudo no mesmo local.

Algumas das alterações propostas para a barra lateral eram controversas, por envolverem questões de usabilidade de interfaces algo subjectivas, outras situações contudo, não levantavam dúvidas quanto à necessidade de serem corrigidas. É o caso da falta de retorno de informação ao utilizador em determinadas situações ou

de não ser pedida confirmação em acções que o exigiam. A falta de informação de retorno era necessária, por exemplo, ao assinalar um bloco ainda vazio para visualização, para indicar porque esse bloco não iria ser apresentado. O mesmo ocorria ao acrescentar um livro, se a procura efectuada na AMAZON originasse mais do que trinta resultados, não era indicado ao utilizador que a consulta tinha sido demasiado genérica e que parte dos resultados haviam sido omissos. A falta de pedido de confirmação ocorria, por exemplo, ao remover um livro conjuntamente com os comentários que o utilizador pudesse ter feito sobre o mesmo. Uma vez que poderia ocorrer a destruição definitiva de informação, justificava-se questionar o utilizador se pretendia de facto efectuar essa acção.

Dadas as questões de utilização levantadas, foi sugerida uma intervenção para sua correcção e, em simultâneo, serem incorporadas algumas novas funcionalidades sugeridas.

5.3 Descrição do trabalho

O redesenho da barra lateral foi feito em vários passos. Partiu-se de uma visão mais global da aplicação, diminuindo depois a abrangência apenas à própria barra lateral e por fim a aspectos de pormenor.

A análise feita no nível de abstracção superior, enquadra a barra lateral com os restantes elementos do BLOG.COM e tenta eliminar a necessidade de manter uma forma de edição distinta para cada secção, tornando toda a aplicação mais coerente. Uma análise semelhante é feita depois, considerando apenas os elementos internos à barra lateral, verifica-se aqui porque certas características surgem apenas em alguns blocos, e tenta-se promover a coerência da própria barra lateral. Por último, são verificadas questões relativas aos blocos individuais, que visam melhorar a sua utilização e corrigir alguns problemas identificados.

5.3.1 O objecto bloco e o polimorfismo

O primeiro passo dado, foi no sentido de tentar uniformizar a interface de edição da barra lateral com a das restantes secções. Para tal procurou-se identificar quais as características partilhadas pelas várias secções que pudessem ser utilizadas, assim como as diferenças fundamentais que fossem necessárias preservar.

Verificou-se que a subsecção MANAGE podia ser vista como sendo a subsecção LIST, presente em todas as outras secções. Era já a secção aberta por omissão e na realidade listava os vários elementos existentes daquela secção. A principal diferença residia no tipo de elementos listados, que implicitamente afectava toda a sua estrutura orgânica. Havia sido denominada POST, a secção que apropriadamente listava os artigos, BLOGS onde listavam-se os *blogs* e PHOTOS para as

Figura 5.3: Secção barra lateral reorganizada.

Blocks:

CLAUDIO GOMES --choose a blog--

TITLE	TYPE	STATE	POSITION	ACTIONS
Search	Search	Visible	Fixed position	Edit | Delete
Advertisement	Advertisement	Visible	Fixed position	Edit | Delete
Monthly Archives	Monthly Archives	Visible	Bottom | Down	Edit | Delete
Literatura obrigatória	Book	Visible	Up | Down	Edit | Delete
Para saber mais	Book	Visible	Up | Down	Edit | Delete
Syndication	Syndication	Visible	Up | Down	Edit | Delete
Powered	Powered	Visible	Up | Down	Edit | Delete
Random Blog	Random Blog	Visible	Up | Top	Edit | Delete

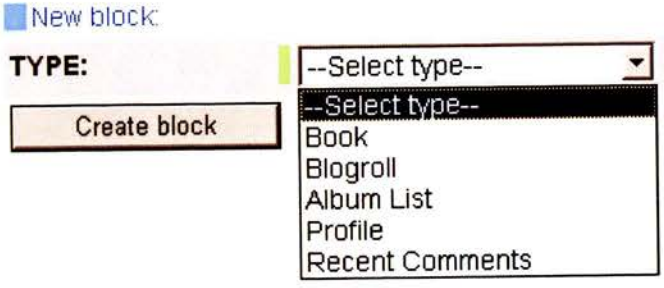
fotografias. SIDEBAR constituía uma excepção, pois não eram listados blocos mas sim tipos de blocos. Por exemplo, podiam ser criados vários *blogrolls* mas apenas uma entrada seria listada em MANAGE.

