Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

RELATÓRIO NOS TERMOS DO DESPACHO
N.º 20/2010 PARA OBTENÇÃO DO GRAU
DE MESTRE EM ENGENHARIA
INFORMÁTICA, POR LICENCIADOS
"PRÉ-BOLONHA"

Seleção e Customização de um Sistema de Gestão de Conteúdos Multilingue

O CASO PRÁTICO
DAISY CMS / A ROCHA INTERNATIONAL

JÚLIO MIGUEL GASPAR REIS
(ALUNO N.º 36 050)

Orientador científico:
Prof. Doutor Henrique João Domingos

Júri:
Prof. Doutor José Augusto Legatheaux Martins
Prof. Doutor Henrique João Domingos
Prof. Doutor Nuno Miguel Cavalheiro Marques

Monte da Caparica, setembro de 2011

Esta página foi intencionalmente deixada em branco.

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

RELATÓRIO NOS TERMOS DO DESPACHO
N.º 20/2010 PARA OBTENÇÃO DO GRAU
DE MESTRE EM ENGENHARIA
INFORMÁTICA, POR LICENCIADOS
"PRÉ-BOLONHA"

Seleção e Customização de um Sistema de Gestão de Conteúdos Multilingue

O CASO PRÁTICO
DAISY CMS / A ROCHA INTERNATIONAL

JÚLIO MIGUEL GASPAR REIS
(ALUNO N.º 36 050)

Orientador científico:
Prof. Doutor Henrique João Domingos

Júri:
Prof. Doutor José Augusto Legatheaux Martins
Prof. Doutor Henrique João Domingos
Prof. Doutor Nuno Miguel Cavalheiro Marques

Monte da Caparica, setembro de 2011

"Seleção e Customização de um Sistema de Gestão de Conteúdos Multilingue: O Caso Prático Daisy CMS / A Rocha International"

DEDICATÓRIA

Gostaria de reconhecer quem me tem apoiado nesta tese, como em tudo. Encontrei palavras melhores do que as minhas para vos dedicar este trabalho.

To my line manager and friend, Barbara Mearns: you make my job the delight it is. May there always be a dragonfly on your path; and may you always have the zest for wandering after it.

"Admiring Nature in her wildest grace,
These northern scenes with weary feet I trace;
O'er many a winding dale and painful steep,
Th' abodes of covey'd grouse and timid sheep,
My savage journey, curious, I pursue
Till fam'd Breadalbane opens to my view."

—Robert Burns [BURN87]

À minha mulher, Ana: contigo atravessei licenciatura, mestrado, perigos enormes e ainda maiores alegrias. Que sempre dês fruto na estação própria.

"Quero fazer contigo
o que a primavera faz com as cerejeiras."

—Pablo Neruda [NERU70]

Aos meus filhos, Eva, Pedro e Alice: vocês são as sequóias que nós plantámos. Que outros um dia vos vejam como nós já vos imaginamos: grandes, fortes, admiráveis.

"Façam as perguntas que não têm respostas.
Invistam no milénio. Plantem sequóias.
Digam que a vossa principal safra é a floresta
que não plantaram,
que não viverão para colher.

—Wendell Berry [BERR73]

Esta página foi intencionalmente deixada em branco.

RESUMO

A tarefa de gerir um *website* com mais do que uma dúzia de páginas não pode ser executada eficientemente com recurso a edição manual. O aumento de importância e de dimensão dos *websites* das entidades coletivas, bem como a sua generalização junto do público individual, gerou a atual proliferação de mais de mil sistemas de gestão de conteúdos (CMS) para a web. Cada sistema tem as suas caraterísticas específicas em termos de licenciamento do código-fonte, complexidade, especialização, frequência de atualização, e penetração no mercado. É possível ainda contratar o desenvolvimento de um CMS à medida. Entre tantas opções, é difícil fazer uma escolha adequada e em tempo útil.

A principal contribuição desta dissertação para o campo da informática é fazer luz sobre o processo de seleção de um CMS, apresentando um caso real. Em 2007, o autor era *webmaster* da organização A Rocha International, cujo *website* contava mais de mil páginas em sete línguas diferentes, editadas a partir de uma dúzia de países, e servidas por um CMS que se revelava inadequado para gerir tal volume de informação. Apresentamos todo o processo de mudança para um novo CMS, incluindo uma análise crítica do CMS existente; avaliação das necessidades da organização e dos critérios de seleção definidos; análise dos CMS considerados e uma exposição do CMS selecionado, Daisy; e descrição da implementação final. A dissertação termina com uma apreciação crítica do processo, tecendo considerações práticas sobre uma próxima escolha de CMS para A Rocha International. Esta apreciação crítica evidencia o caso de estudo apresentado como representativo da experiência profissional desenvolvida pelo autor ao longo da sua atividade profissional durante os últimos 11 anos, na área de projeto e conceção de sistemas e serviços *web*, bem como de análise, desenvolvimento, integração e operação de sistemas de gestão de conteúdos.

Termos-chave: World Wide Web; sistemas de gestão de conteúdos; tomada de decisão; Daisy CMS; código aberto; XSLT.

Esta página foi intencionalmente deixada em branco.

ABSTRACT

The task of managing a website with more than a dozen pages cannot be carried out efficiently by resorting to manual edits. The increase in importance and dimension of corporate entities' websites, as well as the generalization of personal websites, has brought about the current proliferation of over a thousand web content management systems (CMS). Each system has its specific characteristics in terms of source code licensing, complexity, specialization, update frequency, and market share. It is also possible to contract the development of a CMS tailor-made to one's specifications. Among so many options, it can be hard to make a decision that is both adequate and time-effective.

This dissertation's main contribution to the field of informatics is to shed light on the process of selecting a CMS by presenting a real-life case. In 2007, the author was the webmaster of A Rocha International, whose website counted over 1,000 pages in seven different languages, edited from a dozen countries, and served by a CMS which was inadequate to manage such volume of information. We will be presenting the whole process of changing to a new CMS, including a critical analysis of the existing CMS; an evaluation of the organization's needs and of the selection criteria defined; an analysis of the CMSs considered, and a detailed explanation of the selected CMS, Daisy; and a description of the final implementation. This dissertation concludes with a critical analysis of the whole process, as well as considerations about a future decision process concerning the choice of a CMS. This critical analysis illustrates the present case study as representative of the expertise developed by the author, during the last 11 years, in his professional activity in the field of conception and implementation of web systems and services, as well as analysis, development, integration and operation of content management systems.

Keywords: World Wide Web; content management system; decision making; Daisy CMS; open source; XSLT.

Esta página foi intencionalmente deixada em branco.

ÍNDICE DE MATÉRIAS

Dedicatória v
Resumo vii
Abstract ix
Índice de matérias xi
Índice de figuras xvii
Índice de tabelas xix
Lista de abreviaturas, siglas e símbolos xxi
Enquadramento de percurso profissional xxv
Notas prévias xxvii
Percurso profissional xxix
1. Introdução 1
1.1 Âmbito do percurso profissional e motivação da dissertação 1
1.2 Problema 1
1.3 Objetivos e enquadramento da dissertação 1
1.3.1 Objetivos 1
1.3.2 Enquadramento da dissertação 2
1.4 Organização do documento 2
2. Contexto do problema 3
2.1 Situação inicial: site manual 3
2.2 Situação seguinte: CMS Carrelet 4
2.2.1 Problemas verificados com o Carrelet 5
2.2.2 Final de ciclo de vida 6
2.3 Fórum sobre o novo site 6
2.4 Condicionalismos do processo 7
2.5 Princípios gerais 8

2.5.1 PHP considerado perigoso ... 8
2.5.2 Código proprietário considerado perigoso ... 9
2.5.3 A nova solução deve ser definitiva ... 9
2.6 Critérios de escolha do novo CMS ... 9
2.7 Conclusão ... 11
3. Trabalho relacionado ... 13
3.1 Atualização de web sites ... 13
3.1.1 Atualização manual ... 14
3.1.2 Caraterísticas de um CMS ... 15
3.1.3 Funcionalidades de um CMS ... 16
3.2 CMS analisados ... 17
3.2.1 Daisy ... 17
3.2.2 Drupal ... 18
3.2.3 Gadara ... 19
3.2.4 Joomla! ... 20
3.2.5 MediaWiki ... 21
3.2.6 Plone 3 ... 22
3.2.7 SPIP ... 23
3.3 Interatividade no website ... 24
3.4 Os dois finalistas: Daisy e Gadara ... 24
3.4.1 Vantagens adicionais do Gadara ... 26
3.4.2 Vantagens adicionais do Daisy ... 27
3.5 Custos de mudança de CMS ... 27
3.6 Conclusão ... 28
4. Descrição do sistema ... 29
4.1 Instalação ... 29
4.2 Servidor-repositório ... 30
4.2.1 Motor de indexação Jakarta Lucene ... 30
4.2.2 Access Control List ... 30
4.2.3 Publisher ... 31
4.2.4 Outras funcionalidades do repositório ... 32

4.3 Daisy Wiki ... 32
4.3.1 Edição de documentos ... 32
4.3.2 Preparação de documentos ... 33
4.3.3 Skins ... 34
4.3.4 Navegação hierárquica e definição de URL ... 34
4.3.5 Navegador de facetas ... 35
4.3.6 Outras funcionalidades da Wiki ... 35
4.4 Documentos ... 36
4.4.1 Variantes ... 37
4.4.2 Versões ... 38
4.4.3 Partes ... 38
4.4.4 Campos ... 38
4.4.5 Coleções ... 39
4.4.6 Comentários ... 39
4.5 Daisy Query Language ... 39
4.6 Utilizadores ... 41
4.7 Tecnologias utilizadas ... 41
4.8 Conclusão ... 42
5. Implementação ... 43
5.1 Resumo cronológico da implementação ... 43
5.1.1 Fase de preparação ... 43
5.1.2 Fase de instalação ... 43
5.1.3 Fase de manutenção ... 44
5.2 Alojamento ... 44
5.2.1 Alojamento próprio vs. contratado ... 44
5.2.2 Alojamento partilhado ... 45
5.2.3 Servidor dedicado ... 45
5.2.4 Servidor privado virtual ... 46
5.2.5 Co-localização ... 46
5.2.6 Cloud hosting ... 47
5.2.7 Conclusão ... 47

5.3 Sistema operativo..... 49
5.4 Arquitetura da informação..... 49
5.4.1 Servidores web Jetty e Apache..... 49
5.4.2 Estrutura dos sites Wiki..... 50
5.4.3 Workflow de publicação..... 51
5.4.4 Roles e ACL..... 51
5.5 Migração do website..... 52
5.5.1 Migração dos conteúdos..... 52
5.5.2 Migração dos utilizadores..... 53
5.5.3 Migração do design..... 53
5.6 Customização da skin..... 54
5.6.1 Localização..... 55
5.6.2 Formatação de listas..... 56
5.6.3 Google Analytics..... 58
5.6.4 Adaptação da skin para árabe..... 58
5.6.5 Alterações ao código sem impacto gráfico..... 59
5.6.6 Elementos de design não utilizados..... 59
5.7 Tipos de documento..... 59
5.7.1 Image..... 59
5.7.2 LiteralHtml..... 60
5.7.3 IframeWrapper..... 61
5.7.4 News..... 61
5.7.5 MultiColumn..... 62
5.7.6 FormMailer..... 62
5.7.7 GoogleMap..... 67
5.7.8 Event..... 69
5.8 Outras skins executadas..... 69
5.8.1 riadealvor.org..... 70
5.8.2 tanariverdelta.org..... 70
5.8.3 empreintedenature.ch..... 70
5.9 Bugs descobertos..... 72

5.10 Conclusão ... 72
6. Validação ... 73
6.1 Seleção ... 73
6.1.1 Favorecimento da complexidade ... 73
6.1.2 Especificações pouco equilibradas ... 74
6.2 Implementação ... 74
6.3 Desempenho ... 74
6.3.1 Estabilidade ... 75
6.3.2 Segurança ... 75
6.3.3 Escalabilidade ... 76
6.3.4 Extensibilidade ... 77
6.3.5 Roadmap ... 77
6.3.6 Autenticação ... 77
6.3.7 Sistema de permissões incluindo permissão de leitura ... 77
6.3.8 Versionamento ... 77
6.3.9 Histórico para auditoria ... 77
6.3.10 URL estáveis ... 78
6.3.11 Localização e internacionalização ... 78
6.3.12 Gestão de documentos multilingue ... 78
6.3.13 Design customizável através de templates ... 78
6.3.14 Agendamento de conteúdos ... 79
6.3.15 Pesquisa ... 79
6.3.16 Interatividade ... 80
6.3.17 Formulários ... 80
6.3.18 Curva de aprendizagem suave ... 81
6.3.19 Migração de conteúdos ... 81
6.3.20 Suporte técnico e documentação ... 81
6.4 Outros aspetos ... 81
6.4.1 Reação ao Daisy ... 81
6.4.2 Upgrade de versão ... 82
6.4.3 Redimensionamento de imagens ... 82

6.5 Conclusão ... 83
7. Conclusões finais ... 85
7.1 Considerações para o próximo CMS ... 85
7.1.1 Opções estratégicas para a presença online ... 85
7.1.2 Vida útil da solução ... 85
7.1.3 Definição rigorosa de especificações ... 85
7.1.4 Lista de especificações revistas ... 86
7.2 Uma próxima implementação de CMS ... 86
7.3 Candidatos já eliminados ... 87
7.3.1 Business Catalyst ... 87
7.3.2 Drupal ... 87
7.3.3 Joomla! ... 87
7.4 Futuros candidatos a CMS para arocha.org ... 88
7.4.1 BlueBream ... 88
7.4.2 Daisy ... 89
7.4.3 Plone 4 ... 89
7.4.4 Ruby + Sinatra + Haml + Sass ... 90
7.4.5 WordPress ... 90
7.5 Vetores para investigação futura ... 91
8. Bibliografia ... 93

ÍNDICE DE FIGURAS

Figura 0.1: Mapa-mundo das organizações nacionais A Rocha……xxxii
Figura 2.1: arocha.org, novembro de 2002 (atualização manual)……3
Figura 2.2: arocha.org, março de 2007 (feito em Carrelet)……4
Figura 2.3: Painel de controlo Carrelet……5
Figura 3.1: Vulnerabilidades severas de alguns CMS……22
Figura 4.1: Uma instalação padrão Daisy……29
Figura 4.2: Core e extensões do servidor-repositório……30
Figura 4.3: Editor da ACL……31
Figura 4.4: Componente Publisher……32
Figura 4.5: Editor WYSIWYG HTMLArea……33
Figura 4.6: Estrutura de um documento Daisy……37
Figura 5.1: Exemplo de arquitetura cloud……47
Figura 5.2: Resumo das regras na ACL……51
Figura 5.3: arocha.org, setembro de 2011……53
Figura 5.4: Skin 'default'……54
Figura 5.5: Consulta de notícias com 'style_hint'……56
Figura 5.6: Resultado da skin numa página em árabe……58
Figura 5.7: "Love Birds" – um exemplo de LiteralHtml……60
Figura 5.8: IframeWrapper correndo o website da FCT–UNL……61
Figura 5.9: Campos de um documento FormMailer……63
Figura 5.10: Diagrama de funcionamento da API reCAPTCHA……64
Figura 5.11: FormMailer com reCAPTCHA……66
Figura 5.12: riadealvor.org com documento 'GoogleMap'……67
Figura 5.13: tanariverdelta.org……70
Figura 5.14: empreintedenature.ch: designs diferentes em cada área……71
Figura 6.1: Círculo vicioso de falta de recursos humanos……74
Figura 6.2: Problemas na ligação com o servidor-repositório……75
Figura 6.3: Interface de tradução do Plone……78
Figura 6.4: Página de resultados de pesquisa……80

Figura 7.1: Quota de mercado de CMS, setembro de 2011.....88

ÍNDICE DE TABELAS

Tabela 3.1: Funcionalidades mais habituais num CMS..... 16
Tabela 3.2: Comparação entre Daisy e Gadara..... 24
Tabela 5.1: Comparação entre tipos de alojamento..... 48

Esta página foi intencionalmente deixada em branco.

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

4GL – Fourth-generation Language
ACL – Access Control List
ADSL – Asynchronous Digital Subscriber Line
API – Application Programming Interface (interface de programação de aplicações)
ARI – A Rocha International
ARNO – A Rocha National Organization
ASP – Active Server Pages
BD – base de dados
BLOB – Binary large object
BPMN – Business Process Model and Notation
CDN – Content Distribution Network
CEO – Chief Executive Officer (diretor(a)-executivo(a))
CGI – Common Gateway Interface
CMS – Content Management System (sistema de gestão de conteúdos)
CPU – Central Processing Unit (unidade central de processamento)
CRM – Customer Relationship Management
CSS – Cascading Style Sheets
DNS – Domain Name Server
DoS – Denial of Service
DQL – Daisy Query Language
DSL – Domain-specific language (linguagem de domínio específico)
ENS – Empresa Nacional de Software
FCT–UNL – Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa
FOP – Formatting Objects Processor
FOSS – Free and open source software
GB – *gigabyte*
GNU – GNU's Not Unix!
GUI – Graphical User Interface

Haml – HTTP Abstraction Markup Language

HVAC – Heating, Ventilation and Air Conditioning (aquecimento, ventilação e ar condicionado)

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

ID – identificador

IIS – Microsoft Internet Information Services

IMAP – Internet Message Access Protocol

IP – Internet Protocol

ISO – International Standards Organization

ISP – Internet Services Provider

IVM – Integrated Virtualization Manager

JVM – Java Virtual Machine (máquina virtual Java)

LAN – Local Area Network

LDAP – Lightweight Directory Access Protocol

MB – *megabyte*

MIME – Multipurpose Internet Media Extensions

MP3 – MPEG-1 or MPEG-2 Audio Layer III

MPEG – Moving Picture Experts Group

MS – Microsoft

MVC – Model–View–Controller

NOSQL – Not Only SQL

NTLM – NT LAN Manager

ONG – Organização não-governamental

ONGA – Organização não-governamental de ambiente

OOP – Object-oriented Programming (programação orientada para objetos)

OS – Operating System (sistema operativo)

OSI – Open Systems Interconnection

PDF – Portable Document Format

p. ex. – por exemplo

PHP – PHP: Hypertext Processor

POP – Post Office Protocol

RAM – Random-Access Memory (memória de acesso aleatório)

RDF – Resource Description Framework

RDFa – Resource Description Framework in attributes

REST – Representational State Transfer (transferência de estado representacional)
RSS – RDF Site Summary
SaaS – Software as a Service (*software* como um serviço)
SAN – Storage Area Network
Sass – Syntactically Awesome Stylesheets
SGML – Standard Generalized Markup Language
SQL – Structured Query Language
`ssh` – secure shell protocol
SVG – Scalable Vector Graphics
TCP – Transmission Control Protocol
TI – Tecnologias da informação
UPS – Uninterruptible Power Supply (fonte de alimentação contínua)
URI – Uniform Resource Identifier
URL – Uniform Resource Locator
VBScript – Visual Basic Scripting Edition
VM – Virtual Machine (máquina virtual)
VPS – Virtual Private Server (servidor privado virtual)
XHTML – Extensible Hypertext Markup Language
XML – Extensible Markup Language
XSL-FO – Extensible Stylesheet Language – Formatting Objects
XSLT – Extensible Stylesheet Language – Transformations
WAI – Web Accessibility Initiative
WAN – Wide Area Network
WYSIWYG – What You See Is What You Get
WWW – World Wide Web

Esta página foi intencionalmente deixada em branco.

ENQUADRAMENTO DE PERCURSO PROFISSIONAL

O percurso profissional do autor tem-se norteado pelos seguintes critérios-base:

1. *Qualidade de vida.* Viver num local calmo, pouco poluído, que proporcione uma vida descontraída, sem pressas, com um mínimo de tempo desperdiçado em transporte para o trabalho.
2. *Qualidade de trabalho.* Metade do nosso tempo acordado é passado no trabalho; é por isso importante que o trabalho seja do nosso agrado. Mais do que um meio de se obter sustento material, o trabalho é visto pelo autor como uma "vocação"—algo que aponta para o próprio propósito na vida. Estar na profissão que se quer, e no ramo que se quer; e sentir a liberdade de mudar de emprego sempre que tal se mostre necessário, de forma a desfrutar o melhor possível do tempo de trabalho.
3. *Prioridade à família.* A família representa para o autor a unidade-base da vida, a primeira e mais coesa rede social na qual somos inseridos desde o nascimento. Sempre que necessário, as considerações familiares tomam prioridade sobre os outros fatores.

Estes critérios têm sido determinantes no percurso profissional, como se verá na secção prévia sobre a carreira profissional do autor.

Esta página foi intencionalmente deixada em branco.

NOTAS PRÉVIAS

A presente função profissional do autor é exercida num contexto internacional, com *websites* cujas páginas em português servem utilizadores tanto do Brasil, como de Portugal. Assim, o autor decidiu adotar este ano a norma do *Acordo Ortográfico da Língua Portuguesa de 1990*, embora essa norma apenas se vá tornar obrigatória em Portugal no ano de 2015 ([MNE10], [AR08]). A ortografia da presente dissertação segue essa norma.

Em termos de estrutura, este documento segue as recomendações para elaboração de dissertação emitidas pela Universidade ([DI11a], [CC11], [DI11b]). Foi composto exclusivamente com tipos de letra *open source*: Linux Libertine e Linux Libertine C para o corpo do texto, Linux Biolinum para os títulos e `DejaVu Sans Mono` para partes em código e nomes de ficheiros.

As marcas indicadas são propriedade dos seus respetivos titulares. Os símbolos ® e ™ foram omitidos por uma questão de simplicidade.

Esta página foi intencionalmente deixada em branco.

PERCURSO PROFISSIONAL

Banco Nacional Ultramarino (1995–1997)

Em 1995, por razões de ordem familiar, o autor teve que interromper o projeto final de licenciatura, que aliás já se vinha a arrastar em demasia, e encontrar um emprego na cidade de onde era natural: Alcobaça. Assim, o autor aceitou o seu primeiro emprego como Gestor de Contas Particulares em regime de estágio no Banco Nacional Ultramarino. A função envolvia captar novos clientes para o banco, e prestar-lhes um serviço personalizado e completo: crédito, aplicações financeiras, gestão bancária *etc.*.