Se optássemos por uma outra abordagem, onde fossem listados os próprios blocos existentes, em vez dos seus tipos, a gestão da barra lateral ficaria semelhante à das restantes secções. As opções apresentadas como subsecções seriam as mesmas das restantes secções, LIST para listar os blocos existentes e NEW para criar novos blocos.

A diferença fundamental é que, ao contrário dos elementos das restantes secções, os blocos apresentam características distintas entre eles, e como tal, para a sua criação e edição, são necessárias interfaces específicas para cada tipo de bloco.

Ao nível do código, recorreu-se ao polimorfismo para representar os vários tipos de blocos, o que implica que na subsecção NEW da interface planeada, seja necessário questionar o utilizador sobre qual o tipo de bloco que pretende criar.

Figura 5.4: Selecção do tipo de bloco a criar.



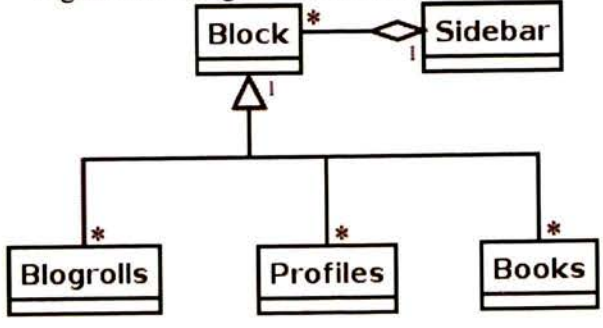
Se o tipo de bloco seleccionado não necessitar de configuração adicional, pode ser imediatamente criado e adicionado à listagem, caso contrário, é apresentada a interface própria de edição à qual o utilizador pode mais tarde voltar para editar seleccionando EDIT na listagem.

O redireccionamento automático, da subsecção NEW e da opção EDIT, para a página com a interface específica de alguns blocos, passa a ser a única diferença de utilização perceptível ao utilizador. Consegue-se assim reduzir o número de subsecções necessárias, tornar a sua utilização idêntica à das restantes secções e tornar as diferenças de edição de blocos de tipos distintos, não inteiramente invisível, mas pelo menos mais transparente ao utilizador.

A intervenção foi planeada de forma a primeiro efectuar as mudanças internas no código e manter a interface com o utilizador inalterada, após o qual seriam alteradas a interface do editor e a própria barra lateral que surge no *blog* para reflectir as mudanças feitas.

O desenho idealizado incluía uma classe SIDEBAR que agrega um ou mais objectos do tipo BLOCK. A classe BLOCK seria estendida para reflectir os objectos especializados como os BLOGROLLS e os PROFILES.

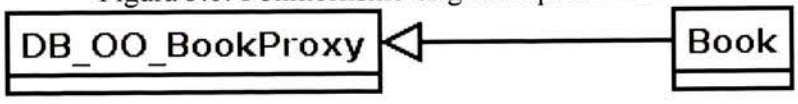
Figura 5.5: Diagrama de classes idealizado.



Este desenho não podia ser integrado na estrutura já existente tal como se apre-

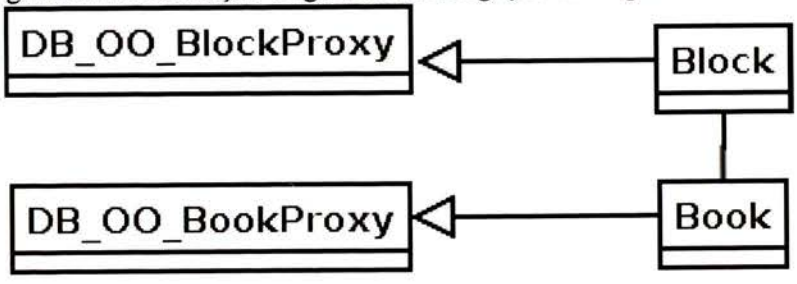
sentava, pois dado o desenho actual da aplicação, cada objecto com informação na base de dados necessitava estender uma *proxy* DB_OO, cuja utilização pretendia-se manter.