A única ligação do emprego com a licenciatura, era que era esperado que os estagiários fossem detentores de um diploma de ensino superior, que o autor se comprometeu a concluir. A informática esteve presente apenas na ótica do utilizador. No entanto, diversos conceitos apreendidos na parte letiva da licenciatura mostraram-se úteis na compreensão dos sistemas informáticos do banco: autenticação, transação atómica de base de dados, normalização e desnormalização de dados.

Contrariamente ao esperado, e por razões alheias ao autor, o contrato de estágio no Banco não foi renovado. A perda de um emprego é uma situação angustiante, ainda mais numa situação de início de vida de casado, na qual se procura alguma estabilidade. No entanto, a saída do Banco acabou por se revelar providencial. Após algumas semanas à procura de emprego, tornou-se evidente que o melhor caminho não era um novo emprego, mas a conclusão da licenciatura. Após obtido o diploma, seria porventura mais fácil obter um emprego gratificante; e além disso, é bom princípio concluir o que se começa. Assim, o autor voltou à faculdade para a escolha de um projeto final.

Empresa Nacional de Software, Ld.ª (1997)

Na primeira tentativa de conclusão da licenciatura, o autor tinha escolhido desenvolver um projeto interno na própria FCT–UNL. Da segunda vez, optou por um projeto numa empresa. A ideia subjacente era que um projeto em empresa seria mais concreto, e teria mais possibilidades de ser concluído dentro de um prazo relativamente curto, e o regresso do autor ao meio profissional seria mais rápido. Essa estratégia revelou-se acertada.

O projeto final de curso foi desenvolvido na Empresa Nacional de Software (ENS), na Maia, em junho e julho de 1997. O projeto foi desenvolvido sobre uma ferramenta de desenvolvimento para a web intitulada *WebSpeed*, que gera *SpeedScript*, um subconjunto da linguagem de programação de 4.ª geração Progress 4GL (hoje OpenEdge Advanced Business Language [PROG]). Na sua formulação original, o projeto previa apenas o estudo das ferramentas de programação, e a sua demonstração na empresa.

No presente ano de 2011, esta formulação é demasiado simples para um projeto final de licenciatura, mas em dezembro de 1996, quando o projeto foi aceite no Departamento de Informática da FCT–UNL, era o estado da arte. Não havia muitas ferramentas de desenvolvimento para a *web*, integradas com base de dados.

Após deslocação à empresa, o autor viu que o projeto havia sido alterado. Já não era suficiente uma demonstração trivial das ferramentas de desenvolvimento, pois entre a data em que o projeto havia sido submetido e aceite pela FCT–UNL (dezembro de 1996) e a data em que o projeto foi aceite pelo autor (junho de 1997), teve lugar na empresa uma ação de formação sobre a ferramenta WebSpeed. Assim, o que a ENS agora pretendia era uma plataforma *web* que se intitulou Sistema de Apoio Técnico Aplicacional, que tinha como como objectivos:

1. Registar e controlar as chamadas telefónicas de apoio técnico;
2. Apoiar os Agentes de software da ENS em todo o país; e
3. Listar os produtos e serviços na base de dados.

Este sistema era um misto de *intranet* (pontos 1 e 2) e de *site* público (ponto 3); era constituído por 45 páginas web, e ficou completamente pronto em dois meses ([REIS97]). A nota final obtida foi dezassete valores.

Esta foi a primeira vez que o autor aprendeu e trabalhou com HTML, que não lhe havia sido ensinado na faculdade. O autor havia tido uma conta em 1993 num sistema com *email* (o seu primeiro endereço de *email* foi jgr@fct.unl.pt, ainda antes das contas dos alunos estarem no subdomínio 'students.fct.unl.pt'). Esse sistema dava também acesso a websites, alguns em protocolo Gopher, outros já em protocolo HTTP. Na licenciatura o autor havia aprendido sobre *internetworking*, mas a ênfase principal havia incidido sobre as camadas de rede, transporte e sessão do modelo ISO/OSI (por exemplo, o protocolo IP, o protocolo TCP, e *named pipes*, respetivamente). A camada de aplicação da Internet não foi explorada de todo.

O conceito subjacente à Internet já havia agradado ao autor aquando do uso do *email* e páginas WWW, e agradou também a construção de conteúdo para a *web*.

Centro Informático da Quinta / Futurekids (1997–1999)

No dia em que terminou o projeto final de curso, o autor teve uma entrevista de emprego, e foi de imediato colocado no seu emprego seguinte. A função exercida no Centro Informático da Quinta incluía:

1. Dar formação de informática a crianças e jovens segundo o método e os programas Futurekids;
2. Dar formação de informática a adultos;
3. Criar e manter o *website* da empresa;
4. Gerir o equipamento informático da empresa, p. ex. instalar software e manter a rede.

Durante o tempo em que esteve neste emprego, para além do *site* da empresa, o autor criou um site pessoal (o *Castelos de Portugal*, [REIS04]). Idealizado como um projeto coletivo, a colaboração era limitada ao fornecimento de texto e imagens, já que em 1998, quando o projeto começou, não existiam CMS que permitissem uma edição descentralizada. O *site* era completamente criado à mão, em HTML 4 e CSS, o que permitiu ao formando obter prática na

manutenção de *sites* com recurso a ferramentas como MS FrontPage e DreamWeaver. O site nunca foi descontinuado, mas por essa altura o advento de outros *sites* com conteúdo sobre o assunto, nomeadamente a Wikipédia, tornou desnecessário este projeto.

Após um ano de trabalho, e apesar do sucesso obtido na função, o autor considerou que a sua realização profissional estaria numa área mais técnica da informática. Assim, no ano seguinte o autor despediu-se e procurou outro emprego, preferencialmente centrado na Internet.

Globalsoft—Desenvolvimento de Software, Ld.ª (2000–2002)

Este foi o primeiro emprego do autor especificamente na área da Internet. A função desempenhada incluía:

1. Desenvolvimento do *website* das empresas do grupo (dois);
2. Desenvolvimento de *websites* de clientes (cerca de cinco); e
3. Criação de *sites* como *front-end* de bases de dados.

A Globalsoft é uma empresa virada para o desenvolvimento de *software* para o mercado autárquico. Desenvolve aplicações p. ex. para gestão de cemitérios, emissão de licenças de cães e gatos, faturação de águas e saneamento, eleições, e inventário. Como parceira Microsoft, a Globalsoft desenvolvia em Visual FoxPro sobre bases de dados FoxBase, uma solução de base de dados um pouco antiquada, mesmo para a altura, já que o desenvolvimento sobre ferramentas Microsoft nessa altura já era feito maioritariamente em Visual Basic.

Os *sites* que o autor desenvolveu eram feitos com páginas ASP servidas via IIS no servidor Windows NT da empresa. As páginas eram programadas em VBScript, com ligação a bases de dados MS Access ou MS SQL. O autor frequentou nessa altura uma formação certificada sobre Microsoft SQL Server 7.0.

Este foi o último emprego em que o autor trabalhou num contexto presencial; desde essa altura tem estado a exercer a sua atividade profissional em regime de teletrabalho.

Durante o tempo em que esteve na Globalsoft, o autor tomou um interesse pela organização ambiental A Rocha Portugal, que já conhecia havia alguns anos. Ofereceu-se como voluntário no que fosse necessário, e começou a ajudar o *webmaster* (também voluntário) da organização A Rocha International.

A Rocha International (2002–presente)

A Rocha é um conjunto de organizações de ambiente constituídas segundo a legislação nacional dos países onde cada uma delas opera (as "A Rocha National Organizations", ARNO), e ainda A Rocha International (ARI), uma organização internacional constituída segundo a lei inglesa, cujas funções são:

1. A divulgação do trabalho das organizações A Rocha a nível internacional;
2. O apoio à liderança das ARNO;
3. A criação de novas organizações A Rocha em novos países.

As organizações A Rocha, em qualquer lugar do mundo onde estejam, são caraterizadas pelos Cinco Compromissos: cristianismo, conservação, comunidade, transculturalidade e cooperação.

Figura 0.1: Mapa-mundo das organizações nacionais A Rocha

Em meados de 2002, a ARI quis melhorar significativamente a forma como comunicava via Internet. O novo modelo de *website* desejado exigiria de um *webmaster* o cumprimento de prazos e um profissionalismo que não poderia ser obtido num regime de voluntariado, e foi decidido que o *webmaster* teria que ser pago para trabalhar algumas horas por semana. Foi proposto ao *webmaster* voluntário principal que ficasse com essa função a título remunerado, mas ele recusou devido a outros compromissos. Foi depois proposto ao autor que exercesse essa função, e assim, em novembro de 2002, o autor começou a trabalhar para a ARI a tempo parcial, 8 horas por semana. Apesar do horário ser tão reduzido, o autor despediu-se do seu emprego na Globalsoft.

Com o tempo, o trabalho com a ARI passou de 8 horas por semana por conta própria, para trabalho por conta de outrem 20 horas por semana, e o autor acabou por abandonar a atividade profissional independente.

A função do autor na ARI denomina-se *webmaster internacional* ("international webmaster"). As suas responsabilidades principais, conforme indicadas na descrição de funções, são as seguintes:

1. Aconselhar a diretoria da ARI sobre questões relacionadas com comunicações por via informática;
2. Atualizar todos os sites da ARI, e monitorizar o seu desempenho segundo objetivos predefinidos;
3. Assegurar que a ARI faz bom uso das outras formas de comunicação via Internet (p. ex. redes sociais);
4. Criar e manter a funcionar as caixas de *email*;
5. Fornecer suporte e treinamento aos outros *webmasters* das ARNO em todo o mundo, que usem os meios de comunicação via Internet fornecidos pela ARI;
6. Verificar a observância das ARNO com as linhas orientadoras da ARI para a Internet.

Entre as funções técnicas desempenhadas pelo autor e não detalhadas acima, contam-se a gestão do servidor DNS, gestão do servidor Debian GNU/Linux e software instalado (incluindo o servidor HTTP Apache).

A equipa da ARI conta com 15 pessoas a trabalhar a partir de 4 países diferentes. O trabalho é feito quase todo em regime de teletrabalho. A diretoria ("management team") reúne em pessoa duas vezes por mês, já que todos os diretores vivem em Inglaterra. O autor trabalha no Departamento de Desenvolvimento e Comunicações, que engloba as áreas de comunicação impressa e via Internet, angariação de fundos e voluntariado. O Departamento reúne também duas vezes por mês, mas via Skype.

Este trabalho tem sido extremamente recompensador para o autor. É estável; existe um bom relacionamento entre todos os colegas; existe uma atmosfera de diálogo entre todos, sem distinções de categoria profissional; o salário é de nível internacional, considerando o tempo de trabalho despendido; não exige deslocações, exceto uma ou duas reuniões presenciais por ano; e visto ocupar apenas 20 horas por semana, deixa bastante tempo livre para a família, para o voluntariado e para o estudo.

teofilos.org (2005–2008)

Em 2005, o autor foi convidado, por um conjunto de escolas teológicas protestantes portuguesas, a dirigir uma plataforma de *e-learning*. Assim nasceu o `teofilos.org`. O objetivo da plataforma é complementar a oferta presencial das diferentes escolas, apresentando *online* um currículo resumido com algumas das disciplinas facultadas.

Sendo a única pessoa paga nesta iniciativa, a função do autor era extremamente diversificada, englobando por exemplo trabalho administrativo, relações públicas com escolas teológicas, e tradução. Dentro da informática, a principal tarefa do autor foi a instalação e configuração do CMS Moodle, e a colocação do conteúdo de disciplinas nesse sistema, bem como o apoio técnico aos professores e alunos utilizadores da plataforma.

Em finais de 2007, após a estabilização da plataforma de *e-learning*, o autor relatou à direção que as questões informáticas estavam ultrapassadas, e que lhe parecia melhor que nesta fase o diretor executivo fosse uma pessoa com um perfil mais orientado para a realidade do ensino teológico. Assim, a direção do teofilos.org aceitou a cessação de contrato proposta pelo autor.

O autor teve mais alguns clientes entretanto, mas desde final de 2010 que não tem assumido mais trabalho remunerado. Isso tem-lhe permitido, por exemplo, concentrar-se no programa "Ser Mestre".

Esta página foi intencionalmente deixada em branco.

1. INTRODUÇÃO

1.1 ÂMBITO DO PERCURSO PROFISSIONAL E MOTIVAÇÃO DA DISSERTAÇÃO

O percurso profissional do autor tem convergido para o desenvolvimento técnico de *websites*. Tem adquirido diversas ferramentas úteis à função de *webmaster*, tais como a marcação de páginas em XHTML, a criação de folhas de estilos CSS e de *templates* XSLT, a criação e gestão de bases de dados relacionais MS Access e MS SQL, a gestão de servidores GNU/Linux e a configuração de servidores HTTP Apache, o registo de domínios Internet e a gestão de serviços DNS, *scripting* em JavaScript, ASP/VBScript e `bash`, programação de CGI em perl, o uso de expressões regulares, a utilização de API como Google Maps e MooTools, noções de usabilidade, o uso de ferramentas estatísticas para a *web*, a escrita de documentação de sistemas informáticos para público não especialista, e prática na resolução à distância de uma ampla gama de questões relacionadas com a Internet.

A evolução futura da carreira do autor aponta para uma função de liderança na área da Internet ou das comunicações, com maior grau de responsabilidade e autonomia na tomada de decisões. A obtenção de grau de mestre é um passo no reconhecimento da qualidade do trabalho que o autor tem vindo a prestar.

1.2 PROBLEMA

O projeto sobre o qual o autor vai dissertar nesta tese diz respeito à escolha do sistema de gestão de conteúdos (CMS) presentemente usado pel'A Rocha International. O CMS Daisy foi escolhido em 2007, e a implementação inicial teve lugar em 2008, com algumas alterações desde essa altura.

1.3 OBJETIVOS E ENQUADRAMENTO DA DISSERTAÇÃO

1.3.1 Objetivos

Com esta dissertação, o autor procura valorizar o seu percurso profissional, bem como o percurso académico anterior, e contribuir para o campo da informática através da apresentação de um caso real de seleção de um CMS.

Será feita luz sobre a complexidade da seleção de um CMS adequado às necessidades de uma organização multinacional e multilingue; sobre as restrições de tempo e de recursos

presentes no dia a dia das organizações, e que condicionam quaisquer processos de decisão; sobre alguns dos CMS existentes no mercado; sobre a arquitetura do CMS selecionado; e sobre a implementação de soluções concretas sobre esse CMS. O CMS selecionado será depois validado em relação à proposta inicial, e finalmente serão apresentadas conclusões tanto quanto ao processo descrito nesta dissertação, quanto a um futuro processo de escolha de um CMS.

1.3.2 Enquadramento da dissertação

Em 2007, o autor era o *webmaster* da organização sem fins lucrativos A Rocha International. O *website* desta contava com mais de mil páginas em sete línguas diferentes, editadas a partir de uma dúzia de países. Estas páginas eram geridas por um CMS, denominado Carrelet, que havia sido desenvolvido em grande parte à medida das necessidades da ARI. O CMS era bastante simples de utilizar, pelo que teve um uso bastante alargado. No entanto, com o crescimento do *website* o CMS revelava-se cada vez mais inadequado para gerir esse volume de informação, e para atender às necessidades futuras da organização. A organização decidiu então mudar de CMS.

1.4 ORGANIZAÇÃO DO DOCUMENTO

O resto deste documento está organizado da seguinte forma:

- O capítulo 2 vai descrever o *contexto do problema*: as formas de atualização do *website* usadas anteriormente; as suas vantagens e inconvenientes; os condicionalismos que envolveram o processo de decisão; e os critérios-base definidos.
- O capítulo 3 refere-se a *trabalho relacionado*: as diferentes formas de gerir o conteúdo de um website; os diferentes CMS considerados no processo de escolha; e as conclusões do processo.
- O capítulo 4 contém a *descrição do sistema* Daisy: as suas principais caraterísticas e componentes.
- O capítulo 5 trata da *implementação*: a instalação do Daisy; as alterações efetuadas ao CMS; a migração do conteúdo; e a transição dos utilizadores.
- O capítulo 6 trata da *validação*: a análise das formas em que o CMS Daisy se revelou adequado (ou não) à tarefa proposta de gerir o website `arocha.org`.
- O capítulo 7 tece *conclusões* sobre a implementação deste CMS, considerações quanto a um próximo processo de escolha de um CMS, e indica alguns potenciais candidatos a CMS para a ARI.

2. CONTEXTO DO PROBLEMA

Um dos mandatos d'A Rocha International é "publicitar a nível internacional o trabalho feito pela família de organizações nacionais A Rocha". Assim, em 1998 a ARI criou o website arocha.org [ARI], que foi até 2009 o único local na internet para divulgação do trabalho que as ARNO desenvolviam em todo o mundo.

2.1 SITUAÇÃO INICIAL: SITE MANUAL

De início, o *website* era completamente criado de forma manual. O conteúdo estava principalmente em inglês, com algumas páginas em português, alemão e francês.

Figura 2.1: arocha.org, novembro de 2002 (atualização manual)

À medida que o número de páginas do *site* aumentava, revelaram-se algumas limitações desta forma de gerir um website:

- *Centralização numa só pessoa.* Quando o autor começou a trabalhar como voluntário no *website*, este passou a ter dois *webmasters*, o que obrigava a sincronizar o *site* nos dois computadores. Na prática, o *webmaster* anterior deixou as tarefas de atualização para o autor, devido também a outras ocupações.
- *Dificuldade em fazer alterações globais.* As alterações à estrutura ou ao *design* do

website tornam-se bastante complicadas se aplicadas página a página. Este método facilmente introduz erros no HTML da página.

Estas limitações do site foram primeiramente faladas numa reunião internacional de líderes em 2001, e na reunião seguinte em 2002 foi decidido que se deveria alterar a forma de construir o *site*, principalmente para permitir que mais pessoas pudessem colocar conteúdos *online*.

É de notar que a criação manual de um *website* também apresentava as suas vantagens, particularmente se nos referirmos a 2000–2002:

- *maior controlo sobre o resultado final*, já que uma só pessoa fazia todas as alterações;
- *possibilidade de fazer as alterações offline* e depois ligar à net para fazer a atualização— muito relevante numa altura em que poucos ISP forneciam ligação ADSL, e as ligações à internet eram pagas ao minuto.

2.2 SITUAÇÃO SEGUINTE: CMS CARRELET

O site da ARI foi atualizado manualmente até ao ano de 2003, altura em que o CMS Carrelet foi adotado. O sistema Carrelet foi desenvolvido pela Cyberporte, a empresa que na altura fornecia o alojamento *web* à ARI, para os seus clientes de alojamento que pretendiam uma forma simples de atualizarem o conteúdo das suas próprias páginas. "Carrelet" significa "solha" (o peixe) em francês.

Figura 2.2: arocha.org, março de 2007 (feito em Carrelet)

A maioria dos clientes pretendia apenas uma dúzia de páginas, e não detinha o seu próprio domínio de internet mas usava o domínio da Cyberporte. O sistema permitia criar, alterar e eliminar páginas, e gerir uma biblioteca de imagens a utilizar nas páginas. O cliente tinha um nome de conta, que vinha referido no URL. As páginas eram todas geradas no momento em que a conta fosse alterada, e permaneciam estáticas, sendo servidas como HTML simples.

Para A Rocha, este sistema foi alterado de forma à organização poder criar as suas próprias contas, a funcionar num domínio separado, e a biblioteca de imagens ser partilhada entre todas as contas. A ARI financiou o desenvolvimento do Carrelet durante os primeiros 12 meses. Era um sistema de funcionamento simples, muito rápido e muito fácil de aprender. Foi

desenvolvido em perl, e funcionou sem *bugs* desde o primeiro dia. Por ser de utilização tão fácil, o sistema Carrelet rapidamente começou a ser utilizado pelas ARNO para colocarem também conteúdos, cumprindo-se assim um dos objetivos iniciais da introdução do CMS. Nos primeiros meses de 2007, o site `arocha.org` era constituído por mais de 1 600 páginas, em sete línguas (1 060 páginas em inglês), servindo 16 organizações nacionais.

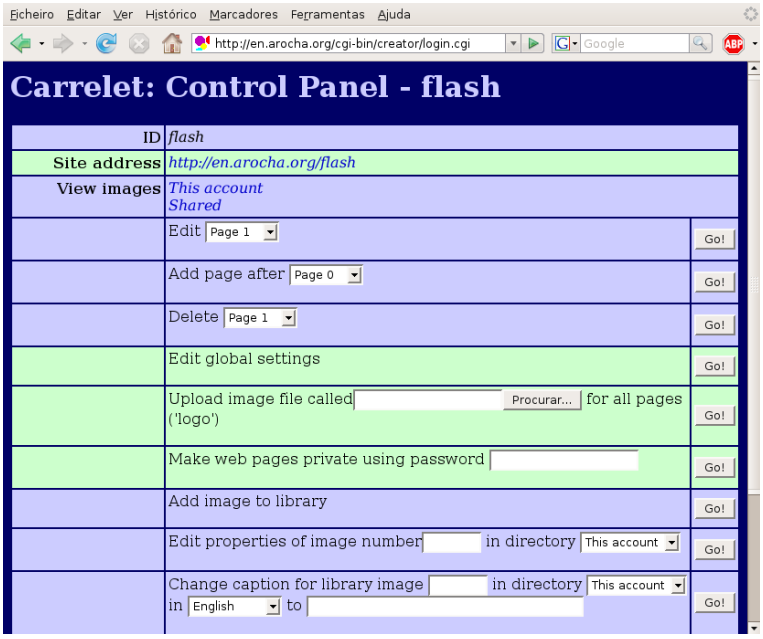

Figura 2.3: Painel de controlo Carrelet

2.2.1 Problemas verificados com o Carrelet

Também imediatamente se verificou que esta ferramenta não era a mais adequada para gerir um site com as caraterísticas do d'A Rocha. Os principais problemas diagnosticados foram os seguintes:

1. *Falta de segurança.* Cada conta tinha uma só *password*, que tinha que ser partilhada por todas as pessoas que editassem conteúdo nessa conta.
2. *Impossível reverter alterações.* O Carrelet não previa nenhum sistema de arquivo de versões anteriores de páginas, pelo que se um dos utilizadores apagasse uma página teria que a reconstruir completamente. Isto aconteceu diversas vezes.
3. *URL não customizáveis.* Os URL obedeciam à seguinte lógica: a primeira página da conta tinha o URL `index.html`; a segunda era `index2.html`, e assim sucessivamente. Não era possível atribuir nomes descritivos aos URL.
4. *URL não permanentes.* Por causa do ponto anterior, quando se inseria uma página numa conta, todas as páginas seguintes mudavam de endereço.
5. *Hierarquia demasiado rígida.* Uma página pertencia a uma só conta. No caso em que o mesmo conteúdo deveria estar presente em dois pontos do *site*, era necessário criar duas páginas, que tinham depois de ser atualizadas de forma independente.
6. *Impossível criar subcontas.* Todas as contas surgiam como diretórios criados diretamente a partir do domínio. Por exemplo, havia uma conta `arocha.org/research`

para a investigação científica, mas para alojar as páginas sobre a investigação científica levada a cabo em Portugal, foi necessário criar uma conta `arocha.org/ptresearch`.

7. *Apenas inserção de texto estruturado.* O Carrelet tinha a sua própria notação de marcação de páginas, que era convertida para HTML. Não permitia a inserção de HTML arbitrário, nem de qualquer código nas páginas.
8. *Impossibilidade de gerir outros conteúdos* como PDF e vídeo. O Carrelet geria apenas texto e imagens. Se fosse por exemplo necessário acrescentar um PDF, seria necessário que o utilizador do sistema enviasse o PDF para o *webmaster*, que o iria colocar numa secção estática do servidor, fora do Carrelet, e posteriormente comunicar o URL ao utilizador.
9. *Impossibilidade de criar formulários.* Os diferentes *emails* para contacto tinham assim que ser escritos diretamente na página, aumentando a quantidade de *spam* recebida nas caixas de correio dos colaboradores.
10. *Árabe e checo não suportados.* A língua checa era suportada indicando o *codepage* ISO 8859–2 para toda uma conta. Esta situação não suportava o uso de carateres checos em páginas de outras contas. Quando se tentou acrescentar conteúdo em árabe de forma a suportar A Rocha Líbano, verificou-se que o Carrelet teria que ser completamente reescrito para suportar escrita da direita para a esquerda.
11. *Todo o site visível.* O Carrelet não permitia esconder um conjunto de páginas. A ARI tinha documentos internos na internet, protegidos por *password*, mas estes tinham que ser atualizados usando um sistema totalmente diferente.
12. *Um template por conta.* Todas as páginas dentro de uma mesma conta obedeciam ao mesmo modelo de página. Essa limitação na apresentação dos conteúdos restringia também os próprios tipos de conteúdo apresentados.
13. *Nenhuma relação entre as diferentes traduções de uma página.* Havia nesta altura uma grande quantidade de traduções, que eram geridas como documentos separados.

2.2.2 Final de ciclo de vida

Conforme já referido, o contrato da ARI com a empresa Cyberporte previa o pagamento de uma quantia adicional para que esta continuasse o desenvolvimento do Carrelet, durante um período de 12 meses. No final deste período, em meados de 2004, a Cyberporte parou de atualizar o sistema Carrelet. Durante o período 2004–2007, a Cyberporte desenvolveu outras ferramentas de criação de sites, particularmente para sites com fóruns e com interação gráfica via avatar em espaço 3D axonométrico, mas esse modelo não se prestava ao site d'A Rocha.

2.3 FÓRUM SOBRE O NOVO SITE

Em março de 2007, foi aberta a discussão sobre um novo CMS, e um novo formato de *website*, para A Rocha. Esta discussão foi colocada no site `wikia.com`, sob o nome "Puzzled Puffin" ([REIS07a]). Uma das razões para o uso da plataforma Wikia é que esta funciona sobre MediaWiki, um dos candidatos a CMS (ver secção 3.2.5, p. 21).

Todos os colaboradores d'A Rocha (seja da ARI ou das ARNO) foram encorajados a

participar com as suas ideias ou comentários. Para além da definição dos critérios-base de escolha do novo CMS (detalhados na secção 2.6, p. 9), o debate trouxe a lume algumas questões, tais como:

- Deveríamos separar os *websites* nacionais do *website* internacional de forma bastante clara, tanto em termos de *design* como de estrutura de conteúdos?
- Como vamos migrar os conteúdos existentes no presente *site* para o novo sistema?

O processo de discussão foi importante, pois permitiu ao *webmaster* recolher informações sobre expectativas em relação ao novo *site*, e deu oportunidade aos restantes colaboradores de se sentirem incluídos no processo.

2.4 CONDICIONALISMOS DO PROCESSO

O contexto de uma decisão influi sempre no resultado da mesma. No caso da escolha do novo CMS para A Rocha, fizeram-se sentir estes condicionalismos:

1. *Limitações de recursos humanos.* A ARI dispunha em finais de 2006 de apenas uma pessoa na área da internet (o autor), a trabalhar apenas 8 horas por semana. O horário de trabalho semanal foi aumentado para 12 horas em julho de 2007, para 16 horas em outubro de 2007, e para 20 horas em janeiro de 2008, e entretanto foi contratada outra pessoa que trabalha parcialmente para a internet, mas na altura da decisão notou-se bastante a limitação dos recursos humanos.
2. *Processo de decisão à distância.* O facto do pessoal da ARI estar espalhado em diversos países tornava o processo de decisão mais demorado. Nessa altura ainda não tinham sido instituídas as reuniões quinzenais via Skype do Departamento de Desenvolvimento e Comunicações, portanto o processo de decisão funcionava por *email*.
3. *Considerações financeiras.* As limitações de custos numa ONG de ambiente são bastante evidentes. Mesmo dentro do trabalho sem fins lucrativos, é ainda mais difícil obter fundos para custos administrativos como websites, servidores e salários, já que os dadores típicos estão mais interessados no trabalho no terreno (p. ex. a educação ambiental ou a investigação científica).
4. *Conservadorismo.* A ARI era uma organização um pouco conservadora em questões administrativas—se estava a funcionar, mesmo que mal, era necessária uma boa razão para mudar. Este condicionalismo deriva não tanto da cultura da organização, mas sobretudo das limitações financeiras e de recursos humanos já referidas.
5. *Uma entidade define o website para um grupo de entidades.* A ARI era responsável por definir a estrutura para um *website* que tinha que atender às necessidades de mais de uma dúzia de organizações, espalhadas pelos cinco continentes. Isto tem o potencial de introduzir alguma indecisão no processo, já que os "clientes" do website são tanto os visitantes do mesmo, quanto as equipas e direções das ARNO, e que dado o caráter aberto do relacionamento entre elas e a ARI, era fácil esquecer que a decisão pertencia à ARI.

Estes condicionalismos atrasaram o início do processo de decisão, prolongaram a duração do processo, e atrasaram a implementação do CMS após a decisão ter sido tomada. São perfeitamente normais e para uma próxima vez serão expectáveis.

2.5 PRINCÍPIOS GERAIS

Esta secção descreve princípios que informaram todo o processo de escolha do CMS. Estes princípios dizem respeito àquilo que se esperava do novo CMS, em termos de segurança, disponibilidade e manutenção do código, e longevidade da decisão. Cada princípio é encarado aqui de forma crítica, pesando-se os seus prós e contras.

Estes princípios não foram apenas assumidos, mas foram objeto de investigação, e do conselho da parte de *webmasters* e de outras pessoas.

2.5.1 PHP considerado perigoso

A linguagem de programação PHP tem fama de ser um perigo para a segurança, tanto de servidores, quanto da informação. Eis algumas das razões apontadas:

1. É fácil misturar lógica de programação com conteúdo. A lógica do programa está contida no ficheiro HTML, o que impossibilita a separação do PHP. Isto torna os *scripts* mais longos do que necessário, já que incluem a lógica de apresentação em HTML, e dificulta o processo de *debug*. O próprio PHP Group apresenta esta forma de programar nos seus tutoriais (ver p. ex. [PHP06]).
2. A "cultura" de programação PHP encoraja práticas erradas. Como exemplo, é comum verem-se sugestões da parte de programadores PHP para se utilizar um só documento PHP (p. ex. `index.php`), usar parâmetros para indicar a ação pretendida, e fazer uso extensivo de `case` para ramificar a execução. Isto torna o código difícil de manter.
3. Como diz um comentador no site do PHP, "It only takes a single wrong character to show everyone in the world [database username and password] information" [JONE11]. Isto demonstra a periculosidade do PHP comparado por exemplo com perl, onde um carater errado significa (regra geral) uma página em branco, sendo o erro registado no *log* do servidor.

Em defesa do PHP e em abono da verdade, deve dizer-se que *todas as plataformas de desenvolvimento permitem a criação de código mal escrito*. Algumas mais do que outras, certo, mas perl, Java ou Python são também passíveis de serem transformadas em "spaghetti code".

De igual forma, *todas as plataformas permitem a escrita de código bem documentado, lógico e fácil de manter*. O PHP não é exceção. Todas as linguagens de programação têm os seus elitistas, e os seus ódios de estimação.

Hoje em dia, algumas das maiores e mais conhecidas plataformas de conteúdo correm em PHP; como exemplo, MediaWiki e WordPress. O facto de estarem em PHP aparentemente não constitui inconveniente à sua disseminação; embora para além da popularidade se deva sempre analisar a qualidade do código, e a correta integração dos diferentes *plug-ins* tais como galerias de imagens, autenticação etc..

2.5.2 Código proprietário considerado perigoso

Uma solução em código proprietário pode ser descontinuada se o dono do código assim o entender; isso foi tomado em conta no processo de escolha do CMS. O software FOSS, por outro lado, tem maior garantia de manutenção do código pois este é independente da vontade de uma corporação.

O código aberto na realidade não significa que o código seja mantido, pois em muitos casos existe uma corporação, ou um grupo significativo de programadores, por trás da iniciativa, e se essa corporação ou grupo desistirem de manter o projeto, o desenvolvimento poderá estagnar; no entanto, é sempre possível continuar a usar o código.

2.5.3 A nova solução deve ser definitiva

O novo CMS foi visto como uma solução *definitiva*. Não foi previsto nenhum limite temporal para o ciclo de vida do novo sistema.

Esta consideração (ou falta dela) relativamente ao tempo de funcionamento do novo sistema compreende-se numa organização onde os recursos para as TI eram bastante escassos, e numa altura em que a área das comunicações ainda não tinha sido considerada estratégica dentro da ARI. O investimento de tempo e de custos acrescidos com servidores e recursos humanos externos, bem como o esforço de transição de conteúdos para a nova plataforma, seria difícil de justificar se a nova solução de CMS fosse apresentada como "temporária", mesmo que o prazo de utilização fosse bastante alargado, p. ex. cinco anos.

Esta falta de limitação do prazo de utilização do CMS tem o inconveniente de deixar demasiado em aberto aquilo que se propõe fazer com o novo sistema, já que este teria que servir quaisquer necessidades presentes e futuras da ARI e das ARNO. As seguintes questões levantavam-se na fase de planeamento do novo *website*:

- Que tipos de conteúdo temos agora? Texto, imagens. Queremos ter vídeos, formulários. Que tipos de conteúdo iremos ter dentro de 5 ou 10 anos?
- Temos cerca de 1 600 páginas agora. Quantas teremos no futuro?
- O tráfego do *website* mais do que triplicou entre 2003 e 2008. O número de páginas servidas por dia subiu de 1 200 para 4 500. Que aumentos de tráfego iremos ter nos próximos anos?
- O *website* deve servir as necessidades das presentes 16 ARNO—mas quantas existirão no futuro?
- Como interagimos com um *website* agora? Como o faremos no futuro?

Por razões de ordem prática, todas estas questões tiveram que ser adiadas. O que fizemos foi selecionar uma solução o mais genérica possível, confiar que o *website* seria adequado durante bastante tempo, e que seria possível levantar a questão no futuro, caso se verificasse a necessidade de reavaliar o *website*.

2.6 CRITÉRIOS DE ESCOLHA DO NOVO CMS

As vantagens e as limitações do CMS Carrelet, que estávamos a utilizar, influenciaram bastante

os critérios-base que foram utilizados para avaliar o novo CMS a adotar.

Os critérios finais relativos ao novo CMS foram definidos num documento interno. Esses critérios abrangem questões arquiteturais, funcionais, e de usabilidade, e podem ser sistematizados da seguinte forma:

1. *Estabilidade.* O sistema tem *bugs* que comprometam a sua execução?
2. *Segurança do sistema.* O CMS é vulnerável a intrusões? As plataformas ou tecnologias subjacentes são seguras?
3. *Escalabilidade de carga.* Qual a exigência do sistema em termos de recursos utilizados, tais como espaço em disco, RAM e CPU? Com que velocidade serve as páginas? Qual a capacidade real do novo CMS para acomodar grandes quantidades de documentos? Que carga impõe o CMS ao *hardware* subjacente?
4. *Extensibilidade.* O sistema é extensível, através de um *framework* potente e flexível?
5. *Roadmap.* Quais os planos de desenvolvimento futuro do CMS?
6. *Autenticação.* O sistema deveria permitir a cada utilizador registar-se, e ter a sua conta para edição (e no caso das páginas restritas, utilização) do *site*.
7. *Sistema de permissões incluindo permissão de leitura.* Pretendia-se um sistema que servisse tanto para o site público, quanto para a *intranet*, e que portanto permitisse ocultar páginas e outros documentos da vista do público em geral.
8. *Upload de ficheiros de qualquer tipo.* Pretendia-se um CMS que gerisse todo o tipo de ficheiros, e não apenas HTML e imagens.
9. *Atualização de documentos.* O novo CMS deveria permitir substituir qualquer documento por uma versão mais recente. No sistema anterior, para atualizar uma imagem era necessário fazer o *upload* da nova versão e substituir todas as referências à versão antiga.
10. *Versionamento.* O novo sistema deveria guardar versões anteriores de todos os documentos, e permitir reverter o documento para uma versão anterior.
11. *Histórico para auditoria.* O novo CMS deveria guardar registo de todas as operações efetuadas, juntamente com a data e o autor da alteração, para posterior auditoria.
12. *Gestão de documentos e de páginas web.* O CMS deveria servir não só como sistema de gestão de documentos, mas também para a criação de *websites* (ser, portanto, não apenas um CMS mas um *web content management system*).
13. *URL estáveis.* O endereço de um documento deveria permanecer válido, independentemente de se criar, eliminar ou reordenar qualquer outro conteúdo.
14. *Localização e internacionalização.* Quando estão a utilizar o CMS, as diferentes equipas em todo o mundo terão obrigatoriamente que ver menus e opções em língua inglesa, ou o sistema vem já traduzido, ou permite a tradução, em outras línguas?
15. *Gestão de documentos multilingue.* O novo CMS deveria permitir a gestão do conteúdo por línguas de forma fácil, e facilitar a tradução.
16. *Compatível com Unicode.* O novo CMS deveria permitir comunicar em todas as línguas das ARNO, incluindo búlgaro, checo e árabe, e estar preparado para suportar de futuro

outros *scripts* de línguas com projeção no mundo, como o chinês.

17. *Tipos de documento.* O sistema deveria permitir a criação de novos tipos de documento, e a alteração dos tipos existentes, de forma o mais livre possível. Isto permitiria servir as diferentes necessidades de apresentação de conteúdo, presentes e futuras, tanto das ARNO como da ARI.
18. *Design customizável através de templates.* O tipo de *template* não deveria estar ligado à secção, mas à página, de forma a permitir que dentro de uma mesma secção de conteúdo houvesse diversos tipos de páginas.
19. *Criação de subpáginas.* No sistema anterior, cada nova secção de conteúdo obrigava à criação de uma conta. O novo CMS deveria permitir a criação de conteúdo com os níveis de profundidade que se desejasse.
20. *Interatividade.* O site existente era considerado demasiado estático, pois não permitia que os visitantes colocassem comentários ou participassem em discussões. O novo CMS deveria permitir diversos tipos de interatividade entre os visitantes, tais como debates e comentários a páginas.
21. *Formulários.* O sistema anterior não permitia. Estes deveriam poder ser criados com quaisquer elementos possíveis de incluir numa página HTML, incluindo *tags* <label>.
22. *Feeds RSS.* Os visitantes deveriam poder subscrever o conteúdo de todo o *site*, ou de secções do *site*, via RSS.
23. *Curva de aprendizagem suave.* É fácil aprender a usar o CMS? O CMS transmite facilidade e simplicidade?
24. *Migração de conteúdos.* A migração dos conteúdos existentes pode ser feita automaticamente? Há ajuda para o trabalho manual de migração? O novo CMS permite exportação de dados em formato utilizável por outro CMS?
25. *Suporte técnico e documentação.* Qual a possibilidade de se obter ajuda para a utilização ou para a customização do CMS? Existe ajuda gratuita, por parte da comunidade?

2.7 CONCLUSÃO

A Rocha International parte para o debate sobre um novo CMS a partir de uma plataforma de *website*, de limitações económicas e de recursos humanos, e de uma estrutura organizacional particulares. Estas condições refletiram-se naturalmente nos critérios e na forma como a nova plataforma foi escolhida.

O conjunto de critérios definidos era no entanto bastante exaustivo e bem pensado, cobrindo aquilo que em 2007 se considerou importante para um *website* que deveria suprir as necessidades presentes e futuras de uma família de organizações em franco crescimento. A maioria desses critérios permanece atual ainda hoje.

Esta página foi intencionalmente deixada em branco.

3. TRABALHO RELACIONADO

Este capítulo apresenta uma análise das diferentes formas de atualizar um *website*, do funcionamento de um CMS, e de algumas das ferramentas de gestão de conteúdos existentes que foram analisadas durante o processo de seleção de um novo CMS para A Rocha.

3.1 ATUALIZAÇÃO DE WEB SITES

Qualquer website, por muito bem planeado que seja, está sujeito a atualizações mais ou menos periódicas, e mais ou menos profundas. Os websites são alterados por diversas razões, p. ex.:

1. *Atualização de informação.* Provavelmente a razão mais comum para atualizar um *website*. A publicação de uma nova notícia deve ser colocada no *website*; a saída de um funcionário da empresa leva à retirada da página sobre o mesmo; uma página que contém uma referência temporal ambígua (p. ex. "este ano…") deve ser corrigida ("em 2010…"); houve uma alteração de preços, e a tabela de preços *online* deve refletir essa alteração.
2. *Correção de erros.* As páginas podem ter erros factuais, ou de ortografia, a corrigir. Grande parte dos *websites* não tem um fluxo de publicação estruturado em que os conteúdos passem por diversas pessoas antes da sua apresentação ao público, o que leva à introdução de erros.
3. *Alteração de design.* O *design* de um website é por vezes alterado. Quando isso acontece, todas as páginas sofrem uma alteração—pelo menos uma alteração visual.
4. *Mudanças organizacionais.* Uma alteração de ramo de atividade ou de *marketing* vai implicar uma revisão de todo o conteúdo, já que aquilo que a entidade é, ou a forma como ela se "explica" a si mesma, sofreu uma profunda alteração.
5. *Mudanças tecnológicas.* Constantemente surgem novas e melhores formas de se resolver os problemas existentes com a gestão, atualização e apresentação de conteúdos para a web. Surgem novas tecnologias que substituem as anteriores. Um exemplo de mudança tecnológica é a transição de linguagem de marcação: HTML 4.01 → XHTML → XHTML + RDFa. Outro exemplo é o surgimento de *toolkits* (conjuntos de software para a web), como sejam o Google Web Toolkit, o Dojo e o Mootools, que modularizam certos problemas, e simplificam coisas anteriormente muito difíceis.
6. *Mudanças comportamentais.* O comportamento dos utilizadores da Internet muda, adaptando-se às mudanças na própria Internet. Os motores de pesquisa tornaram as listas de links, tão comuns na década de 1990, obsoletas; o mesmo sucedeu, com o

advento da Wikipédia, a muitos sites amadores "especialistas" sobre determinados conteúdos; as redes sociais virtuais criaram uma nova forma de interação entre utilizadores e com os *websites*; as compras e os donativos *online* estão em franco crescimento. Os *websites* são assim alterados de acordo com as mudanças na "ecologia" da Internet; p. ex. com a remoção de conteúdos obsoletos, a colocação de botões "like" nas páginas, e a criação de carrinhos de compras virtuais.

3.1.1 Atualização manual

Denomina-se "atualização manual" de *websites* o processo de criação de um *website* por uma pessoa, num computador, e o posterior envio das atualizações para o servidor de alojamento.

Este processo requer a instalação de diversos tipos de software:

- Aplicações de desenvolvimento para a web, como p. ex. Adobe Dreamweaver ou Aptana Studio;
- Software de FTP para transferência de ficheiros para o servidor; p. ex. FileZilla;
- Um web browser, para verificar os resultados.

As ferramentas de desenvolvimento habitualmente apresentam duas vistas para edição de páginas: uma vista de código que expõe a linguagem de marcação subjacente, e um modo WYSIWYG que é semelhante a um editor de texto. Boa parte das ferramentas de desenvolvimento são software pago.

O processo de atualização revela-se moroso e repetitivo. Apesar de algumas inovações nas aplicações de desenvolvimento para a *web*, é particularmente mal adequado para alterações que afetem todas as páginas de um *website*.

É também fácil introduzir erros de codificação nas páginas. Esse erros podem ser introduzidos pelo programador, quando funciona em vista de código, já que este tem controlo sobre a qualidade do código. Muitas vezes, é a própria aplicação de desenvolvimento web que introduz erros no código, quando o utilizador está a escrever a página em modo WYSIWYG.

Um exemplo: na ferramenta open source KompoZer, em modo WYSIWYG, normalmente quando o utilizador pressiona o botão *itálico*, o HTML resultante contém a marcação `<span style="font-style:italic">`. Isto não representa um erro completo, pois de facto o texto resultará em itálico. É, no entanto, um erro em termos de semântica, já que o HTML contém duas tags próprias para texto em itálico que são ignoradas: uma *tag* `<em>` ("emphasis"), para denotar texto em destaque e que é por defeito apresentada em itálico, e uma *tag* `<i>` que significa diretamente "texto em itálico". O mesmo se passa com as tags `<b>` e `<strong>`. Existe uma opção de configuração "usar tags HTML para formatação" que pode ser ligada; no entanto, fica a questão: porque é que alguém quereria usar formatação CSS *inline* (cujo uso é desaconselhado) para fazer aquilo que o HTML já faz?