Figura 5.6: Polimorfismo originado pela DB_OO.



Optou-se assim por utilizar a delegação de responsabilidade entre objectos para representar o polimorfismo pretendido, de forma semelhante à que seria feita numa linguagem que não suportasse de raiz este padrão.

Figura 5.7: Associação originada na delegação de responsabilidades.



O objecto bloco foi construído normalmente, de forma a conter os dados e métodos comuns a todos os blocos. Criaram-se então objectos próprios para cada tipo de bloco que fosse necessário especializar, contendo os dados e métodos desse tipo específico.

O relacionamento entre os blocos pai e filhos é feito pelo identificador na base de dados. O bloco pai é criado com um identificador único e sequencial atribuído pela base de dados, que é depois utilizado para criar os blocos filhos.

Recorre-se a restrições na base de dados e à utilização de transacções na criação e eliminação dos objectos compostos para manter a sua coerência. Para exemplificar, tome-se como exemplo a eliminação de um bloco de *blogrolls*. Este é representado pelo objecto que guarda a informação relativa ao facto de ser um bloco, e por um outro objecto que guarda a informação específica dos *blogrolls*. Na base de dados a informação encontra-se armazenada em duas tabelas distintas, em linhas com o mesmo identificador. O segmento de código PHP que efectua esta operação está contido numa transacção, pelo que ou ambos os objectos são apagados, como é pretendido, ou caso algum erro ocorra, ambos vão falhar em simultâneo. Também não será possível eliminar apenas uma das linhas na base de

dados, ao tentar iremos receber um erro de SQL indicando que estamos a violar uma das restrições colocadas.

O acesso das novas classes aos dados na base de dados é feito recorrendo à classe DB_OO, respeitando o desenho de padrões global do BLOG.COM.

5.3.2 Blocos de livros múltiplos

Ao implementar os livros, optou-se por acrescentar a possibilidade de existirem vários blocos de livro em simultâneo, sendo que cada bloco contínua a poder conter vários livros. Isto vai permitir aos utilizadores agruparem os seus livros por temas e poder manter vários blocos de dimensão razoável, em vez de um único, com dimensões desproporcionais aos restantes. Esta característica, de poder manter vários blocos do mesmo tipo, já ocorria nos *blogrolls*, pelo que os dois tipos de blocos passam a ter um comportamento mais semelhante.

Outra alteração efectuada, que visa oferecer mais flexibilidade ao utilizador e promover a coerência na utilização dos blocos, foi tornar o título do bloco de livros personalizável. Este apresentava forçosamente o título READING que agora passa a ser escolhido pelo utilizador, permitindo-lhe nele indicar, por exemplo, qual o tema dos livros apresentados no bloco.

Foi ainda alterada a questão colocada na caixa de validação que permite tornar o bloco visível, de modo que onde antes aparecia ON BLOGROLL, passasse a constar SHOW BLOCK IN THE SIDEBAR? por considerar-se mais explícito sobre a função daquela opção.

Optou-se ainda por desactivar esta opção sempre que o bloco de livros estiver vazio, já que nessa situação ele não vai ser apresentado. Em substituição da opção para tornar visível o bloco, passa a surgir sempre que o bloco estiver vazio, uma mensagem a avisar o utilizador do estado actual, sugerindo-lhe a adição de um livro.

Por último, criou-se uma mensagem de aviso aos utilizadores quando as suas procuras de livros ultrapassam o limite de livros permitido.

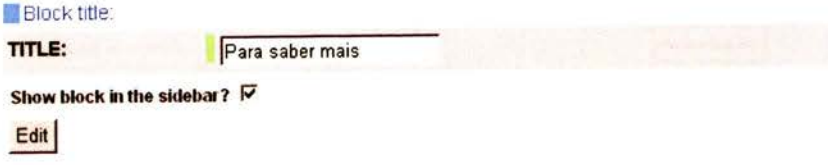
Para implementar a multiplicidade de blocos, foi criado o objecto BOOKBLOCK e reutilizado o objecto BOOK que já existia no desenho anterior. Definiu-se que um BOOKBLOCK pode conter vários BOOK mas que cada BOOK só pode surgir num único BOOKBLOCK.