É apenas um exemplo das "ideias próprias" do software de criação de páginas web; não é o exemplo mais extremo, mas revela como pode ser difícil para um programador trabalhar com essas ferramentas. Certos programadores consideram de tal modo desadequado usar os ambientes de desenvolvimento integrados para a *web*, que usam simples editores de texto para criarem *websites* completos.

A atualização manual de páginas tem as suas vantagens. Por exemplo, se as páginas criadas manualmente forem constituídas exclusivamente por conteúdo estático, poderá ser possível visualizá-las em qualquer computador, sem ser necessário recorrer a um servidor *web*.

Como regra geral, diríamos que a atualização manual deverá ser utilizada para *websites* até 20 páginas, no máximo, ponto a partir do qual se justifica a utilização de um CMS.

3.1.2 Caraterísticas de um CMS

Um sistema de gestão de conteúdos, ou CMS, é uma aplicação web que permite aos seus utilizadores criar, editar e organizar conteúdos para websites sem recurso à programação ([DOCF11]). Os CMS foram desenhados para ajudar pessoas sem formação técnica a publicar documentos de forma mais simples e mais automatizada.

Os CMS correm diretamente num servidor *web*, e são acessíveis usando apenas um *web browser*. Isso traz diversas vantagens:

1. *Sem software específico.* Não requerem a instalação de mais nenhum software, o que permite a uma pessoa gerir o website usando mais do que um computador, sem ter que adquirir licenças de software que cubram as diferentes máquinas.
2. *Sem dependência de sistema operativo.* Todos os sistemas operativos atuais têm um *browser* moderno que permite aceder ao CMS, enquanto que as ferramentas de desenvolvimento poderão não existir para o OS do utilizador.
3. *Desenvolvimento em equipa.* Diversas pessoas podem participar do processo de criação do *website*.
4. *Diversificação de recursos humanos.* É mais fácil pedir a alguém com um perfil menos técnico que use o seu *browser* para gerir algum aspeto do *website*, do que convencê-lo(a) a instalar e aprender a lidar com um *software* com perfil marcadamente técnico. O *web browser* é bastante menos intimidador. Essa diversidade de pessoas é benéfica para a construção de *websites*.

Um CMS é uma aplicação genérica de construção de websites. A sua estrutura pode ser mais ou menos rígida; regra geral, quanto mais rígida, mais simples é de usar, mas menos variada é nas suas possibilidades de uso; por outro lado, alguns CMS expõem muita da tecnologia subjacente ao utilizador do sistema, o que aumenta a sua funcionalidade em detrimento da facilidade de uso, e aumenta também a possibilidade dos próprios utilizadores do sistema introduzirem *bugs* no *website*.

Uma das vantagens dos CMS é que tratam dos elementos fixos das páginas. Este elementos, como sejam o logotipo, a navegação, o rodapé, a pesquisa, são mais trabalhosos de fazer numa atualização manual do que o conteúdo principal da própria página, e são a maior fonte de erros, como é comprovado pessoalmente por quem quer que já tenha construído sites manualmente.

É, no entanto, igualmente possível introduzir erros na codificação das páginas quando se usa um CMS. Os erros podem ter três origens:

1. *O utilizador final*, no caso em que o sistema permita a introdução direta de código, pode introduzir erros, p. ex. de HTML, JavaScript ou CSS.

2. *Os templates* ou modelos de página podem estar mal definidos pelos arquitetos do sistema.
3. *O próprio CMS* pode ter bugs, e isso efetivamente acontece. Por norma, as atualizações de software dos CMS não apenas aumentam a funcionalidade do sistema, como também corrigem *bugs* introduzidos em versões anteriores.
4. *O servidor web* pode introduzir erros nas páginas. P. ex. se o CMS produzir uma página em codificação ISO-8859-1 e o servidor web a fornecer marcada como sendo em UTF-8.

O utilizador final apenas tem controlo sobre o item n.º 1 acima. Os CMS são ferramentas de software bastante complexas, com a potencialidade de serem mal conhecidas; aliás, nenhum utilizador de um CMS conhece o sistema completamente.

Os CMS regra geral servem *páginas dinâmicas*—isto é, cada vez que uma página é solicitada, o CMS vai gerar essa página no momento. Apesar das grandes evoluções de *hardware* dos últimos anos, e de algumas evoluções no *software* que permitem gerar as páginas mais rapidamente, segundo [CHAL04] servir uma página dinâmica pode ser duas ordens de grandeza mais lento do que servir páginas estáticas. Mesmo que o conteúdo de uma página seja estático, a maioria dos CMS irá gerar essa página no momento.

Os motores de pesquisa foram criados para indexar páginas estáticas. Isto quer dizer que os CMS devem ter em consideração esta realidade: a página gerada pelo CMS deverá conter toda a informação possível para indexação, sem requerer nenhum tipo de interação da parte do utilizador, já que tal interação não se verifica durante a visita de um motor de pesquisa.

Regra geral, os CMS utilizam uma base de dados relacional para guardar os documentos, enquanto que nas atualizações manuais usa-se o sistema de ficheiros. Muitos CMS incluem um motor de pesquisa próprio, que indexa os conteúdos no sistema.

Para além de gerarem as páginas, os CMS geram elementos com base no conteúdo inserido no sistema, como sejam menus e submenus; navegação estrutural ("breadcrumb trails"); feeds RSS; pesquisa; e mapa do site. Esta geração de elementos com base em conteúdos não é possível num site manual, já que nem sequer existe a noção de conjunto de páginas.

3.1.3 Funcionalidades de um CMS

Estas são algumas das funcionalidades mais habituais presentes nos CMS. Para uma lista mais exaustiva de caraterísticas de diferentes CMS, ver [MATR].

Tabela 3.1: Funcionalidades mais habituais num CMS

Categoria	Funcionalidade
Segurança	Autenticação; Perfis de utilização; Sistema de privilégios por documento ou grupo de documentos; Versionamento; Histórico de auditoria.
Facilidade de utilização	Edição visual; Possibilidade de desfazer alterações; URL customizáveis; Notificações pela alteração de conteúdos específicos; Interface multilingue.

Categoria	Funcionalidade
Caraterísticas técnicas	Utilização de standards web, tais como XHTML, CSS, WAI; Codificação UTF-8; Vários tipos de *output* com base nos dados inseridos (p. ex. gerar um conteúdo como página HTML, como XML ou como PDF).
Gestão de conteúdos	Edição *inline*; Gestão do sistema totalmente online, sem necessidade de instalar *software*; Repositório de elementos, p. ex. imagens e vídeo; Reutilização de conteúdos; Gestão de traduções; Agendamento de conteúdos; Reutilização de conteúdos; Estatísticas de visualização; Agregação de metadados; Fóruns ou comentários de visitantes; *Feeds* RSS.
Comércio eletrónico	Carrinho de compras; Pagamentos eletrónicos; Gestão de inventário.

3.2 CMS ANALISADOS

Após a definição de critérios para o CMS (já referidos na secção 2.6, p. 9), foram analisados diversos sistemas existentes no mercado. Esta análise decorreu em três etapas:

1. Numa primeira etapa, foi analisado aquilo que seria de esperar de um CMS nessa altura, vistos diversos *websites* e dada uma rápida vista de olhos sobre os CMS que eles corriam. Foram aceites sugestões de CMS de outros *webmasters*, e de colaboradores tanto da ARI, como das ARNO.
2. Posteriormente, foram analisados em traços gerais sete CMS quanto à sua adequação à finalidade pretendida para o *site* `arocha.org`.
3. Dois desses CMS passaram à terceira fase, na qual os seus méritos foram comparados um com o outro, e concretamente contra a lista de critérios-base indicados em 2.6.

O *website* CMS Matrix, já referido, foi também usado no processo de escolha dos candidatos, de forma a se poder fazer comparações entre as caraterísticas de diferentes CMS. No entanto, a informação presente no CMS Matrix não está completamente atualizada, referindo-se por vezes a versões antigas dos CMS.

3.2.1 Daisy

O sistema Daisy será explicado em profundidade no capítulo seguinte. Por agora, podemos dizer que este CMS é constituído por dois componentes separados:

1. Um repositório de conteúdos, acessível tanto via HTTP/XML através da técnica REST, como via uma API Java;
2. Uma aplicação *front-end web* que corre sobre Apache Cocoon, que serve para a edição e publicação.

O CMS Daisy foi especialmente concebido para gestão de documentos, e tem uma boa escalabilidade, sendo usado para sites que indexam milhões de documentos (ver p. ex. [PATL]).

Este CMS não é muito divulgado, mas serve websites de vários tipos, como

organizações governamentais, empresas, instituições de ensino e entidades científicas. O sistema está concebido de modo ao *front-end* poder ser substituído por outro, o que por vezes acontece (ver [OT-LIVE]).

A informação sobre este sistema de gestão de conteúdos foi dada pela Dr.ª Marie Connett-Porceddu, que era CEO d'A Rocha International em 2007. A Dr.ª Connett-Porceddu havia anteriormente trabalhado no Cambia, uma organização internacional baseada na Austrália, que usava (e ainda usa) o CMS Daisy para os seus *websites*, entre os quais se conta o já citado Patent Lens.

Embora os *sites* que usam Daisy apresentem uma grande diversidade de *designs*, ao contrário de outros CMS não existem galerias de *layouts* Daisy disponíveis para *download*. As razões para este facto tornam-se evidentes ao se analisar melhor a forma como os *layouts* são aplicados no Daisy (ver secção 6.3.13, p. 78).

3.2.2 Drupal

O Drupal é um CMS desenvolvido em PHP, originalmente pelo programador belga Dries Buytaert (o nome "Drupal" é um inglesamento da palavra neerlandesa *druppel*, "gota de água"). A sua primeira release foi em janeiro de 2001, e encontra-se presentemente na versão 7.8; em 2007, a *release* corrente era o Drupal 5.

O Drupal é amplamente utilizado para a construção de *websites* de diferentes dimensões, sobretudo de *sites* produzidos por mais do que um utilizador, o que o colocou no âmbito dos CMS a verificar para A Rocha.

O sistema Drupal é constituído por duas partes distintas:

- *Drupal Core.* Esta é a release do Drupal propriamente dita, desenvolvida de forma modular, e contém a funcionalidade base do CMS.
- *Contribs*: os módulos contribuídos pela comunidade de utilizadores Drupal. Estes comunicam com o Drupal Core através de uma API. [DRUP-MOD] regista presentemente mais de 2 600 módulos compatíveis com a versão 5.x do Drupal.

A filosofia de desenvolvimento do Drupal encoraja o desenvolvimento de *contribs* pela comunidade, e por vezes as novas versões do *core* acabam por integrar *contribs* anteriores.

O Drupal requer o uso de uma base de dados para guardar os dados do *website* e as configurações de sistema; a presente camada de abstração de bases de dados suporta diversos tipos de BD, tais como PosgreSQL, mas o mais comum é o Drupal ser instalado sobre MySQL.

Entre as funcionalidades do *core* Drupal 5, contavam-se as seguintes ([DRUP-CORE]):

1. Agregador de *feeds* RSS
2. *Blogs*, comentários, fóruns e sondagens
3. *Caching* de conteúdos
4. Menus multinível
5. Pesquisa avançada
6. Registo e estatísticas de acesso
7. Sistema de controlo de acessos (p. ex. baseado no perfil ou no endereço IP do

utilizador)

8. Sistema de notificações de segurança e de atualizações
9. Suporte para múltiplos *websites*
10. Suporte para OpenID
11. Taxonomia hierárquica de conteúdos, o que permite atribuir *tags* às páginas
12. URL descritivos

Entre os principais defeitos do Drupal contam-se:

1. *Ser programado em PHP.* Por mais estável e bem programado que o *core* seja, os *contribs* muito provavelmente introduzem incongruências entre si, e terão um efeito cruzado sobre o *core* e uns sobre os outros, que aumenta a instabilidade do sistema. O PHP é uma linguagem feita para tornar fácil a programação *web*, mas em termos de segurança deixa um pouco a desejar (conforme já indicado na secção 2.5.1, p. 8).
2. Dependência de *contribs*. Uma instalação de Drupal que acomodasse o *website* `arocha.org` teria obrigatoriamente que incluir dezenas de *contribs*. A qualidade dos *contribs* nem sempre é controlada, e é plausível que seja inferior ao *core*. Haveria também que escolher cada funcionalidade entre dezenas de *contribs* disponíveis; p. ex., que *contrib* deveríamos usar para a gestão de imagens?

Os defeitos indicados fizeram com que surgissem suspeitas sobre a estabilidade e qualidade do produto final. Assim sendo, o CMS Drupal não foi considerado um dos candidatos finalistas a CMS para A Rocha.

3.2.3 Gadara

O CMS Gadara foi-nos indicado pelos seus criadores, Mark Howe e Steve Poulard, criadores também do CMS Carrelet anteriormente usado pel'A Rocha.

O desenvolvimento do Gadara foi financiado pela Igreja Metodista da Grã-Bretanha, para suportar a criação de uma igreja virtual, a St Pixels ([STPX]). O nome "Gadara" é uma referência a uma localidade da Palestina onde, segundo o relato bíblico, Jesus expulsou os demónios de um homem para uma vara de porcos ([MARC5]). Tal como o Carrelet, o Gadara foi programado em perl 5.

O Gadara foi criado de raiz para superar as limitações do Carrelet, em resposta ao *feedback* vindo da ARI e das ARNO:

1. Usava codificação UTF-8, permitindo assim a visualização de carateres em todas as línguas;
2. Usava uma base de dados em vez de um sistema de ficheiros, o que lhe permitia mais flexibilidade em termos de tipos de conteúdo;
3. Tinha um sistema "como deve ser" (nos termos dos próprios programadores) de utilizadores e de categorias de utilizadores, o que significa que era agora possível controlar o acesso a diferentes partes do site, e vários tipos de permissões de edição, utilizador a utilizador;
4. A estrutura de todo o site podia ser gerida a partir de uma localização centralizada,

contrariamente ao Carrelet onde cada diretoria ou secção era basicamente independente;

5. A linguagem de formatação usada para o Gadara permitia *layouts* de página mais sofisticados;
6. Tinha já *feeds* RSS;
7. Incluía um sistema completo de *email* interno, bem como *blogs*, comentários de página, e posteriormente foram-lhe acrescentados fóruns de discussão;
8. Os *templates* de página foram especificados em XSLT, muito mais flexível do que os templates Carrelet;
9. Permitia *upload* de documentos em formato PDF e de ficheiros de som;
10. Continha um sistema de notícias para as página principais de cada área do *site*, que permitia destacar "histórias" importantes por categoria, com uma data futura de publicação opcional.

Caso a ARI optasse pelo CMS Gadara, haveria apoio da parte da Cyberporte para a migração dos dados do Carrelet, o que era um grande ponto a favor do sistema. Seria possível migrar praticamente todo o conteúdo existente, mantendo a formatação, embora fosse ser necessária alguma intervenção manual para definir a estrutura do novo *website* com esses conteúdos, já que a estrutura seria alterada.

O Gadara não era código aberto, o que pesava contra a sua escolha. Se os programadores haviam deixado de manter o código do Carrelet por haverem mudado de paradigma de construção de websites, isso poderia acontecer também com o Gadara. Não obstante este contra, o Gadara foi um dos dois finalistas a CMS, e por isso foi analisado de forma mais intensa.

3.2.4 Joomla!

Joomla! é um CMS *open source*, que funciona igualmente como um *framework* para aplicações *web* segundo o paradigma de separação de funções Model–View–Controller (MVC). Corre sobre uma base de dados MySQL, e é programado em PHP, segundo o paradigma de programação orientada para objetos (OOP).

O Joomla! foi criado a partir de uma divisão da base de programadores do CMS Mambo, no ano de 2006. Aparentemente, as feridas causadas por essa fratura ainda não foram sanadas: ainda há animosidade entre os utilizadores que saíram para o Joomla! e os que ficaram no Mambo, e há uma base reduzida de programadores, o que dificulta o desenvolvimento do produto. Em 2007, o Joomla! era muito comparado com o Drupal (e ainda é, ver p. ex. a excelente infografia em [RUCK11]). Parecia ser preferido pelos *web designers*, enquanto o nicho do Drupal se adequava mais aos programadores.

Quem espreitou dentro do código do Joomla! diz horrores da experiência: código *spaghetti*, confuso e embrulhado, extremamente difícil de corrigir quando isso se torna necessário (ver p. ex. [KINN11]). O *core* Joomla! dava razão às desconfianças que pairavam sobre o PHP. Outros problemas apontados incluem:

1. Performance de origem bastante lenta;

2. URL de origem demasiado longos, sem grande riqueza semântica pois refletem apenas o nome da página;
3. Difícil de usar como *blog*;
4. Difícil de customizar o tema;
5. Problemas de funcionamento do sistema SEO de origem;
6. A maioria dos *plug-ins* bons são pagos;
7. Muita variação na qualidade (e preço) dos programadores Joomla!, com muitos deles a criarem código para versões antigas do CMS.

Além do acima descrito, o Joomla! não foi considerado na altura suficientemente poderoso para as necessidades de `arocha.org`, e foi considerado para a última fase de seleção.

3.2.5 MediaWiki

MediaWiki é um CMS *open source*, escrito em PHP e correndo sobre base de dados MySQL ou outra. Foi desenvolvido pela Wikimedia Foundation, e corre todos os seus *sites*, incluindo a Wikipédia, Wiktionary, Wikinews e o seu *website* institucional.

O tipo de *website* a que o MediaWiki se adequa é o chamado *wiki*: um *site* em todas páginas têm a mesma importância. As páginas podem ser classificadas em categorias, que funcionam de alguma forma como *tags*. O MediaWiki não é adequado a *websites* hierárquicos, com sistemas de menus. Este CMS era à partida o favorito do autor, pelas seguintes razões:

1. O autor era voluntário na Wikipédia desde 2005, e estava bastante familiarizado com a sintaxe de formatação MediaWiki. Considerava a sintaxe extremamente acessível, já que os *websites* da Wikimedia Foundation têm milhares de editores ativos.
2. A estrutura que o autor havia idealizado para o website era também ela enciclopédica, com uma grande quantidade páginas de informação, e onde os conteúdos seriam bastante ligados entre si (*cross-linking* das páginas), de modo a refletir a grande quantidade de páginas no *website*, e a variedade e interligação entre as diferentes áreas de atuação da organização por todo o mundo.
3. A construção do menu é uma questão complexa: onde colocar cada página de informação? De notar que `arocha.org` já contava com mais de um milhar de páginas, e que não estavam estabelecidos limites para o número de páginas que o *website* pudesse vir a ter no futuro. O problema do menu ficaria trivialmente resolvido com a adoção do MediaWiki, já que num site MediaWiki não é suposto sequer haver um menu.

O autor verificou estar errado quanto à aceitação da sintaxe de formatação do MediaWiki. Houve bastante resistência à adoção desta da parte das pessoas que participaram no Puzzled Puffin, o que fez pensar que seria melhor ter um editor WYSIWYG.

A estrutura idealizada pelo autor devia-se a um pendor pessoal para conteúdos enciclopédicos, e não era adequada aos objetivos da organização. Num site *wiki*, a descoberta da informação é um objetivo em si mesmo, enquanto que um *site* duma ONG de ambiente deve estruturar-se de forma a envolver os visitantes no seu trabalho, com chamadas à ação claras: voluntarie-se; visite-nos; faça um donativo. Assim, o MediaWiki foi igualmente colocado de

lado nesta fase.

3.2.6 Plone 3

Plone é um CMS de código livre, construído sobre a plataforma Zope, e programado em Python. Está em desenvolvimento desde 1999. O nome do CMS deriva da banda de música minimalista Plone ([PLON-NAME]).

O Plone está muito bem cotado em termos de segurança (ver figura seguinte, segundo [WALS11]). É muito usado por organismos governamentais. Em Portugal, o Plone é usado p. ex. pela Entidade Reguladora da Saúde, e pela própria FCT-UNL ([PLON-SITE]).

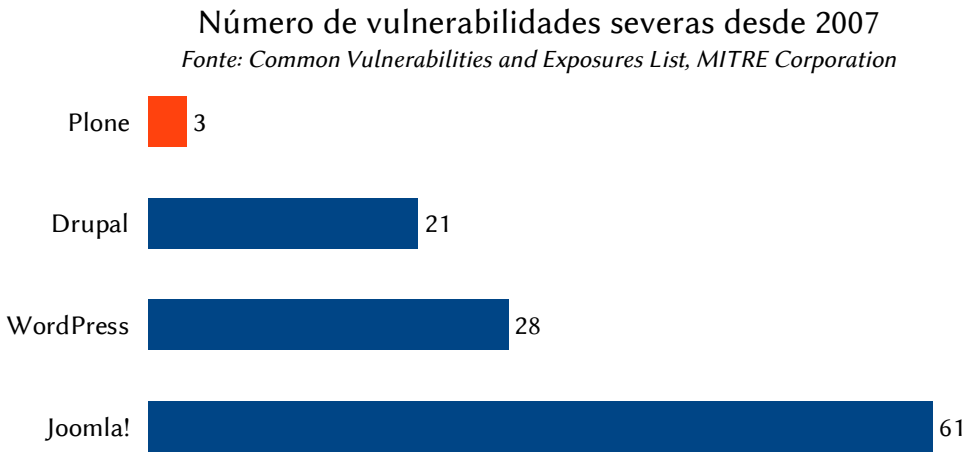

Figura 3.1: Vulnerabilidades severas de alguns CMS

Em maio de 2007 estava em fase de finalização a versão 3 do Plone, que foi lançada em agosto desse ano ([PLON07]). Estas são as principais funcionalidades dessa versão:

1. Edição *inline*: duplo clique num documento para o editar
2. Editor HTML visual
3. Suporte para múltiplos formatos de marcação (p. ex. Markdown ou Textile), extensível
4. Bloqueio automático de documentos em edição
5. Verificação automática de *links* e referências no documento
6. Reorganização de conteúdos intuitiva através de clicar, arrastar, cortar, copiar e colar
7. Rascunhos de documentos: fazer edições num documento para publicar no futuro
8. Registo de versões anteriores de documentos (versionamento)
9. Indexação completa de documentos em MS Word ou PDF
10. LiveSearch (exibição de resultados de pesquisa logo que se começa a escrever os termos de pesquisa)
11. Coleções de documentos, com geração automática de *feeds* RSS
12. Inserção automática de *links* para documento anterior / próximo dentro de uma pasta
13. Suporte para sítios *wiki*

14. *Triggers* de conteúdo (p. ex. enviar *email* automaticamente quando um documento é criado; mudar um documento de local após determinado tempo)
15. Geração automática de índices de documentos
16. Motor de *portlets*: permite formatar e mostrar *feeds* de qualquer *site*
17. URL legíveis
18. Navegação flexível
19. Templates customizáveis, aplicáveis a um documento ou a uma pasta
20. Tipos de conteúdo nativos poderosos (p. ex. imagens com redimensionamento e geração de *thumbnails* automática, eventos exportáveis para calendários), extensíveis através do *framework* Archetypes, que permite novos campos, *widgets* e validação de propriedades
21. Possibilidade de fazer comentários nas páginas
22. Fluxos de trabalho customizáveis
23. Gestão de traduções
24. Agendamento de conteúdos (publicação automática em data futura)
25. *Back-end* de autenticação flexível, e suporte para OpenID
26. XHTML e CSS bem formados, e site acessível segundo a norma WAI
27. RSS gerado automaticamente (p. ex. para pastas, ou para resultados de pesquisa)
28. Suporte para o protocolo auxiliar de motores de busca Sitemap
29. Suporte para linguagens escritas da direita para a esquerda
30. Biblioteca de *add-ons*
31. Suporte para montar qualquer secção do *site* como uma pasta FTP ou WebDAV
32. Compressão de recursos (gzip)
33. Integração com *cache proxy*
34. Exportação em XML da configuração do *site*
35. "Hot backups" (sem *downtime*)

A conclusão sobre o Plone, escrita pelo autor em 2007, diz: "O Plone permanece uma opção viável, mas parece demasiado flexível para ser utilizável sem bastante trabalho preparatório: aparenta ser mais um conjunto de peças para montar um CMS, do que um CMS já montado." Portanto, o Plone não foi um dos dois CMS finalistas. No entanto, verificou-se depois que o Daisy poderia muito ser descrito pelas mesmas palavras, portanto talvez o Plone tivesse sido mesmo uma boa escolha. O autor definiu como seu objetivo experimentar o Plone quando tiver que criar um *website* novo para a ARI.

3.2.7 SPIP

SPIP é um CMS que se define como sendo apto para trabalho coletivo e para conteúdos em múltiplas línguas ([SPIP]). SPIP significa "Système de Publication pour l'Internet Participatif", e está em desenvolvimento desde 2001.

O SPIP é programado em PHP e corre sobre base de dados SQL. As páginas são

geradas dinamicamente com recurso a esqueletos de apresentação ("squelettes") que misturam HTML com a notação própria do SPIP. O sistema contém uma cache com imagens das páginas, e permite configurar a idade máxima de uma imagem de página na cache, antes que esta seja criada novamente a partir da informação contida na base de dados.

Em 2007, o SPIP não parecia ter um grupo de programadores que fizessem uma manutenção decente do sistema e assegurassem a sua continuidade. Na altura, o sistema tinha bastantes *bugs*. Para além disso, os sites feitos em SPIP não entusiasmam de forma alguma em termos de design, o que dá a ideia que, por mais flexível que seja o sistema de *templates*, não deve facilitar muito o processo de *web design*. Por estas razões, o SPIP acabou por não ter mais do que uma vista de olhos superficial, e não foi considerado como finalista para CMS da ARI.

3.3 INTERATIVIDADE NO WEBSITE

O objetivo do Gadara era providenciar oportunidades para o maior número de pessoas possível contribuir com conteúdos de forma apropriada. Segundo os seus programadores, isto iria tornar o *site* mais interessante de visitar, e encorajar visitas repetidas. O Gadara havia sido feito para facilitar a tarefa de monitorizar conteúdo inserido pelos utilizadores, algo que não sucedia com o Carrelet. Isto criaria a necessidade de moderação de conteúdos. Por razões de limitação de tempo foi decidido que, embora pudesse ser uma boa ideia para o futuro, no momento atual a ARI não dispunha de recursos humanos para fazer funcionar devidamente um *site* com contribuições dos visitantes.

A decisão pela não-interatividade da nova versão do *website* retirou peso a muitas das funcionalidades do Gadara, e fez pender a solução para o Daisy.

3.4 OS DOIS FINALISTAS: DAISY E GADARA

Entre os sete CMS analisados para uso em `arocha.org`, dois foram selecionados como sendo mais promissores: Daisy e Gadara. A tabela seguinte compara os dois CMS, tomando por base os critérios de funcionalidade, usabilidade e técnicos indicados na secção 2.6, p. 9.

Tabela 3.2: Comparação entre Daisy e Gadara

Item	Comparação entre Daisy e Gadara
1. Estabilidade do sistema	Ambos os sistemas pareciam suficientemente estáveis. Considerou-se que o *design* modular do Daisy poderia aumentar a possibilidade de instabilidade devido a interações imprevistas entre componentes de origens diferentes.
2. Segurança do sistema	Ambos os sistemas foram considerados seguros. Não foi no entanto efetuada nenhuma auditoria aos mesmos.
3. Escalabilidade de carga	Ambos os sistemas geravam páginas sob pedido, e não continham nenhuma *cache* de origem, pelo que seriam ambos lentos comparativamente com um sistema estático.

Item	Comparação entre Daisy e Gadara
4. Extensibilidade	A extensibilidade do Daisy era bastante superior. Permitia transformar via XSLT qualquer componente *web* e servi-lo como uma página Daisy.[1] Para além das API era extensível através de módulos programados sob o *framework* Cocoon.
5. *Roadmap*	Os criadores do Gadara haviam contratado um programador para continuar o desenvolvimento do produto. Sendo o Daisy *open source*, esperava-se que toda a comunidade de utilizadores contribuísse para o seu desenvolvimento.
6. Autenticação	Cada utilizador teria a sua própria *password*, em vez do sistema utilizado no Carrelet (*passwords* partilhadas por utilizadores e definidas para cada secção do website).
7. Sistema de permissões incluindo permissão de leitura	Ambos os sistemas permitiam controlar o acesso de leitura, sendo portanto aptos para gerir não só o *site* público mas também uma *intranet*. Enquanto o Gadara permitia controlar o acesso ou edição ao nível da secção do *website*, o Daisy contava com um ACL, simples de usar mas poderoso, com uma granularidade ao nível da variante de documento, e permitindo definir tipos de utilizador (*roles*).
8. *Upload* de ficheiros de qualquer tipo	Em ambos os casos, quaisquer utilizadores poderiam fazer *upload* de ficheiros, desde que tivessem as permissões necessárias. O sistema do Daisy parecia mais atraente, com uma gestão de ficheiros mais simples.
9. Atualização de documentos	O Daisy tinha esta funcionalidade. Não foi possível verificar isto relativamente ao Gadara.
10. Versionamento	O Daisy incluía esta funcionalidade; o Gadara não.
11. Histórico para auditoria	O Daisy incluía esta funcionalidade; o Gadara não.
12. Gestão de documentos e de páginas *web*	Ambos os sistemas estavam vocacionados para servirem páginas em HTML. O Daisy permitia a exportação das páginas em formato PDF, e o acoplamento de outros módulos de exportação para outros formatos.
13. URL estáveis	Ambos os sistemas permitiam criar URL que ficavam estáveis indefinidamente, não sendo alterados quando outra página fosse criada.

1 Por exemplo, o website do Cambia incluía o blog do seu diretor, o Dr. Richard Jefferson, produzido em WordPress mas servido dentro do Daisy ([RAJ11]).

Item	Comparação entre Daisy e Gadara
14. Localização e internacionalização	O interface do Gadara era exclusivamente em inglês, e não previa internacionalização. O interface do Daisy estava já traduzido para cinco línguas.
15. Gestão de documentos multilingue	O Daisy havia sido criado de raiz com variantes, organizando conteúdos por língua e facilitando a gestão de traduções. O Gadara não tinha esta funcionalidade.
16. Compatível com Unicode	Ambos os sistemas funcionavam em UTF-8.
17. Tipos de documento	O Gadara permitia apenas dois tipos de documento: páginas de topo (p. ex. inícios de secção), e outras páginas. O Daisy permitia a definição de quaisquer tipos de documento.
18. *Design* customizável através de *templates*	Ambos os sistemas permitiam criar *templates* em XSLT, mas enquanto o Daisy permitia vários *templates* por *website*, o Gadara apenas permitia um.
19. Hierarquia flexível	O Daisy fornecia total liberdade em termos da hierarquia do *website*, através dos documentos de navegação.
20. Interatividade	O Gadara estava pensado para fomentar a criação de comunidades *online*. Incluía fóruns e inquéritos na sua configuração base. O Daisy não.
21. Formulários	Ambos os sistemas permitiam a criação de formulários. O Gadara permitia igualmente a criação de inquéritos.
22. *Feeds* RSS	Ambos os sistemas tinham esta funcionalidade.
23. Curva de aprendizagem suave	O Daisy tinha um editor WYSIWYG, permitia cortar e colar texto vindo do MS Word. No Gadara, a formatação seguia um sistema de marcação proprietário.
24. Migração de conteúdos	Os programadores do Gadara ofereceram-se para desenvolver uma pequena aplicação para converter páginas entre o sistema antigo e o novo. Nenhum dos CMS previa a exportação de configurações ou conteúdos para outro CMS.
25. Suporte técnico e documentação	O Gadara não tinha documentação escrita; o Daisy sim. Em matéria de suporte técnico, havia uma grande proximidade com os dois programadores do Gadara, pelo que quaisquer questões seriam respondidas com celeridade; mas o Daisy era *open source* e tinha uma base de utilizadores maior.

3.4.1 Vantagens adicionais do Gadara

Para além do apontado acima na comparação entre os dois CMS finalistas, escolher o Gadara

apresentava ainda estas duas vantagens sobre a possibilidade de escolher o Daisy:

1. *Poder ficar com a equipa da Cyberporte.* Os dois programadores tinham-nos servido bem durante vários anos. A Cyberporte havia decidido não suportar nenhum outro CMS para além do Gadara, portanto adotar outro CMS seria obrigatoriamente mudar de fornecedor de alojamento.
2. *Desenvolvimento da ferramenta.* Os programadores do Gadara disseram que poderiam incluir algumas da funcionalidades que a ARI pretendia.
3. *Não ter que mover o email.* Pelas razões apontadas acima, caso escolhêssemos outro CMS teríamos que mudar também de servidor de *email*, o que implicaria voltar a criar os 180 endereços de *email* existentes.

Relativamente ao ponto anterior, a ARI continuou a ser cliente da Cyberporte para o serviço de email até janeiro de 2009, altura em que se contratou o serviço de *email* da Rackspace, que possibilita acesso via IMAP para além de POP, com ligação encriptada.

3.4.2 Vantagens adicionais do Daisy

Adotar o Daisy apresentava ainda estas vantagens sobre o Gadara:

1. *Inserção de código HTML.* O Daisy permitia-nos inserir trechos arbitrários de HTML em qualquer parte de uma página. Por não permitir inserir código, o Gadara tornava impossível criar um mapa clicável, ou inserir um vídeo do YouTube.
2. *Agendamento de conteúdos.* Podemos criar uma página e dar-lhe uma data de publicação; nesse dia, a página ficaria visível.
3. *Pesquisa.* O Daisy incluía o seu próprio motor de indexação de texto (Lucene), por isso a caixa de pesquisa nas nossas páginas deixaria de depender do Google.

3.5 CUSTOS DE MUDANÇA DE CMS

Mudar de CMS acarreta custos, que devem ser planeados de antemão. Alguns desses custos são despesas adicionais, ou aumento de despesas existentes, e outros são custos em termos de tempo de trabalho da parte das pessoas que já estão a trabalhar para a ARI. Estes são os custos que foram antecipados pelo autor, e indicados na altura aos diretores da ARI:

1. *Tempo de aprendizagem do webmaster.* O autor deveria dominar o novo sistema.
2. *Tempo de aprendizagem dos outros editores.* Todas as outras pessoas com funções que implicassem introdução de conteúdos no *website* deveriam saber trabalhar com ele.
3. *Tempo de conversão de conteúdos.* Havia na altura em `arocha.org` cerca de 1 500 páginas e 1 800 imagens. Se fossem calculados 5 minutos por imagem, e 10 por página, apenas mover os conteúdos levaria 400 horas de tempo de trabalho.
4. *Custos de alojamento.* A previsão de custos de alojamento apontava para uma fatura duas a três vezes superior, após a mudança para o novo CMS.
5. *Realojamento dos emails.* Seria necessário mudar os emails para outro servidor, conforme já mencionado.
6. *Custos imprevistos.* Sempre surgem outras despesas que não se estava a prever, seja

diretamente no processo de mudança de CMS, seja em consequência deste. P. ex. uma consequência da mudança de CMS foi que a ARI contratou serviços de DNS separados.

3.6 CONCLUSÃO

As recomendações do autor relativamente ao novo CMS para `arocha.org` foram apresentadas à CEO da ARI no documento [REIS07b]. Entre os dois CMS finalistas, o Gadara era o que se apresentava menos flexível e menos potente. O autor recomendou que a ARI adotasse o Daisy, o que veio a acontecer.

4. DESCRIÇÃO DO SISTEMA

Este capítulo descreve em profundidade a infraestrutura e os componentes do CMS Daisy, versão 2.2 ([OUTE10]), que está presentemente em uso em `arocha.org`.

4.1 INSTALAÇÃO

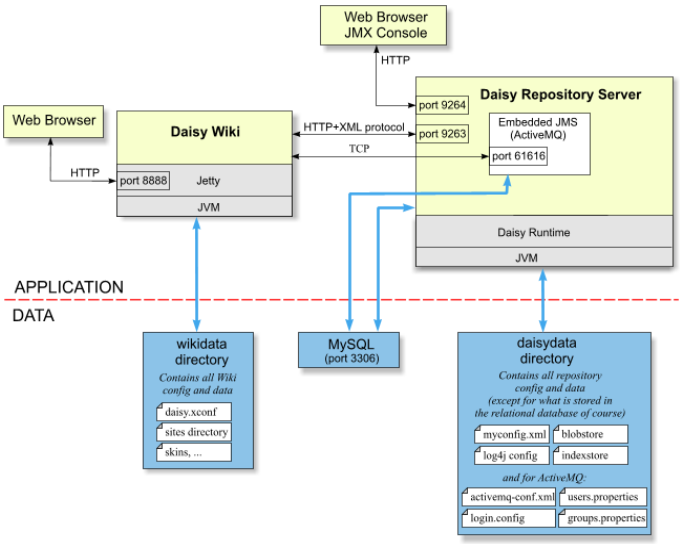

Figura 4.1: Uma instalação padrão Daisy

O Daisy corre sobre diversos sistemas, como Mac OS X, GNU/Linux e Windows 2000 em diante. O Daisy necessita de uma Java Virtual Machine e de um servidor MySQL. O pacote de instalação do Daisy contém o servidor HTTP Jetty. Foi desenvolvido sobre o *framework* Apache Cocoon, versão 2.2 ([APAC08]).

Uma instalação padrão Daisy é constituída por três componentes: o servidor-repositório, a base de dados MySQL e o *front-end* Daisy Wiki. Estes componentes podem estar em servidores separados. A BD MySQL pode ser substituída por outra com igual funcionalidade para a qual exista um API Java. O *front-end* de publicação pode ser substituído, ou até eliminado, no caso em que não se pretenda funcionalidade gráfica, mas apenas um gestor de documentos acessível através de API.

O Daisy guarda as partes dos documentos fora da base de dados, numa diretoria `daisydata/blobstore` do disco rígido, de forma a tornar mais eficientes as operações sobre os dados.

4.2 SERVIDOR-REPOSITÓRIO

O servidor-repositório é o componente principal do Daisy. Fornece toda a funcionalidade de gestão de conteúdos, sem ter um GUI. É constituído por um *core* e por extensões que lhe acrescentam algumas funcionalidades. O repositório pode ser acedido através de HTTP+XML, ou da API Java nativa. Vejamos algumas das suas principais funcionalidades.

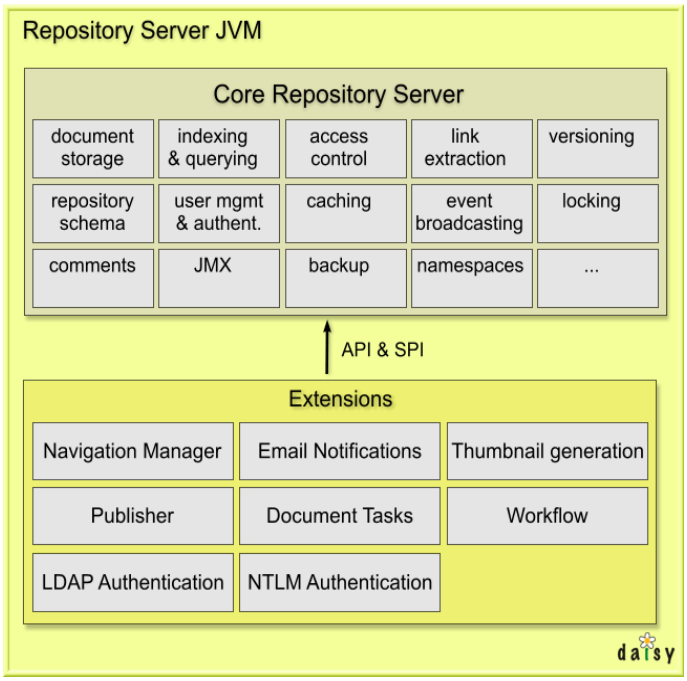

Figura 4.2: Core e extensões do servidor-repositório

4.2.1 Motor de indexação Jakarta Lucene

O Daisy indexa automaticamente o conteúdo dos documentos, sempre que estes são alterados. Apenas é indexada a versão publicada das variantes, e não todas as outras. O motor indexa o título dos documentos, o conteúdo dos campos, e o conteúdo das partes se estas tiverem um dos tipos MIME suportados: texto simples, (X)HTML, XML, ou documentos em PDF, OpenOffice Writer, Word, PowerPoint ou Excel.

4.2.2 Access Control List

Uma ACL é um conjunto de regras que define as permissões de acesso aos objetos num CMS. O Daisy tem uma ACL global que gere as permissões de todas as variantes, e que é avaliada de cima para baixo, com as regras posteriores a anularem o efeito das anteriores, se se aplicarem à mesma variante. Cada regra está ligada a uma condição, que indica em que casos esta se aplica.

Há quatro tipos de operações sobre as variantes: leitura, escrita, publicação e eliminação. Para cada uma destas operações, cada regra da ACL indica se a operação é negada, se é permitida, ou se as suas permissões não são alteradas. As permissões de leitura estão ainda diferenciadas segundo o tipo de objeto: a versão publicada, as versões antigas, os campos da variante, o índice de texto e o sumário do documento. É possível assim dar permissão de leitura a alguns destes componentes, e não a outros.

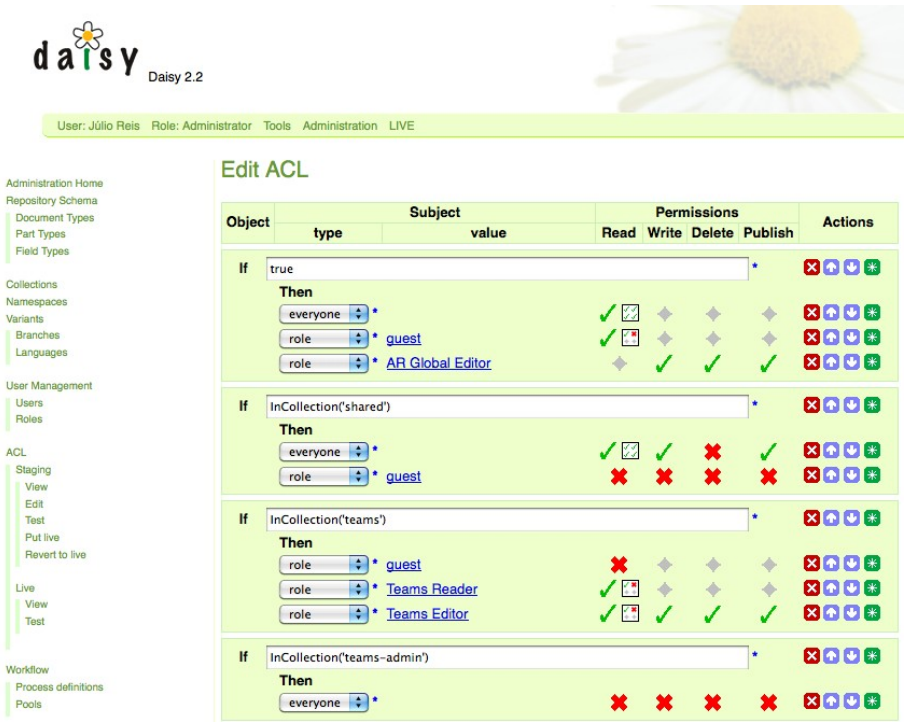

Figura 4.3: Editor da ACL

A ACL tem uma versão de trabalho ('staging') e uma versão ativa ('live'). As alterações são feitas na versão de trabalho, que pode ser testada antes de ser colocada como ativa.