Se um utilizador pretender colocar o mesmo livro em dois blocos, necessita de repetir o processo de adição para um novo livro. Isto vai posteriormente simplificar o processo de remoção, já que assim, apagar um bloco completo não acarreta o risco de apagar livros de estejam em simultâneo noutros blocos, reduzindo-se, ainda que ligeiramente, a possibilidade de erros por parte do utilizador.

O BOOKBLOCK é criado na base de dados usando como chave primária, a chave do BLOCK correspondente. Os BOOK, pela relação de multiplicidade que

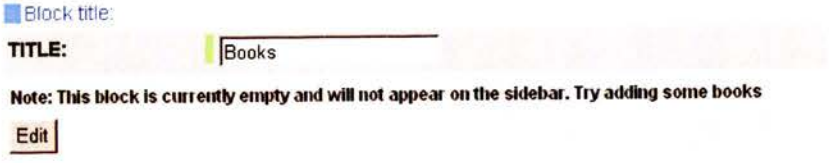
Figura 5.8: Múltiplos blocos de livro na mesma barra lateral.

Figura 5.9: Opção para apresentação do bloco na barra lateral.



mantêm com o BOOKBLOCK, recebem uma chave primária sequencial, atribuída pela base de dados, e guardam a chave do BLOCK a que pertencem como chave

Figura 5.10: Opção de apresentação do bloco desactivada.



estrangeira numa outra coluna. O título do bloco é guardado como atributo deste novo objecto, e armazenado na base de dados na tabela BOOKBLOCKS e coluna TITLE.

O texto presente na opção de visibilidade STATE da barra lateral foi alterado. Para sua substituição criaram-se mensagens distintas para apresentar conforme o bloco esteja vazio, escondido ou visível na barra lateral.

Criou-se para testar a condição anterior, um método na classe BOOKBLOCK que retorna um booleano em função da existência de livros associados a esse bloco.

Por último, para a mensagem de aviso de que o limite de resultados de pro-

Figura 5.11: Multiplicidade das associações da classe BookBlock.

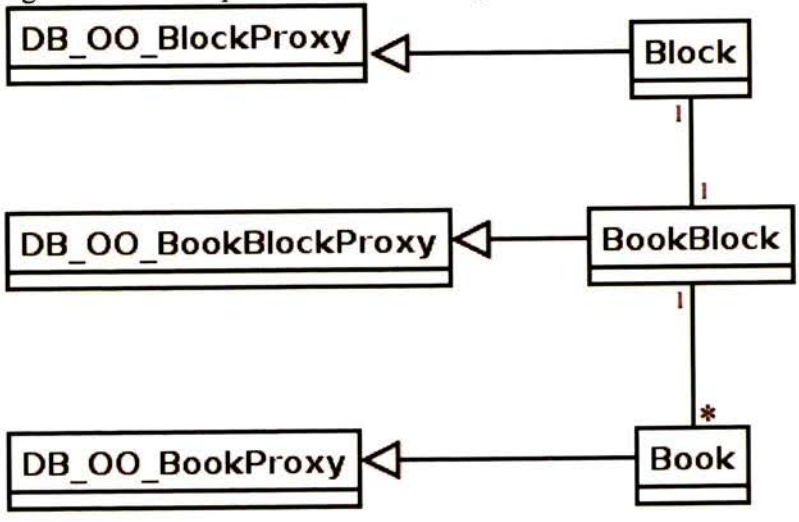
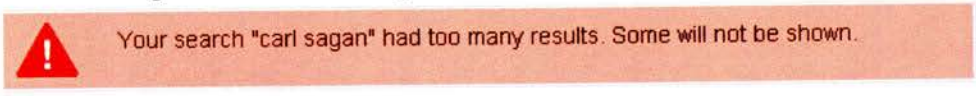


Figura 5.12: Indicação do estado de visibilidade dos blocos.