A avaliação das permissões é feita segundo esta ordem:

1. Se o utilizador é Administrador, detém todos os direitos, e a ACL não é avaliada.
2. Se o utilizador é o proprietário do documento, tem permissões de leitura, escrita e eliminação da variante. A ACL é avaliada para determinar permissões de publicação.
3. Se o documento estiver marcado como privado, e o utilizador não for proprietário do mesmo, todas as permissões são negadas. A ACL não é avaliada.
4. As permissões são inicializadas com todas as operações negadas, e a ACL é avaliada, de cima para baixo. Se a condição de uma regra se verificar, as permissões da regra são aplicadas. A avaliação não termina por aqui: todas as regras são avaliadas.
5. Depois da última regra da ACL, se o utilizador não tiver acesso de leitura, são-lhe negadas todas as outras permissões. Se o utilizador não tiver permissão de escrita, é-lhe negada a permissão de eliminação.

A implementação concreta da ACL encontra-se na secção 5.4.4, p. 51.

4.2.3 Publisher

O propósito básico do componente Publisher é devolver em formato XML toda a informação necessária para mostrar uma página. Pode ser no entanto usado para outros fins, como produzir um relatório de diferenças entre duas versões de uma variante.

O Publisher é extremamente poderoso, e é a base da transformação do conteúdo da base de dados nas páginas servidas pelo Daisy.

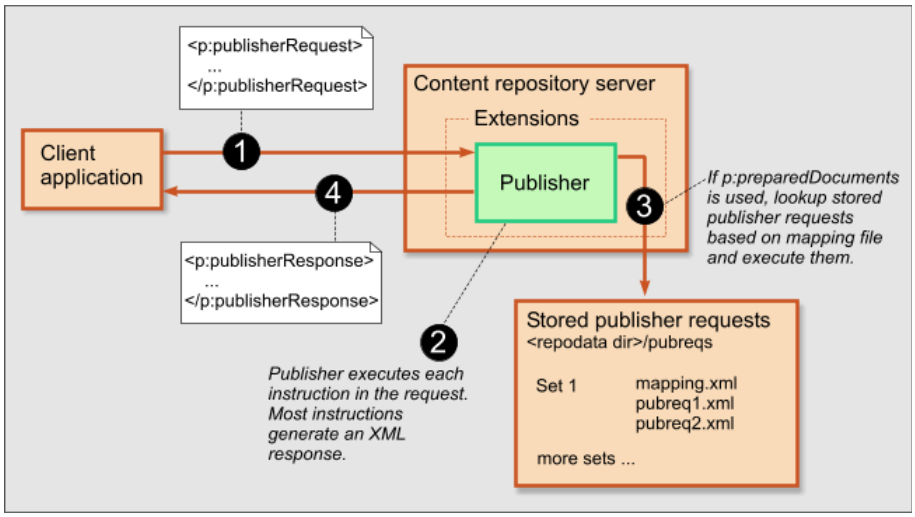

Figura 4.4: Componente Publisher

4.2.4 Outras funcionalidades do repositório

O servidor-repositório tem outras funcionalidades, entre as quais as seguintes:

1. *Notificações via email.* O Daisy pode ser configurado para enviar email quando se verificarem algumas condições, como o registo de um novo utilizador, a colocação de um comentário ou a criação de um documento.
2. *Gestor de tarefas sobre documentos.* Apresenta a possibilidade de se selecionarem variantes e se executarem ou agendarem *scripts* JavaScript sobre elas. Uma das tarefas mais utilizadas é a conversão de documentos de um tipo para outro.
3. *Backups.* O Daisy permite bloquear as edições sobre os dados de forma a que se faça uma cópia de segurança coerente.

4.3 DAISY WIKI

O *front-end* do CMS Daisy chama-se *Daisy Wiki*. Conta com um editor WYSIWYG, uma componente de navegação, e expõe todas as funcionalidades do servidor-repositório, tais como diferentes tipos de documento e um mecanismo flexível de pesquisa.

4.3.1 Edição de documentos

Um documento em Daisy pode ter múltiplas partes e campos, segundo o que for definido no seu tipo de documento. O editor de documentos da Daisy Wiki adapta-se automaticamente às partes existentes.

Para a edição das partes marcadas como "Daisy HTML", a Daisy Wiki utiliza um editor WYSIWYG incorporado, o HTMLArea. Este editor foi fortemente customizado para o Daisy, permitindo funções específicas para este CMS, como a previsualização de documentos incluídos. Isto faz com que os diretores de desenvolvimento não considerem mudar para outro editor, apesar da versão padrão do HTMLArea não ser atualizada desde 2007 (ver [NOEL10]). O HTMLArea não é suportado pelos *browsers* Safari ou Google Chrome.

Para outras partes de documento, é usado um controlo de *upload*, mas é possível

implementar editores de partes customizados, conforme foi feito para o editor de navegação.

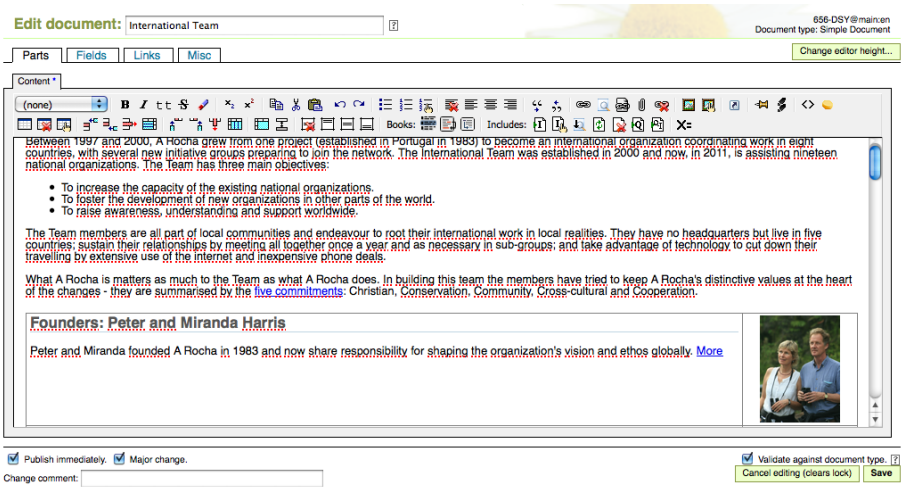

Figura 4.5: Editor WYSIWYG HTMLArea

O editor HTMLArea restringe o HTML aceite a um subconjunto da linguagem, compatível com XML. Este conteúdo é estandardizado e formatado com recurso a uma ferramenta de limpeza de código que corre no servidor, de forma a que o resultado possa ser facilmente transformado e comparado entre versões. É possível alterar a forma como essa ferramenta de limpeza funciona, de forma a permitir outras *tags* ou outros atributos (ver na secção 5.7.6, p. 62 como foi acrescentada uma tag `<captcha/>`).

O editor permite a inserção de imagens, através de pesquisa no repositório ou do carregamento de imagens novas. As imagens são um tipo de documento no servidor--repositório, de forma que têm variantes e versões como qualquer outro documento.

O editor de documentos cria editores para os campos de acordo com o seu tipo de dados, mas tal como com as partes, é possível criar editores customizados para campos.

4.3.2 Preparação de documentos

Entre as capacidades do modelo de preparação de documentos do Daisy Wiki, destacam-se:

1. *Inclusão de documentos.* É possível incluir dentro de um documento outros documentos dentro do repositório, outras páginas da Internet, ou mesmo extensões programadas sobre Apache Cocoon.
2. *Inserção de consultas.* É possível incluir *queries* DQL nas páginas, e controlar a formatação da apresentação dos resultados (ver 5.6.1, p. 55).
3. *Transformação via XSLT.* O documento a servir é formatado usando um *template* para cada tipo de documento, quer o resultado seja apresentado como página web, como inclusão dentro de outro documento, ou via *feed* RSS.
4. *Substituição de variáveis.* Os documentos podem fazer referência a variáveis, que são definidas num único local e incluídas durante a publicação. Um uso típico pode ser facilitar a edição ou evitar a repetição de elementos, como p. ex. definir a variável `{CO2}` para o valor "CO_2" de forma a evitar que os editores escrevam "CO2".
5. *Publicação de PDF.* A Daisy Wiki usa XSL-FO através de Apache FOP, com as mesmas

funcionalidades da publicação em HTML (ou seja, *templates* definidos por tipo de documento).

A Daisy Wiki não tem nenhum tipo de *cache* de documentos: visto que a publicação de um documento depende dos direitos de acesso do utilizador atual, os documentos são sempre publicados dinamicamente.

4.3.3 Skins

O visual da Daisy Wiki pode ser customizado através das chamadas *skins*. Uma *skin* consiste em vários recursos tais como ficheiros CSS, imagens, possivelmente algum JavaScript, e folhas de estilo XSLT. É possível fazer alterações gráficas básicas alterando os ficheiros CSS, enquanto que as mudanças nos *templates* XSLT permitem modificar a estrutura da página por completo.

O nome da *skin* que um *site* deve usar está definida no ficheiro de configuração do *site*, `siteconf.xml`. A skin propriamente dita estará em `resources/skins/<nome da skin>`.

O mecanismo de skins suporta a herança de recursos entre *skins*; isto permite criar uma *skin* derivada de outra, incluindo apenas os recursos que sejam diferentes da *skin* anterior.

O *skinning* de um documento segue as seguintes etapas:

1. É feito um pedido `<p:publisherRequest>/<p:document>` ao módulo de preparação de documentos, que responde com `<p:publisherResponse>/<d:document>`.
2. Este elemento é passado à folha de estilos correspondente ao tipo de documento, dentro da *skin*. P. ex. se a skin se chamar 'trueblue' e o documento for do tipo MultiColumn, o documento será passado para `resources/skins/trueblue/document-styling/html/MultiColumn.xsl`.
3. O resultado será ainda passado à folha `layout.xsl`, que irá colocar a "mobília" da página, p. ex. a navegação.

O resultado de uma consulta pode ser formatado segundo modelos especificados na DQL com a cláusula `OPTION style_hint = '<formato>'`, ao qual deverá corresponder um elemento `<xsl:template match="d:searchResult[@styleHint='formato']">` dentro da folha de estilos `<skin>/xslt/query-styling-html.xsl`. Ver exemplo na secção 5.6.2, p. 56.

4.3.4 Navegação hierárquica e definição de URL

É possível definir tantas árvores de navegação quantas se queira, de forma a providenciar formas de navegação hierárquicas.

O Daisy inclui um editor customizado para árvores de navegação, de forma a ser possível criá-las sem recurso a codificação em XML.

Uma árvore de navegação é um documento em XML do tipo "Navigation". Pode conter nós indicados individualmente; consultas que gerem listas de nós; pastas de documentos; URL arbitrários; e outras árvores de navegação. Quando uma árvore de navegação é gerada, os nós são filtrados segundo as regras de acesso do utilizador que a pediu.

A árvore de navegação controla a definição de URL: o caminho corresponde à posição do documento na árvore; p. ex. um documento com o ID `123-DSY` servido na raiz da árvore de navegação terá um caminho `/123-DSY.html`. Se o documento `456-DSY` estiver incluído na

árvore sob o elemento `123-DSY`, o seu caminho será `/123-DSY/456-DSY.html`.

É possível customizar os elementos do caminho diretamente na árvore de navegação, de forma a torná-los mais legíveis por seres humanos, e mais relevantes do ponto de vista da indexação pelos motores de pesquisa. Se o documento `123-DSY` for customizado com o nome 'research', e o `456-DSY` com o nome 'storm-petrel', os exemplos acima terão os caminhos `/research.html` e `/research/storm-petrels.html`, respetivamente.

Um documento pode ser incluído mais do que uma vez numa árvore de navegação. O mesmo documento `456-DSY` pode ser incluído como 'seabirds' sob um documento `789-DSY` com o nome 'portugal', e assim ser servido também sob o URL `/portugal/seabirds.html`.

É sempre possível obter um documento com base no seu ID; no exemplo acima, um pedido por `/456-DSY` irá servir esse documento, e o seu caminho será alterado para o primeiro caminho definido na árvore de navegação, de cima para baixo; se a primeira referência a `456-DSY` ocorrer sob `123-DSY`, o caminho será transformado em `/research/storm-petrels.html`.

4.3.5 Navegador de facetas

A Daisy Wiki inclui um navegador de facetas ('faceted browser'): um motor de pesquisa de documentos do repositório com base em meta-informação (as chamadas facetas) dos mesmos documentos. Uma pesquisa pode ser refinada através da seleção de uma ou mais facetas.

Para colocar em funcionamento o navegador, há que definir num ficheiro de configuração em XML quais as facetas, i. e. quais as caraterísticas dos documentos, sobre as quais se deve fazer a busca.

Como exemplo, imagine-se que se pretende que as imagens sejam visualizadas numa galeria com navegação por facetas. Uma imagem inclui já (entre outras) as propriedades 'Criador inicial', 'Data da última alteração', 'Tipo MIME', 'Coleção' e 'Língua', e meta-propriedades como se é usada em algum documento ou não.

Este tipo de documento pode ser estendido com a inclusão dos campos 'Ano', 'País' (com valores predefinidos), 'Assunto' (com valores predefinidos), e 'Tamanho' (preenchido na altura em que a imagem é guardada com um valor da lista 'Pequena/Média/Grande/Muito grande', calculado com base na área da imagem). Assim, simplesmente configurando um ficheiro XML, é possível ter uma página onde se podem pesquisar as imagens e que responde a perguntas tais como "quero imagens sobre aves, tiradas no Quénia" ou "Que imagens é que não são alteradas desde 2008?"

Outro exemplo, mais simples, encontra-se em funcionamento no site [COCO]. Aqui, as facetas são apenas as propriedades já existentes nos documentos, como o tipo de documento, a língua, o ramo e o seu último editor.

4.3.6 Outras funcionalidades da Wiki

A Daisy Wiki fornece diversas páginas de pesquisa, entre as quais: uma página onde é possível fazer qualquer consulta usando a DQL ("Query Search"); uma página de pesquisa de texto completo; e uma página que lista as mudanças mais recentes ("Recent Changes"). Todas essas páginas são consultas DQL, com uma formatação específica dos resultados.

Podemos definir múltiplos *sites*, que são diferentes vistas sobre o mesmo repositório. Cada *site* pode ter a sua própria árvore de navegação e *skin*, e está associado a uma coleção (ver secção 4.4.5, p. 39).

O conjunto de ferramentas de gestão de versões da Wiki permite-nos ver, para cada variante, uma lista das versões disponíveis, o conteúdo das versões, efetuar uma comparação entre quaisquer duas versões, reverter para uma versão anterior, e mudar o estado de uma versão de publicada para rascunho.

A Daisy Wiki pode ser mudada na sua totalidade entre o modo 'live', no qual são mostradas apenas as versões publicadas de cada variante, e o modo 'staging', no qual são mostradas as últimas versões de cada variante. Esta mudança afeta também a execução de consultas e a árvore de navegação. De uma forma geral, o modo 'staging' permite-nos ver como ficaria o *site* se fossem publicadas as últimas versões de todos os documentos.

É possível acrescentar funcionalidades à Daisy Wiki através de extensões construídas sobre Apache Cocoon. O Daisy já contém algumas extensões, por exemplo os *feeds* RSS.

A Daisy Wiki suporta internacionalização através de *bundles* de recursos. Estão já incluídas diversas línguas, como inglês, holandês, alemão, francês e russo.

A Wiki tem o conceito de 'cesto de documentos' ("document basket"). O cesto é uma forma de colecionar diversos documentos para depois efetuar alguma ação sobre eles, como seja gerar um PDF, ou executar uma tarefa de documento sobre eles.

A componente de publicação de livros permite gerar livros devidamente formatados, com índices, numeração de secções, referências cruzadas, notas de rodapé e listas de imagens.

4.4 DOCUMENTOS

Num documento Daisy, a informação relevante está contida em *partes* e em *campos*; tudo o resto é meta-informação. As partes contêm qualquer conjunto de bytes de informação (por exemplo, uma imagem, texto em XML, ou um PDF); os campos contêm informação simples, de determinado tipo (um número, uma data, uma cadeia de texto etc.).

O Daisy é um sistema não-hierárquico: não contém diretórios, ou outros elementos de níveis diferentes. Todos os documentos são guardados numa coleção uniforme, e são obtidos por meio de consultas ao servidor-repositório. O Daisy Wiki, ou outros *front-ends*, permitem definir múltiplas vistas hierárquicas sobre o mesmo conjunto de documentos.

Cada documento Daisy é definido por um identificador único, que nunca muda. Este identificador é constituído por um número e pela indicação do *namespace* ao qual pertence, p. ex. `11302-DSY` identifica o documento n.º `11302` no repositório designado por `DSY`. O *namespace* é definido durante a instalação de um repositório.

Os *namespaces* permitem manter a unicidade da chave dos documentos entre repositórios; p. ex. se tivermos um repositório de documentação com o *namespace* `DOC`, e um repositório de documentos legais com o *namespace* `LEX`, possivelmente em outro servidor, os dois repositórios poderiam partilhar documentos entre si preservando a distinção p. ex. entre `14-DOC` e `14-LEX`. O *namespace* por defeito é `DSY`, de "Daisy".

Um documento tem um proprietário, que é inicialmente o seu criador, mas que pode ser alterado posteriormente. O proprietário de um documento tem todos os direitos sobre o mesmo, independentemente das indicações da ACL (ver secção 4.2.2, p. 30).

Cada documento (e por extensão, todas as suas variantes) pertencer a um só tipo de documento. Um tipo de documento é constituído por partes e campos, de tipos predefinidos. O Daisy traz já alguns tipos de documento predefinidos, p. ex. 'SimpleDocument' (uma página simples, de uma só coluna), 'Navigation' (ver 4.3.4, p. 34) e 'Image'.

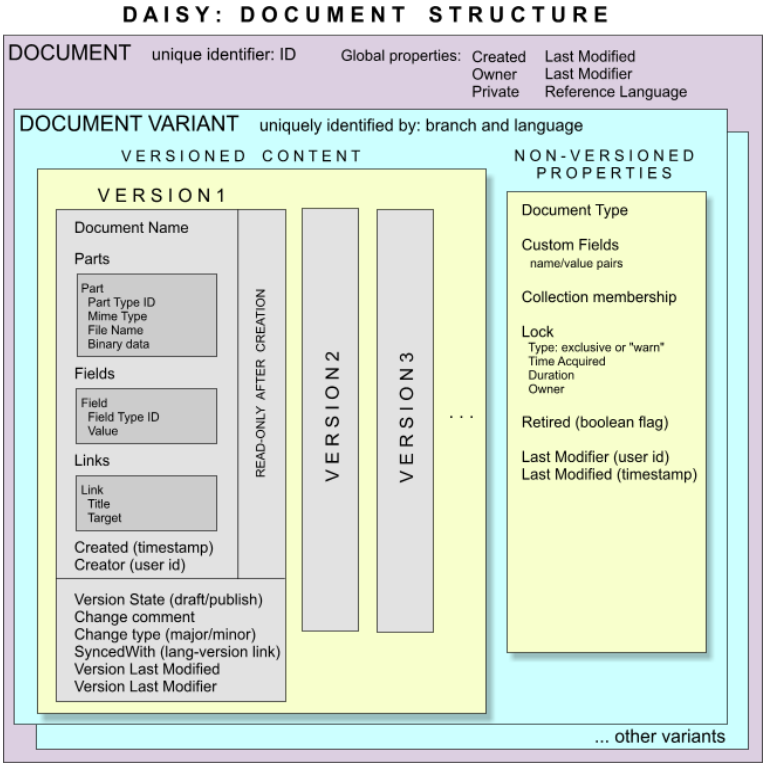

Figura 4.6: Estrutura de um documento Daisy

4.4.1 Variantes

Um documento contém algumas propriedades globais, como a coleção a que pertence, mas a informação em si está contida nas *variantes*. Cada documento tem no mínimo uma variante. Cada variante é definida por uma língua ("language") e um ramo ("branch"). A língua é usada para traduções do mesmo conteúdo; o ramo serve para publicar diferentes versões editáveis do mesmo conteúdo. O site da Outerthought usa ramos para publicar a documentação sobre as diferentes versões do CMS Daisy, em diversas línguas. A língua por defeito é "default", e o ramo por defeito é "main". O Daisy permite procurar os documentos que não tenham uma determinada variante, para detetar traduções em falta.

Um documento pode ter uma língua de referência, que serve para gestão de traduções. Imagine-se que um documento seja definido com o francês como língua de referência, e que tenha variantes em inglês, francês e árabe. Se a variante em francês for mais recente do que as variantes em inglês e árabe, então estas duas versões serão consideradas pendentes para tradução. É possível indicar que uma edição não deve provocar revisão das traduções, para permitir p. ex. corrigir erros tipográficos no francês sem que isso faça incluir as suas variantes

na lista de documentos a traduzir.

Uma variante pode ser marcada como privada. As variantes privadas apenas podem ser vistas pelo seu proprietário, e pelos administradores do sistema. Pode também ser marcada como retirada. Retirar um documento faz com que este deixe de ser visível. Esta operação pode ser desfeita; pelo contrário, a eliminação de uma variante torna-a irrecuperável.

4.4.2 Versões

As variantes de um documento contêm *versões*, que são diferentes edições da mesma. Todas as versões anteriores são guardadas no repositório; não é possível apagar uma versão.

Uma versão pode estar *ativa*, o que quer dizer que é a versão que será servida se se pedir uma variante sem se especificar o identificador da versão. A versão ativa será normalmente a mais recente, mas é possível fazer edições do conteúdo (e portanto, criar versões) sem as colocar como ativas, de forma a ir trabalhando em novas edições da variante.

4.4.3 Partes

Uma parte contém dados binários arbitrários (é portanto um BLOB). Pode guardar qualquer tipo de informação, como HTML, um documento Word, ou um MP3. Tem um tipo MIME, que descreve o tipo de dados ali guardado. Se for marcado com a *flag* "Daisy HTML?", o Daisy Wiki irá renderizar o seu conteúdo como HTML.

No Daisy, uma variante pode ter mais do que uma parte, o que permite guardar diferentes tipos de dados. É p. ex. possível ter um documento SVG que contenha, para além da parte em XML/SVG, uma descrição do mesmo em XHTML, e também a sua renderização em PNG em diferentes tamanhos. Será possível assim p. ex. obter uma lista de descrições dos SVG, com um PNG em miniatura do mesmo.