Album List	Album List	Visible	Up | Down	Edit | Delete
Sobre Refactoring	Book	Hidden	Up | Down	Edit | Delete
Livros	Book	Empty	Up | Top	Edit | Delete

cura havia sido ultrapassado, utilizou-se o sistema de notificação já existente no BLOG.COM, ficando assim homogéneo com as restantes notificações.

Figura 5.13: Alerta de procura de livro demasiado genérica.



5.3.3 Blogrolls. Refactoring do design obsoleto

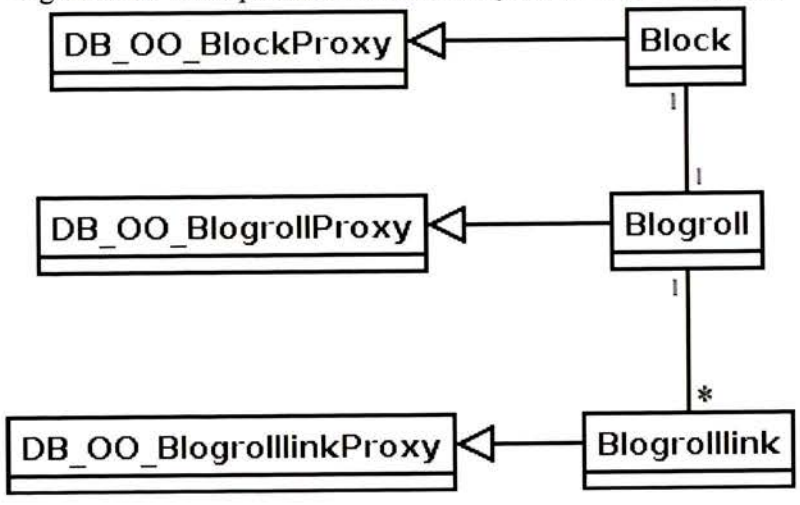
Os *blogrolls* são colecções de hiperligações para outros *blogs*. A utilização de *blogrolls* no BLOG.COM havia já sido simplificada, sobretudo pela redução de funcionalidades que se encontravam presentes em versões anteriores. A utilização dos *blogrolls* actuais consiste resumidamente na adição do título e endereço WEB do *blog* pretendido. Propositadamente, não é feita qualquer validação do endereço colocado, de modo que um *blogroll* é na realidade uma lista de *bookmarks*, seja para outros *blogs* ou não.

A alteração anterior, que levou à redução de funcionalidades, foi efectuada de modo a modificar apenas o código necessário para obter o resultado externo pretendido. Isso manteve a complexidade do desenho anterior, que utilizava mais classes do que aquelas que o desenho actual necessitava, levando a que várias delas já não tivessem qualquer uso. A nomenclatura utilizada na denominação destas classes, recorrendo frequentemente a abreviaturas, piorava a situação, pois aumentava o grau de dificuldade para quem necessitasse compreender a sua função. Optou-se assim por efectuar o *refactoring* destas classes para melhorar o seu desenho, de modo a reflectir melhor a funcionalidade actual e a facilitar a compreensão do código.

O novo desenho utiliza um objecto que representa um bloco com hiperligações. Utilizou-se a classe BLOGROLL para manter a informação do objecto especializado. Não podendo estender a classe BLOCK pela utilização da DB_OO, é utilizada a delegação de responsabilidade entre as classes para implementar o polimorfismo.

O desenho resultante reduz significativamente o número de classes utilizadas, pelo que as restantes foram removidas. Devido à utilização da DB_OO, esta acção tem um impacto directo na estrutura da base de dados, pelo que foi necessário modificar as suas tabelas para reflectir as mudanças efectuadas, resultando igualmente num menor número de tabelas.

Figura 5.14: Multiplicidade das associações da classe Blogroll.



5.3.4 Perfis. De subsecção a secção independente

Como verificamos na descrição do bloco PROFILE, em 5.1.11, uma característica dos perfis é poderem ser associados a vários *blogs*. Implicitamente, isto indica-nos que os perfis não estão contidos na barra lateral, sendo antes um elemento que coexiste em paralelo e situado ao mesmo nível.

A apresentação do perfil é, neste momento, feita exclusivamente dentro de um bloco da barra lateral, encontrando-se planeado, para uma próxima versão do BLOG.COM, que estes venham a ganhar mais destaque, sobretudo no editor de *blogs*. Optou-se assim colocar a edição deste elemento fora da secção da barra lateral, criando-lhe uma secção própria, de modo a reflectir uma posição como elemento de topo do BLOG.COM.

Outra alteração proposta foi a possibilidade de apresentar vários blocos de perfil na mesma barra lateral. Esta necessidade havia já sido demonstrada por utilizadores de *blogs* mantidos em co-autoria, e em *blogs* de natureza política, que necessitavam apresentar vários candidatos em simultâneo.

Para implementar a nova secção PROFILES ao nível da interface com o utilizador, utilizaram-se os mesmos elementos, já existentes na criação das restantes secções. Conseguiu assim manter a homogeneidade e equilíbrio da interface. O conjunto de funcionalidades disponibilizadas na subsecção da barra lateral foi depois aqui duplicado, mantendo em grande parte a forma de utilização anterior. Pretendeu-se assim manter uma linha condutora para os utilizadores já se encontrassem familiarizados ao formato. Também a pensar na coerência, ao nível das subsecções utilizou-se o formato das restantes secções. Uma subsecção que permite listar e visualizar os perfis criados, denominada LIST, e outra que

Figura 5.15: Nova secção Profiles.

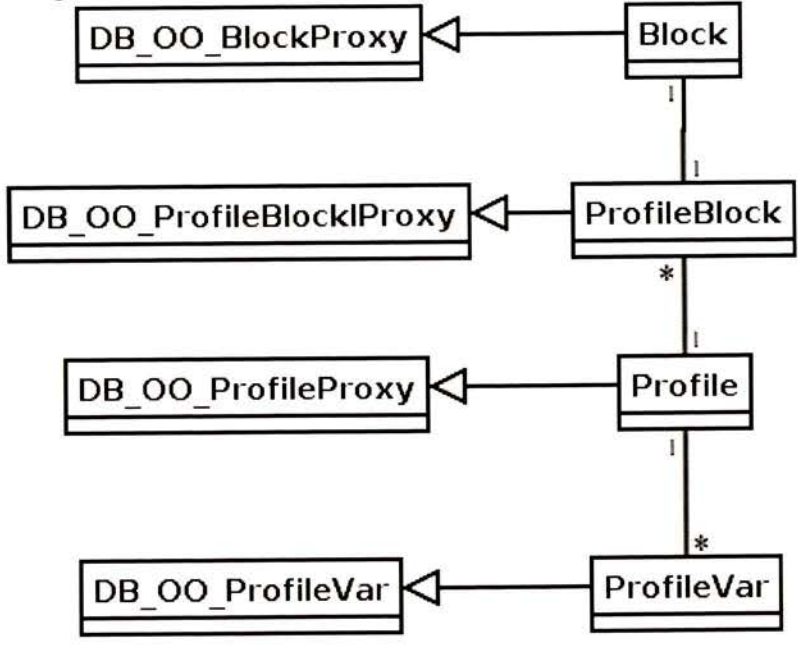
permite criar novos perfis, denominada NEW. A única correcção efectuada dentro da secção, foi relativamente à edição de alguns elementos do perfil, pois era encontrava-se a ser efectuada na página de listagem de perfis. Optou-se por transferir a edição destes elementos para o interior da opção EDIT, onde já se encontravam outros elementos.

Com a possibilidade de a barra lateral apresentar vários perfis, passou a existir uma relação com multiplicidade de muitos para muitos. Na implementação, manteve-se a classe PROFILE, para conter os dados e métodos associados ao perfil, assim como a classe PROFILEVAR, para os campos adicionais do perfil, e criou-se uma nova classe PROFILEBLOCK. Esta vai manter a informação própria do bloco, como o título, e conter a indicação de qual o perfil que lhe está associado.