Para cada tipo de parte, pode ser definido um extrator de *links*. O Daisy inclui o extrator `daisy-html`, que extrai links no formato `daisy:<documentId>@<branch>:<language>:<version>#fragment_id`, onde o único elemento obrigatório é o ID do documento. P. ex., perante o fragmento de HTML `<a href="daisy:113@:en">bom dia</a>` o extrator de links irá convertê-lo num link para a seguinte variante: documento `113`, variante do ramo atual (pois foi omitida a designação do ramo), variante de língua inglesa (`en`), e versão que estiver marcada como `LIVE` (pois foi omitido o número de versão).

4.4.4 Campos

Os campos contêm informação simples de um certo tipo (cadeia alfanumérica, URL, data, número…) Podem ser usados para guardar dados ou metadados. Um dos tipos de dados possíveis é o *link*, que associa um documento com outro. A Daisy Wiki permite usar informação de documentos associados desta forma; vejamos o seguinte exemplo:

- Um tipo de documento "Região Vinícola" com duas partes: uma descrição XHTML da região, e um mapa da região.
- Um um tipo de documento "Vinho", que contém uma parte com a descrição XHTML do vinho, uma parte com uma imagem da garrafa, e um campo *link* que liga a um documento do tipo "Região Vinícola".

É possível, para a renderização de um documento do tipo "Vinho", incluir o mapa da região vinícola, se o campo link estiver preenchido.

Um campo pode ser de inserção livre, ou estar limitado a texto predefinido. Em ambos os casos, é possível definir uma lista de seleção para ajudar a escolher (ou para limitar a escolha) dos valores a inserir. A lista de seleção pode ser uma enumeração estática de valores, ou pode ser resultado de uma consulta à BD.

Um campo pode ser multi-valor, contendo vários valores do mesmo tipo. Pode também ser hierárquico—o que quer dizer que o seu valor é constituído por um encadeamento de valores, p. ex. "Animal → Ave → Gaivota".

4.4.5 Coleções

As coleções funcionam como categorias não-hierárquicas de informação: um documento pode pertencer a zero ou mais coleções. A principal utilidade das coleções é servir para marcar grupos de documentos; é comum usar coleções para atribuir permissões na ACL, ou restringir uma consulta a documentos de determinada coleção.

4.4.6 Comentários

O Daisy conta com um sistema de comentários bastante simples: permite a qualquer utilizador criar comentários desde que se registe no sistema. Pode também apagá-los, mas não editá-los. Para colocar um comentário não é necessário ter permissões de escrita na variante em questão. Os comentários surgem sequencialmente no final da página.

Há três tipos de comentários em Daisy:

1. *Comentários públicos:* criados e vistos por todas as pessoas que tenham permissões de leitura na variante.
2. *Comentários para editores:* só podem ser criados, e apenas são vistos, pelos utilizadores que tenham permissões de escrita na variante.
3. *Comentários privados:* apenas o criador do comentário, e os administradores do sistema, têm acesso a estes.

Quando uma variante é apagada, os seus comentários são igualmente apagados, sem que o comentador seja notificado.

4.5 DAISY QUERY LANGUAGE

O Daisy vem com a sua própria linguagem de consulta, a Daisy Query Language. Uma consulta produz como resultado uma lista de variantes de documentos; não é possível consultar outros objetos na base de dados, tais como utilizadores.

Na Daisy Wiki, as consultas podem ser usadas de três formas: feitas diretamente através da página "Query Search"; colocadas dentro de documentos; e dentro de árvores de navegação. É possível executar consultas programaticamente usando as API em HTTP ou Java. As consultas permitem pesquisar com base em diferentes propriedades das variantes (incluindo os campos), bem como no texto completo das partes. A lista de variantes resultante é filtrada para só mostrar resultados aos quais o utilizador tenha permissão de leitura.

Internamente, a consulta DQL é convertida em SQL, e a parte que incide sobre o texto dos documentos é passada para o motor de pesquisa Jakarta Lucene. A sintaxe de uma consulta DQL é superficialmente semelhante a SQL, e é constituída pelas seguintes partes:

```
SELECT <lista de campos>
WHERE <condições>
ORDER BY <ordenação>
LIMIT <número de elementos a mostrar>
OPTION <controlo sobre os resultados visualizados>
```

A DQL é insensível a maiúsculas ou minúsculas. As partes da consulta podem ser definidas resumidamente desta forma:

1. `SELECT`: indica os campos a obter. Obrigatória.
2. `WHERE`: filtra os resultados. O DQL contém uma ampla lista de operadores, funções e pseudo-funções, que testam diversas condições e devolvem informação sobre as variantes. Obrigatória; para mostrar todos os documentos, usar `WHERE TRUE`.
3. `ORDER BY`: permite ordenar os resultados da consulta, indicando os campos da ordenação, opcionalmente seguidos de DESC para indicar uma ordenação descendente.
4. `LIMIT`: limita o número de resultados mostrados.
5. `OPTION`: controla os resultados visualizados, permitindo p. ex. mostrar documentos retirados, ou formatar o resultado segundo modelos de listagem definidos na *skin*.

Como exemplo, a seguinte consulta obtém as 10 variantes mais recentes de documentos do tipo 'Image' pertencentes à coleção 'india', e devolve os resultados formatados segundo o modelo de listagem 'image-preview' definido na *skin*:

```
SELECT id, name WHERE InCollection('india') AND documentType='Image' ORDER BY
variantLastModified DESC LIMIT 10 OPTION style_hint='image-preview'
```

É possível aceder aos campos definidos numa variante, usando `$` como prefixo; p. ex. `$Subject` refere-se a um campo 'Subject' da variante.

As pesquisas sobre o motor de indexação de texto Jakarta Lucene são indicadas usando a pseudo-função `FullText()`.

Quando uma consulta é usada dentro de um documento ou de uma árvore de navegação, é possível obter valores do documento dentro do qual esta está inserida (o documento de contexto), usando a pseudo-função `ContextDoc()`. Veja-se p. ex.:

```
SELECT id, name WHERE documentType='News' AND branchID=ContextDoc(branchId)
AND languageID=ContextDoc(languageID) ORDER BY $Date DESC LIMIT 3
```

Esta consulta devolve os três documentos do tipo 'News' mais recentes (não segundo a hora de atualização do documento, mas segundo o campo `$Date` da variante), que pertençam ao ramo e língua do documento de contexto. Esta consulta inserida num documento em inglês irá produzir resultados em inglês; se estiver num documento em português, produzirá variantes em português.

4.6 UTILIZADORES

Todas as operações efetuadas no servidor-repositório são feitas por um utilizador, usando um determinado papel. Cada utilizador tem um identificador, um *login* e um nome únicos. Um utilizador pode ter um ou mais papéis ("roles"), e pode usar um de cada vez ou vários ao mesmo tempo. A autenticação de utilizadores é feita por um módulo separado, de forma a poder ser facilmente substituída por outro, se necessário. O Daisy traz já suporte para LDAP e NTLM.

O servidor-repositório tem um papel predefinido: *Administrator* (ID: 1). Os utilizadores com o papel de Administrador ativo têm uma série de privilégios adicionais:

1. Detêm todos os direitos sobre os documentos no repositório, e por isso a ACL não se lhes aplica.
2. Podem modificar o esquema do repositório (tipos de documento, partes e campos), e gerir utilizadores, coleções e a ACL.

A Daisy Wiki predefine um utilizador chamado 'guest' (visita), e um papel chamado 'guest'. Este utilizador fica automaticamente ativo quando alguém usa a aplicação Daisy Wiki, sem ser necessário fazer *login*. A ACL está preconfigurada para negar quaisquer operações de escrita a utilizadores com o papel 'guest'.

4.7 TECNOLOGIAS UTILIZADAS

A programação de um *website* em Daisy implica conhecimentos sólidos das seguintes tecnologias para a *web*:

1. CSS: uma linguagem de folhas de estilos usada para definir a semântica de apresentação de um documento. A especificação mais recente é a 2.1, definida em [BOS11].
2. HTML: uma linguagem de anotação ("markup") de texto predominante na WWW. A especificação estável mais recente é a 4.01, definida em [RAGG99]. Deriva da norma SGML, definida em [ISO8879].
3. XML: um conjunto de regras para codificar documentos de forma legível por computadores. A especificação mais recente é a 1.1 SE, definida em [BRAY96].
4. XHTML: um conjunto de linguagens de anotação ("markup") de documentos definida sobre XML, semelhante ao HTML. A especificação mais recente é a 1.1 SE, definida em [MCCA10].
5. XSLT: Uma linguagem declarativa baseada em XML para transformação de documentos XML. A especificação mais recente é a 2.0, mas o CMS Daisy usa a versão 1.0, definida em [CLAR99a], devido à plataforma Apache Cocoon suportar XSLT 1.0.
6. XPath: uma linguagem de seleção de nós XML. A especificação mais recente é a 2.0, mas o Daisy usa a versão 1.0, definida em [CLAR99b].
7. JavaScript, ou outra linguagem de execução no lado do cliente. Diversas ações predefinidas na *skin* que vem com o Daisy usam JavaScript; é também necessário para

outras funções.

8. Daisy Query Language, a linguagem de consulta de documentos própria do Daisy.
9. Apache Cocoon: o Daisy expõe a funcionalidade subjacente desta plataforma de desenvolvimento, permitindo que o Daisy seja estendido com recurso a programação mais avançada.
10. Linguagens CGI. Não sendo obrigatório (até porque quase tudo pode ser feito sobre Apache Cocoon), o autor usou CGI perl em alguns casos.

4.8 CONCLUSÃO

O Daisy é um CMS extremamente flexível e poderoso. Tem um *design* modular, pelo que qualquer componente pode ser substituída por outra. É extensível, através de Apache Cocoon. Está particularmente apto para servir grandes repositórios de informação, para a sua consulta eficaz, e para a apresentação dessa informação em qualquer forma desejada.

5. IMPLEMENTAÇÃO

Este capítulo irá expor o processo de implementação do CMS Daisy para a ARI.

5.1 RESUMO CRONOLÓGICO DA IMPLEMENTAÇÃO

A decisão de adotar o CMS Daisy foi tomada pelos diretores da ARI em agosto de 2007, um mês após a recomendação do autor. O processo de implementação do Daisy pode ser dividido em três etapas; esta secção irá descrever em traços gerais os principais eventos dessas fases, deixando as questões técnicas para secções posteriores.

5.1.1 Fase de preparação

Esta fase decorreu de julho de 2007 a maio de 2008. As principais atividades foram:

1. *Contratação de um servidor Internet.* A contratação foi feita junto de especialistas em alojamento de websites em Daisy, a empresa belga Kangaroot.
2. *Contratação dos serviços duma especialista em Daisy.* Foi contratada a Sr.ª Nik Haswati Hatta, que trabalhava no laboratório Cambia, em Canberra, e havia criado as *skins* desta entidade.
3. *Aumento dos recursos humanos da ARI.* A ARI alargou as horas de trabalho do autor, e para além da voluntária para a Internet que já tinha (Linda Sewell, no Canadá) conseguiu ainda mais uma voluntária, a Sarah Young (em Inglaterra).
4. *Estudo do CMS.* O autor passou uma grande parte do seu tempo a estudar o sistema Daisy, de forma a entendê-lo melhor.
5. *Definição da arquitetura de informação.* Como seriam estruturados os conteúdos?
6. *Definição do plano de migração do website.* Que conteúdos migrar? Quem faria o quê?
7. *Comunicação com as ARNO.* Todas as organizações A Rocha iriam utilizar o *website*, e por isso tornava-se necessário inseri-las no processo de migração.
8. *Primeiros testes.* Criação dos sites das ARNO, inserção dos primeiros conteúdos.

5.1.2 Fase de instalação

A fase de instalação correspondeu ao período em que o *website* `arocha.org` foi migrado do CMS Carrelet para o CMS Daisy. Decorreu entre junho e julho de 2008, e constou de:

1. *Criação da skin Daisy.* Foi criada principalmente pela Nik Hatta, tendo o autor se envolvido nesse processo progressivamente em maior grau.

2. *Migração dos conteúdos da ARI.*
3. *Acompanhamento da migração dos conteúdos das ARNO.* As ARNO haviam sido avisadas da necessidade de migrarem os seus conteúdos, e do prazo limite.
4. *Migração dos conteúdos das ARNO.* A ARI prestou esse serviço às ARNO que indicaram que lhes seria completamente impossível migrarem os seus próprios conteúdos.
5. *Colocação do site Daisy ao vivo.* Simultaneamente, o site em Carrelet foi colocado numa localização secundária, e ficou disponível internamente durante mais uns meses.

5.1.3 Fase de manutenção

As alterações ao Daisy não pararam após o site estar operacional, e continuam até hoje. As principais alterações têm sido:

1. *A introdução da versão em árabe*, que obrigou a adaptar o *layout* para funcionar também com linguagens com escrita da direita para a esquerda.
2. *A criação de tipos de documento* adicionais, tais como News, MultiColumn, GoogleMap e FormMailer.
3. *A criação de outros websites*, como 'Ria de Alvor' e 'Tana River Delta'.

5.2 ALOJAMENTO

Na contratação de *hardware* para alojamento, existem diversas opções à partida, que serão seguidamente analisadas, de forma conceptual e quanto à sua aplicabilidade no caso concreto.

5.2.1 Alojamento próprio vs. contratado

A primeira questão relativamente ao *hardware* prendia-se com a decisão de se a ARI iria alojar o seu próprio servidor Internet (alojamento *in-house*), ou adquirir o serviço a uma empresa de *web hosting*. Esta decisão foi debatida numa reunião com consultores de IT.

Ambas as opções de alojamento são válidas, e têm as suas caraterísticas, que se revelam melhores ou piores em função das circunstâncias específicas de cada organização:

1. *Centralização de serviços web.* Em vez de se contratarem serviços separados p. ex. para *email*, DNS, alojamento de *websites*, ferramentas de colaboração, listas de distribuição, CRM e *backup* remoto, todos estes serviços seriam corridos a partir do servidor (ou servidores) da organização. Isto poderia apresentar vantagens (discutíveis) em termos de custos, mas tornaria a gestão do servidor bastante complexa. Além disso, nem todos os serviços Internet podem ser servidos localmente: p. ex. CDN é um serviço que, pelas suas caraterísticas, tem que ser distribuído.
2. *Ligação dedicada à Internet.* Alugar e manter uma ligação de alta velocidade à Internet seria mais rentável caso o servidor estivesse num escritório, onde os funcionários ali presentes já necessitariam de uma ligação.
3. *Contratação de pessoal de TI.* Com mais serviços a serem geridos a partir da organização, seria necessário contratar profissionais para supervisionarem o *hardware* e o software, no mínimo um administrador de sistemas. A ARI é composta apenas por

uma dúzia de pessoas, pelo que seria algo desequilibrada a contratação de mais duas ou três pessoas só para tratarem de infraestrutura.

4. *Necessidade de espaço físico para os servidores.* A ARI não tinha um escritório, pelo que esta questão levantava um problema real. O aluguer de espaço físico propositadamente para os servidores (e pessoal afeto) iria tornar incomportáveis os custos desta opção. Haveria também que lidar com questões como HVAC, UPS e segurança contra roubo, inundação e incêndio.
5. *Controlo total sobre o servidor.* A ARI teria total responsabilidade e liberdade de usar o seu servidor. Isto é uma vantagem na perspetiva de inovação e experimentação de serviços, controlo do software instalado e atualizações de *hardware*; mas seria igualmente responsável pela segurança do servidor, o que é um problema se não se investir seriamente nesta área.
6. *Estrutura de custos em "picos".* Embora uma parte considerável da despesa com alojamento próprio sejam custos de recursos humanos, comprar os próprios servidores implica um pico de custos cada quatro ou cinco anos, quando se renova o *hardware*. Os picos podem não ser uma desvantagem completa, pois numa ONG é possível tentar obter fundos especificamente para IT, e neste caso as verbas vêm ligadas a um projeto com componente de *hardware*.

A opção *in-house* é geralmente mais adequada a organizações grandes com um departamento de TI, já que as atividades de desenvolvimento e suporte do *hardware* e do *software* associado exigem recursos humanos consideráveis. Um servidor próprio implicaria uma mudança muito grande na organização, principalmente por falta de instalações próprias. A ARI optou pela contratação do *web hosting*, de forma a baixar custos e ficar focada no seu *core business*.

5.2.2 Alojamento partilhado

No alojamento partilhado, o website fica alojado numa pasta de um servidor. O website tem acesso limitado aos recursos do servidor, tipicamente restringidos em termos de espaço em disco e largura de banda. Há um elevado rácio de contenção entre *websites*, já que cada servidor poderá alojar dezenas ou centenas de clientes. O cliente não tem acesso à gestão do servidor, e o software está limitado ao que tenha sido instalado pelo fornecedor. Regra geral, não é permitido executar CGI, por isso o fornecedor habitualmente instala *scripts* ou utilitários para tarefas frequentes, p. ex. formulários de contacto ou estatísticas de utilização.

Esta é a opção de alojamento mais económica, mas também a mais limitada. É a mais usual para *websites* pequenos, e todos aqueles em que apenas seja necessário servir ficheiros estáticos, sem funcionalidades dinâmicas.

5.2.3 Servidor dedicado

No alojamento dedicado ("dedicated hosting"), o fornecedor é proprietário de um servidor, que dedica exclusivamente a um cliente. A principal vantagem para o cliente é não ter que partilhar recursos da máquina com outros utilizadores, o que reduz o problema da contenção de recursos. O único recurso que é partilhado entre clientes é a largura de banda. No entanto, esta restrição é bastante menos limitativa do que no alojamento partilhado.

A largura de banda é cobrada separadamente. O método de cálculo de largura de banda varia entre fornecedores, e deverá ser alvo de atento escrutínio antes da contratação dos serviços. Duas das métricas mais comuns são:

- **Medida total**: uma contagem exaustiva da quantidade de informação enviada e recebida pelo servidor. É cobrado um valor por GB dos montantes que excedam a quota já incluída nos custos de alojamento.
- **Percentil 95**: o fornecedor anota a largura de banda consumida pelo cliente a intervalos regulares (p. ex. cada 5 minutos). Calcula posteriormente o percentil 95 da largura de banda usada, e cobra com base nesse valor, o que quer dizer que o cliente não paga pelos 5% de pico de uso.

A gestão feita pelo fornecedor poderá incluir a monitorização do servidor para assegurar o seu efetivo funcionamento, serviços de *backup*, instalação de *patches* de segurança, e diversos níveis de suporte técnico. Esta gestão recorre a processos e ferramentas sistemáticas, pelo que existem poupanças de escala consideráveis face à gestão de um só servidor.

Um servidor dedicado pode permitir ao cliente um acesso total ao servidor ("full root"), ou pelo menos bastante alargado, permitindo a instalação de aplicações completas.

5.2.4 Servidor privado virtual

Num servidor privado virtual (VPS), diversos servidores lógicos correm num mesmo servidor de forma conceptualmente independente, com recurso a tecnologias de virtualização. A vantagem principal é mostrar ao cliente uma "máquina completa", mas a uma fração do custo total da máquina, já que a mesma se encontra dividida entre vários clientes. Existe algum *overhead* face a um servidor dedicado: tempo de CPU, espaço em disco e RAM consumidos pela camada de virtualização.

Um VPS é semelhante à opção de servidor dedicado, já que para todos os fins de gestão existe uma máquina dedicada ao cliente. A RAM e espaço em disco estão divididas entre os clientes, pelo que são um pouco mais limitadas. O tempo de CPU é dividido entre os clientes, não de forma equitativa mas segundo as necessidades do momento. Assim, as especificações de VPS não indicam o CPU mas "Max CPU"—o máximo de potência disponível para o CPU. As especificações indicam ainda o rácio de contenção ("contention ratio"), que é o número de clientes por máquina.

5.2.5 Co-localização

O serviço de co-localização ("co-location") compete diretamente com o alojamento em servidor próprio nas instalações do cliente. É uma das opções mais caras. Aqui, o servidor é adquirido e totalmente gerido pelo cliente, mas encontra-se localizado em parques de servidores pertencentes a uma empresa que fornece diversas ligações de alta velocidade à Internet, segurança das instalações, UPS, HVAC e monitorização 24/7 do servidor. Os recursos estão limitados pelas *specs* do servidor, que o cliente compra e manda entregar no *server park*.

As vantagens da co-localização relativamente à colocação do servidor em instalações próprias são as seguintes:

- *Segurança do perímetro.* O acesso a um *server park* é extremamente limitado—nem os

próprios clientes podem ir "visitar a máquina".

- *Ligações redundantes à Internet.* Se uma ligação falhar, existem sempre outras.
- *Monitorização constante do servidor,* com pessoal de serviço 24 horas por dia, para além de aplicações de monitorização que verificam constantemente o estado das máquinas.

5.2.6 Cloud hosting

Cloud hosting (também denominada *on-demand hosting*) é um subsetor de *cloud computing*, ou "computação na nuvem". Uma empresa de fornecimento de serviços IT detém uma enorme capacidade de computação, constituída por milhares de servidores localizados na Internet; conceptualmente, esta capacidade de computação é representada por uma nuvem. O cliente contrata serviços de computação, e paga apenas os recursos efetivamente usados, como o tempo de CPU utilizado, a largura de banda consumida ou o espaço em disco utilizado. A capacidade contratada pode ser aumentada ou diminuída à medida das necessidades de computação em *near real time*, ou seja, sendo atendida quase de imediato. A seguinte figura (segundo [JOHN08]) ilustra uma arquitetura *cloud* típica, contendo armazenamento, um gestor de virtualização (no exemplo, IVM), uma plataforma e um serviço de *software*.

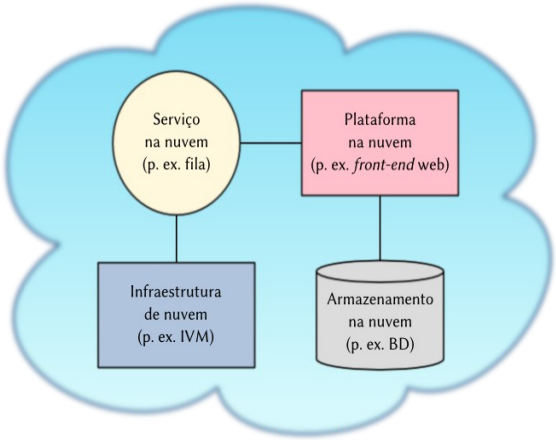

Figura 5.1: Exemplo de arquitetura cloud

O *cloud hosting* traça paralelos de "consumo da rede" com as *utilities*, pois tal como no caso da rede elétrica não sabemos de onde vem a eletricidade que consumimos, e a potência disponível é aparentemente infinita, também no *cloud hosting* não é relevante a localização física dos servidores que nos fornecem o serviço.