Devido a relação de multiplicidade, o acesso aos perfis passa a poder ser efectuado a partir de dois locais distintos da interface. Ou acedendo pela secção perfis, onde é possível criar o perfil e associa-lo depois a um *blog*, altura na qual é criado um novo bloco, ou pela secção da barra lateral. Aqui podemos seleccionar a criação de um novo bloco, onde vai surgir a opção PROFILE BLOCK, que se seleccionada permite ou utilizar um dos perfis já criados ou criar um novo. Esta última opção redirecciona o utilizador para a subsecção NEW de PROFILES.

Como resultado, conseguiu-se obter um sistema que mantêm em simultâneo a coerência com a utilização dos perfis existentes anteriores, com a nova disposição da barra lateral e com os restantes blocos, como os livros e *blogrolls*.

Figura 5.16: Multiplicidade nas associações da classe Profile.



Capítulo 6

Conclusões

O projecto de reengenharia efectuado na BLOG.COM, caracterizou-se por ter actuado na plataforma desta aplicação de uma forma transversal aos seus componentes, permitindo tomar contacto com os vários elementos constituintes e com um amplo conjunto de tecnologias complementares.

A variedade de tecnologias utilizada e a análise da sua interacção, tornou-se assim um dos principais aliciantes deste projecto, potenciado pelo facto de que, de um ponto de vista pessoal, ser a primeira experiência em alguns dos domínios abrangidos, como a internacionalização e sobretudo o REFACTORING.

Os objectivos iniciais foram integralmente cumpridos e dentro dos prazos estipulados, tendo sido mesmo possível ir um pouco mais além do pretendido, que se reflecte no trabalho efectuado ao nível da barra lateral.

Estes resultados foram possíveis dado o excelente ambiente de trabalho vivido na PORTUGALMAIL - COMUNICAÇÕES, S.A., que apesar de manter um número reduzido de colaboradores, tem neles concentrado um elevado nível de experiência, conhecimento e capacidade de produção. Apesar de este ser um projecto individual, ficou nele patente o contributo do trabalho de equipa produzido no interior da empresa, com inúmeras e valiosas contribuições dadas pelos restantes colaboradores.

Dada a diversidade das intervenções, a melhor forma de verificar os resultados obtidos será efectuando uma análise individual das várias partes.

Assim, ao nível da internacionalização, verifica-se que a escolha tecnológica foi acertada, resultado de uma aposta em tecnologias com provas dadas na área, como o GETTEXT. Esta escolha consiste uma mais-valia para o BLOG.COM, respondendo positivamente às necessidades actuais e mantendo potencial para as necessidades que possam surgir.

A escolha tecnológica e implementação são ainda complementadas pelo cuidado tido quanto à facilidade de utilização do produto final. A tradução consiste em vários passos que requerem conhecimento ao nível da utilização da plataforma

de tradução, o GETTEXT, do sistema de templates SMARTY e da estrutura da aplicação no sistema de ficheiros, de modo a conseguir-se localizar os ficheiros chave, pelo que definiu-se um processo que descreve sucintamente as várias etapas necessárias sendo secundado por *scripts* que permitem automatizar e simplificar todo o processo.

Foi ainda considerado o impacto tido no restante desenvolvimento da plataforma. Verifica-se que as decisões tomadas quanto à marcação do texto a traduzir, para além de resultarem numa utilização simples e versátil, conseguem ainda minimizar o seu impacto quanto à legibilidade do código, não se tornando um obstaculo ao desenvolvimento por colaboradores que não familiarizados com o processo de tradução.

Apesar de a solução implementada ser funcional, durante a sua colocação em produção foram detectadas anomalias que afectam a actualização do texto a traduzir, aparentemente relacionadas com o servidor WEB de produção. Concretamente, verifica-se que a actualização dos ficheiros MO levam a que o servidor APACHE deixe de responder, requerendo a sua reinicialização. O tempo necessário à operação é reduzido, bastando alguns segundos, mas será perceptível aos utilizadores que nesse período solicitem uma determinada página. Até localizar-se e ser possível solucionar o problema, é possível atenuar esta situação agrupando as alterações ao texto em versões de lançamento, de modo a serem colocadas de uma única vez em produção.

A reestruturação da árvore da aplicação representa também um contributo positivo para o futuro desenvolvimento do BLOG.COM. A nova estrutura torna o propósito do conteúdo de cada ficheiro nítido, por facultar significado à sua localização na árvore da aplicação, permitindo ainda identificar quais os componentes afectados. Em simultâneo, esta operação permitiu eliminar código obsoleto e identificar possíveis acções de *refactoring*.

A migração da antiga estrutura para a mais recente não foi efectuada na totalidade. Tal modificação, a ser efectuada por alguém não familiarizado, obrigaria a uma análise completa de todos os ficheiros de modo a poder identificar a sua função e qual o componente que lhe estaria associado, editor, portal ou ambos. Optou-se por uma abordagem de melhoria progressiva na qual os ficheiros seriam transferidos à medida que eram necessários nas restantes acções, altura na qual o seu propósito já seria claro. Como resultado, ficaram correctamente colocados os ficheiros utilizados nas várias intervenções descritas neste projecto, assim como aqueles em que era possível compreender o seu propósito por manterem funções idênticas. O trabalho por realizar nos restantes ficheiros, é sobretudo na identificação do componente que os utiliza, estando já maioritariamente correctamente colocados quanto a sua função.

O trabalho de *refactoring* constituiu o maior desafio, por obrigar a uma análise crítica do trabalho desenvolvido por colaboradores já muito familiarizados com o

desenho de aplicações. Por este motivo, poucas alterações foram efectuadas ao desenho geral da aplicação. Aquelas que foram feitas rompem significativamente com o desenho anterior, pelo que serão necessárias futuras acções de *refactoring* para tornar o conjunto das alterações e código existente mais elegante.

Melhores resultados foram obtidos utilizando o REFACTORING como ferramenta para a correcção de situações que estavam na origem de erros já visíveis, como *breadcrumbs* incorrectas, do que para melhorar o desenho global da aplicação. Concluí-se que o motivo encontra-se na relação entre o REFACTORING e os DESIGN PATTERNS. Nas intervenções onde foi possível utilizar a técnica do REFACTORING isoladamente, como forma de melhorar o código, obteve-se em geral bons resultados, ao contrário do que naquelas que obrigavam à identificação de possíveis acções de melhoria. Estas últimas, entram no domínio dos DESIGN PATTERNS, sendo então neste domínio que as futuras acções ao BLOG.COM deverão incidir.

O redesenho da barra lateral permitiu tornar a edição deste componente dos *blogs* mais semelhante à edição dos restantes componentes, reestruturando em simultâneo as secções de navegação. O resultado final torna a utilização global da aplicação mais simples e versátil, pois os conhecimentos que um utilizador adquire numa secção podem ser facilmente utilizados para compreender as restantes. As possibilidades de edição da barra lateral foram também aumentadas. A estrutura do código desenvolvida para implementar as novas funcionalidades permite ainda futuros desenvolvimentos na barra lateral, tornando-se agora mais simples articular a interface e criar novos tipos de blocos.

Em termos geral, pode-se concluir que apesar de o projecto ter cumprido os seus objectivos, não esgotou as possibilidades de reengenharia do BLOG.COM. É necessário continuar o desenvolvimento aqui iniciado, sobretudo na área do desenho da aplicação, e em simultâneo abranger novas áreas, mantendo-se assim o dinamismo do projecto.

Bibliografia

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[2] Martin Fowler. *Refactoring: Improving the Design of Existing code*. Addison-Wesley, 1999.

[3] Grady Booch, James Rumbaugh, Ivar Jacobson. *Unified Modeling Language User Guide, The*. Addison-Wesley, 1998

[4] James Rumbaugh, Ivar Jacobson, Grady Booch. *Unified Modeling Language Reference Manual, The*. Addison-Wesley, 1998

[5] Tex Texin, Yves Savourel. *Web Internationalization: Standards and Practice*. 2004.

[6] Sagi Bashari. *SmartyGettext*. Smarty Wiki.

[7] André Rabold. *SmartyMultilanguageSupport*. Smarty Wiki.

[8] Tom Anderson. *SmartestSmartyPractices*. Smarty Wiki.

[9] Michael Wimmer. *PHP-FLP: Flaimos Little Package*. Sourceforge.

[10] *Application Structure*. phpguru.org.

BIBLIOTECA F.E. PORTO

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000081403