A vantagem principal do *cloud hosting* ocorre nos casos em que o acesso aos serviços apresentem picos muito marcados. Imagine-se um website que sirva em média 3 600 páginas por hora; o servidor pode estar sobredimensionado para acomodar um máximo de 15 000 páginas por hora. Assim, pagam-se recursos que normalmente não são usados; por outro lado, se numa hora houver 20 000 pedidos, o servidor poderá não conseguir atendê-los a todos. No *cloud hosting*, toda a nuvem está grandemente sobredimensionada para o cliente, de forma a que os seus recursos parecem infinitos: os picos de uso estão devidamente cobertos.

5.2.7 Conclusão

Para a escolha do tipo de serviço de alojamento, a primeira decisão feita foi de separar o

serviço *web* das outras necessidades na Internet. Em particular, tendo em conta as necessidades de fiabilidade do serviço de *email*, e a falta de pessoal qualificado para gerir um serviço de *email*, foi decidido que este seria contratado a uma empresa especialista em *email hosting*. A aquisição de um servidor próprio tornou-se assim menos atrativa.

No entanto, considerou-se útil que a solução de *hosting* permitisse total liberdade de execução de código, e de instalação de aplicações *web*, de forma a servir não só para o *website* mas também como *sandbox* para testar soluções *online*, pelo que comparadas as caraterísticas de cada um dos tipos de alojamento externo (ver tabela abaixo), e as necessidades da ARI presentes e futuras, optou-se por contratar um VPS.

Tabela 5.1: Comparação entre tipos de alojamento

Item	**Partilhado**	**Dedicado**	**VPS**	**Co-location**	**Cloud hosting**
Gestão do servidor	fornecedor	cliente	partilhada	cliente	fornecedor
Propriedade do servidor	fornecedor	fornecedor	fornecedor	cliente	fornecedor
Obsolescência do *hardware*	não	depende do contrato	sim	sim	não
Limitações de *hardware*	limitada por contrato	specs da máquina	specs do VPS	specs da máquina	na prática, ilimitado
Liberdade de instalação de *software*	muito restrita	total	total	total	alguma
Contenção entre clientes	alta	nula[2]	baixa	nula[2]	nula[2]
Custos do serviço	baixos	médios	médios	altos	médios

O VPS contratado à Kangaroot tinha as seguintes caraterísticas técnicas: Plataforma de virtualização: Xen Enterprise hypervisor; Infraestrutura de rede local Cisco / Allied Telesis; Múltiplas conexões externas, com capacidade da ordem do Gigabit; Debian GNU/Linux 4.0 "Etch"; Acesso "full root"; VM armazenada na SAN central, para fácil migração; 10 GB em disco; 256 MB RAM; Max CPU: Dual Xeon 3.0; Largura de banda: 50 GB; Rácio de contenção: 10:1.

2 A contenção entre clientes é considerada "nula" nestes casos porque cada cliente tem a sua própria máquina (servidores dedicados, co-locação), ou os serviços disponíveis são extremamente abundantes (*cloud hosting*). Sempre existe alguma contenção; p. ex. na co-locação os clientes competem entre si pela largura de banda tanto LAN como WAN, embora estas sejam normalmente sobredimensionadas de forma a permitirem um serviço confortável a todos os clientes.

A RAM cedo se revelou insuficiente para suportar a JVM, pelo que foi feito um *upgrade* para os atuais 768 MB. A quota de disco rígido foi entretanto aumentada para 20 GB.

5.3 SISTEMA OPERATIVO

Como já foi referido, o servidor Internet da ARI corre o OS Debian GNU/Linux Server 4.0 "Etch". Debian é uma das distribuições de Linux mais usadas, seja sob o seu próprio nome, seja como base de outros OS como o Ubuntu. É composta exclusivamente por *packages* de *software* livre, e dá elevada prioridade à segurança. O Debian é usado numa grande variedade de *hardware*, de telefones a servidores.

O servidor da ARI contém assim um OS completo, com todas as potencialidades e complexidades do mesmo. Não contém nenhum painel de controlo; todas as operações de manutenção são feitas via `ssh`. O servidor já serviu entretanto para experimentar algum outro software, p. ex. Mantis BT, SugarFree CRM, CiviCRM e WordPress.

5.4 ARQUITETURA DA INFORMAÇÃO

Esta secção vai debruçar-se sobre as questões relativas à estruturação dos conteúdos na sua forma geral, à configuração do servidor e ao sistema de controlo de permissões de acesso.

5.4.1 Servidores web Jetty e Apache

Logo que o Daisy é instalado está já a servir páginas: é possível aceder a `arocha.org:8888` e ver páginas servidas pelo servidor incorporado, Jetty. Quando se pretende maior funcionalidade do que a que vem configurada no Jetty, é usual configurar-se um servidor Apache, que passa por defeito todos os pedidos para o Jetty, e depois programar as exceções para serem tratadas pelo Apache. O Daisy traz já um ficheiro de configuração para Apache para este fim, onde a definição de uma variável de ambiente `no-jk` impede o Jetty de tratar o *request*. Os exemplos dados de seguida cobrem os tipos de casos de exceções Apache nos *websites* da ARI:

1. A redireção de `arocha.org` para `www.arocha.org`, necessária do ponto de vista da usabilidade, é conseguida da seguinte forma:

```
<VirtualHost 62.213.193.100:80>
  ServerName arocha.org
  ServerAlias
    christiansinconservation.net www.christiansinconservation.net
    christiansinconservation.com www.christiansinconservation.com
    christiansinconservation.org www.christiansinconservation.org
  RedirectMatch permanent /(.*) http://www.arocha.org/$1
</VirtualHost>
```

2. Servir estaticamente o conteúdo de um diretório `static`, de `robots.txt` e dos ficheiros `index.html.*` para a negociação automática de conteúdos do Apache:

```
SetEnvIfNoCase Request_URI ^/robots.txt$ no-jk
SetEnvIfNoCase Request_URI ^/$ no-jk
SetEnvIfNoCase Request_URI ^/index.html$ no-jk
```

```
SetEnvIfNoCase Request_URI ^/static/ no-jk
RewriteCond %{REQUEST_URI} !^/$
RewriteCond %{REQUEST_URI} !^/index.html$
RewriteCond %{REQUEST_URI} !^/robots.txt$
RewriteCond %{REQUEST_URI} !^/static/
<Directory /home/arocha/www/>
  Options Indexes FollowSymLinks MultiViews
  AllowOverride None
  Order allow,deny
  allow from all
</Directory>
```

3. Atalhos para URL, para uso sempre que seja necessário escrevê-los diretamente no *browser* (p. ex. para publicitar numa revista):

```
Redirect permanent /donate http://www.arocha.org/int-en/11102-DSY.html
SetEnvIfNoCase Request_URI ^/donate no-jk
RewriteCond %{REQUEST_URI} !^/donate
```

4. Finalmente, para servir CGI utiliza-se a seguinte configuração:

```
SetEnvIfNoCase Request_URI ^/cgi-bin/.*$ no-jk
RewriteCond %{REQUEST_URI} !^/cgi-bin/
ScriptAlias /cgi-bin/ /home/arocha/cgi-bin/
<Directory /home/arocha/cgi-bin/>
  AllowOverride None
  Options None
  Order allow,deny
  Allow from all
</Directory>
```

5.4.2 Estrutura dos *sites* Wiki

O servidor-repositório do Daisy é estritamente não-hierárquico: todos os documentos são iguais, e são pesquisados pelas suas caraterísticas, ou facetas. O *front-end* Daisy Wiki define estruturas sobre os documentos do repositório, com recurso a coleções e a documentos de navegação.

Para o Daisy Wiki, um *website* é constituído pelas variantes de um ramo e uma língua pertencentes a uma coleção; o website é definido num ficheiro XML que indica a coleção, ramo e língua, bem como do nome do diretório que vai ser montado, já que o Daisy não define páginas para serem corridas diretamente a partir da raiz de um domínio.

Esta definição não encaixa na definição comum de *website*. Assim, p. ex. A Rocha Portugal tem dois *websites*: um com páginas em inglês, e outro com páginas em português. Logo que se fez clara esta distinção, partiu-se para a definição de uma estrutura de nomes que cobrisse necessidades presentes e futuras. Optou-se seguir o padrão `<país>-<língua>`; p. ex., as páginas de Portugal em inglês estão em `/pt-en`, e as páginas de Portugal em português estão em `/pt-pt`, enquanto que `/int-pt` contém as páginas internacionais em português.

No CMS Carrelet, que não geria conteúdo multilingue, os conteúdos em diferentes línguas estavam em subdomínios separados, p. ex. `pt.arocha.org` para as páginas em

português e `en.arocha.org` para as páginas em inglês. `www.arocha.org` continha apenas a redireção para os subdomínios de línguas. Na transição para o novo CMS, todas as línguas foram colocadas sob `www.arocha.org`, o que obrigou ao uso de *redirects* Apache para mapear links antigos em novos, p. ex. `en.arocha.org/canada` em `www.arocha.org/ca-en/`.

5.4.3 Workflow de publicação

O Daisy possui uma extensão Workflow, programada em jBPM. O jBPM é uma ferramenta em Java, e baseia-se num mecanismo genérico de descrição de processos, para suportar diversas linguagens de processo. A principal dessas linguagens é a BPMN, uma aplicação de XML extensível que define a visualização e serialização de processos.

A ARI não usou o Workflow. Este método de controlar o processo de produção e publicação é naturalmente mais indicado para organizações bastante hierárquicas, ou com um número de funcionários que justifique o investimento numa definição de processos rigorosa.

5.4.4 Roles e ACL

O Daisy permite a definição de modos de utilização, em inglês *roles*. Cada utilizador pode ter mais do que um *role*; o *role* ativo controla as operações que o utilizador pode efetuar sobre cada objeto no repositório. O seguinte excerto mostra os principais *roles* que o autor definiu:

Object	Subject		Permissions			
	type	value	Read	Write	Delete	Publish
If true						
Then						
	everyone	(-1)	✓			
	role	guest (101)	✓			
	role	AR Global Editor (2)		✓	✓	✓
...						
If InCollection('teams')						
Then						
	role	guest (101)	✗			
	role	Teams Reader (103)	✓			
	role	Teams Editor (104)	✓	✓	✓	✓
...						
If InCollection('riadealvor')						
Then						
	role	Ria de Alvor Editor (109)		✓	✓	✓
...						
If InCollection('ghana')						
Then						
	role	AR Ghana Editor (114)		✓	✓	✓
...						
If language = 'zh-Hans' or language = 'zh-Hant'						
Then						
	role	AR Language Editor: Chinese (130)		✓		✓
...						

Figura 5.2: Resumo das regras na ACL

1. *Editores por país.* Estes estão restritos a editar conteúdos pertencentes a determinada coleção. Há assim *roles* "Brazil Editor", "Ghana Editor", "UK editor" etc.
2. *Editores por língua.* Estes *roles* permitem editar qualquer página numa determinada língua, independentemente da coleção em que esteja, e são atribuídos aos tradutores. Existe p. ex. um *role* "Language Editor: Chinese", que permite criar páginas em ambos os sistemas ideográficos chineses (tradicional e simplificado).
3. *Editores por website.* A cada um dos outros websites criados posteriormente (p. ex.

`tanariverdelta.org` e `riadealvor.org`) corresponde um *role* de editor.

4. Para o caso especial da *intranet* (documentos da coleção 'teams'), foram definidos dois papéis: um exclusivamente para leitura ("Teams Reader") e outro que permite edição de conteúdos ("Teams Editor").

Todos os *roles* criados fornecem as mesmas permissões de *write*, *publish* e *delete*; não foi criado nenhum papel de supervisão. No entanto, será de considerar criar um papel com as permissões exclusivas de *publish* e *delete*, retirando essas permissões aos editores, assim que seja possível em termos de recursos humanos ter uma pessoa responsável exclusivamente pela edição dos *websites*. De momento, a Editora da ARI (Barbara Mearns) é responsável não apenas pelos conteúdos online, mas pelos *emails* informativos, revista impressa e relatório anual, e portanto não tem a capacidade de rever todos os conteúdos que são criados.

5.5 MIGRAÇÃO DO WEBSITE

5.5.1 Migração dos conteúdos

Dado o grande volume de informação existente e a falta de recursos humanos, a primeira questão que se coloca: que conteúdos migrar? Todas as páginas de um *website* são importantes, mesmo as muito antigas. [NIEL98] indica dois grupos de razões para manter o conteúdo antigo:

1. *O conteúdo antigo continua a trazer visitas.* Uma página sobre a investigação em morcegos em Portugal em 2003 irá sempre ser encontrada por alguém que procurou por termos semelhantes num motor de pesquisa.
2. *Evitar o link rot.* O *link rot* ("podridão das ligações") ocorre sempre que alguém tenta usar um URL que deixou de funcionar. Os *links* podem estar em muitos sítios: em material impresso, nos *bookmarks* dos *web browsers*, nas BD dos motores de pesquisa— e até memorizados na cabeça dos visitantes. Se se remover uma página, o *link* vai deixar de funcionar.

Dada a dimensão da tarefa de migração, foram feitas as seguintes opções estratégicas:

1. *A ARI criaria os documentos de navegação para as ARNO.* A navegação funcionava de forma totalmente diferente no novo CMS, e por isso criar a navegação baixaria essa barreira de entrada no Daisy.
2. *As ARNO deveriam definir quais os conteúdos a mover.* Cabia a cada ARNO indicar o que deveria ser levado para o novo *website*, dentro das suas páginas.
3. *A ARI moveria os conteúdos das ARNO que o solicitassem.* Nem todas as ARNO tinham as mesmas capacidades técnicas, ou davam o mesmo uso ao *website*; e assim a ARI iria responsabilizar-se por mover ou ajudar a mover os conteúdos, conforme necessário.
4. *As ARNO deveriam mover os seus próprios conteúdos.* Foi definida uma data limite para a migração, dadas instruções via *email*, e construído um micro-*site* dentro da *intranet* para apresentação das principais caraterísticas do Daisy e ensinar a trabalhar com este CMS [REIS08a]. No entanto, sempre que possível seriam as ARNO a mover os seus conteúdos, por um lado porque os recursos da ARI eram escassos, e por outro porque

mover os seus conteúdos lhes iria dar a prática necessária para continuarem a atualizar as suas páginas.

5. *A ARI não iria mover todos os seus conteúdos.* Em particular, as notícias existentes no *website* foram deixadas para trás. Foram quatro anos de notícias que se perderam, mas foi uma opção necessária, até porque diversas ARNO solicitaram a ajuda da ARI para a migração dos seus conteúdos.
6. *A migração deveria estar concluída antes de agosto.* Agosto era um mês particularmente complexo em termos de fluxo de trabalho para a ARI e para as ARNO, já que a maioria delas se encontram no hemisfério norte, no qual agosto é tipicamente um mês de férias, e que por isso os recursos humanos se encontram mais escassos.

5.5.2 Migração dos utilizadores

O CMS anterior, Carrelet, não tinha contas de utilizadores. Como tal, não foi necessário migrar informação relativa aos mesmos. Foi solicitado às pessoas que já editavam o *website* em Carrelet que criassem a sua conta de utilizador no Daisy. Em alguns casos, o autor criou as próprias contas dos utilizadores, de forma a remover essa potencial barreira de entrada.

5.5.3 Migração do design

Figura 5.3: arocha.org, setembro de 2011

Foi decidido não fazer alterações radicais ao *design* do *website*, de forma a agilizar o processo

de migração. Assim, o autor e a Sr.ª Hatta criaram uma *skin* para Daisy, que ainda hoje se encontra em uso, o mais semelhante possível ao *design* usado sob o Carrelet, incluindo apenas as alterações que se tornassem obrigatórias devido às diferenças de funcionamento dos dois CMS. Esta *skin* foi posteriormente refinada, mas nos seus traços gerais o *design* hoje existente em `arocha.org` é o mesmo que existia em 2007, e o esquema de cores é idêntico ao que existia em 2003.

A ARI instalou a versão mais recente do Daisy, a 2.1. A Sr.ª Hatta estava habituada à versão 1.6, usada nos sites da organização Cambia, de forma que o processo de criação da *skin* teve algumas questões iniciais de compreensão do que havia mudado no CMS. A Sr.ª Hatta não participou na migração dos conteúdos, e os seus serviços terminaram em finais de julho de 2008, e nessa altura o autor assumiu total responsabilidade sobre o Daisy.

5.6 CUSTOMIZAÇÃO DA SKIN

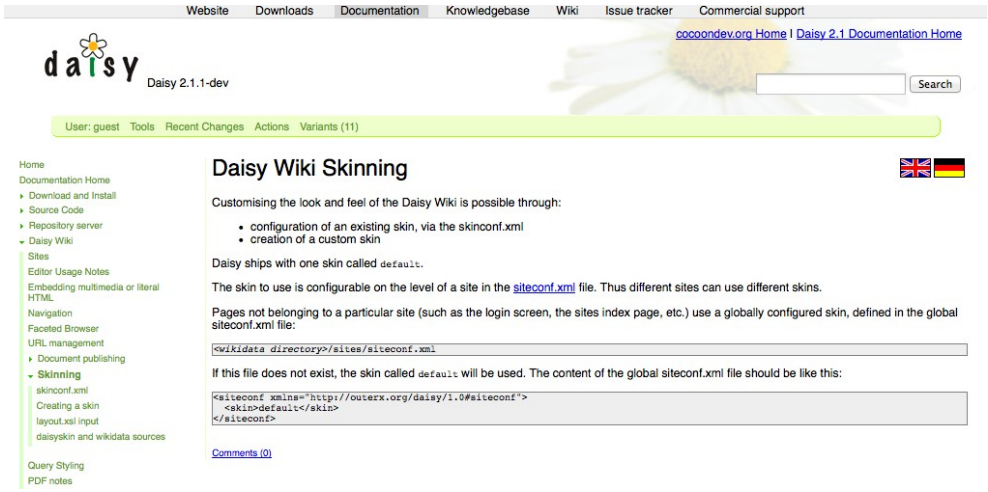

Figura 5.4: Skin 'default'

A *skin* usa o mecanismo de herança de *skins* do Daisy, e é baseada na *skin* `default`. As principais alterações feitas à *skin* foram as seguintes:

1. Remoção dos *links* para as *homepages* do domínio e do *site*. Estes *links* não faziam sentido no website `arocha.org`.
2. *Criação de um menu de nível superior*, só com elementos de primeiro nível.
3. *Transformação do menu lateral num submenu.* Este mostra apenas os descendentes do item de nível superior em que se está; p. ex. se estivermos na página *Our work* → *Conservation* → *Species profiles* → *Cork Oak*, o submenu irá mostrar todos itens sob *Our work*. O submenu só surge quando houver páginas para mostrar.
4. *Lista de traduções da página.* A *skin* mostra uma lista de traduções da página atual, sempre que as mesmas existam.
5. *Remoção dos comentários.* Os comentários no fundo da página deixaram de ser mostrados para utilizadores com *role* 'guest'. Esta decisão deriva de não haver recursos humanos para moderar esses comentários, e também da falta de flexibilidade do sistema de comentários presente no Daisy.

6. *Itálico no nome da página.* Em Daisy, o nome dado à página serve como o seu título, sendo inserido num tag `<h1>` no topo da página. Não é possível inserir código HTML no título de uma página; era no entanto necessário formatar os nomes científicos em itálico. Para resolver este problema, o autor deu um valor "mágico" aos parênteses retos, e com base na manipulação do texto contido no título, e transformou-os via XSLT em abertura e fecho de *tag* `<i>`. Assim, o título da página inserido como 'Sobreiro [Quercus suber]' fica renderizado `Sobreiro <i>Quercus suber</i>`.
7. *Tratamento de imagens.* Na *skin* `default`, as imagens são mostradas no seu tamanho real apenas se forem mais pequenas que uma determinada dimensão; se forem maiores, surge uma miniatura com um link "click here to enlarge". Funciona bem em *sites* constituídos por *screenshots* como o do próprio Daisy, mas não em `arocha.org`. A imagem é agora mostrada no tamanho que se especificar. Se a imagem estiver inserida num *tag* `<a>`, a imagem liga a esse URL; se não, ela faz de *link* para a versão de tamanho maior da imagem. O autor reutilizou o mecanismo usado para o título das páginas de forma a permitir que a legenda da imagem contivesse texto em itálico.
8. *Alterações ao código sem impacto gráfico.* Estas outras alterações serão detalhadas na secção 5.6.5, p. 59.

Em toda a customização da *skin*, usámos sempre o *branch* por defeito ("main"); não houve necessidade de usar *branches*, apenas as variantes de língua para tradução.

Nota relativa ao ponto 6: A alteração à *skin* foi feita em XSLT, que é uma linguagem direcionada para a manipulação declarativa de documentos XML, portanto pouco vocacionada para a manipulação de *strings*. Se o número de funções a criar fosse superior, seria preferível criar um *jar* Java com as funções necessárias, e importá-lo para o XSLT através do mecanismo de *namespaces* do processador Xalan-Java. Como exemplo, uma declaração

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:my="xalan://org.arocha.daisy.utils">
```

—iria associar o *jar* `org.arocha.daisy.utils` ao prefixo `my`. Se esse *jar* contivesse a função `bracesToItalics`, para converter uma string em itálico no XSLT bastaria usar:

```
<xml:copy-of select="my:bracesToItalics(…)"/>
```

5.6.1 Localização

À medida que se ia definindo a *skin*, o autor chegava à conclusão que seria necessário localizar diversos elementos de texto que surgiam na mesma. O Daisy dispunha já de um mecanismo de localização, usado na interface de edição e administração, mas o autor achou melhor criar o seu próprio mecanismo de localização do que mexer no do CMS e ter que fazer *merge* das *diffs* sempre que houvesse um *upgrade* ao Daisy.

Um ficheiro em texto simples não satisfazia as necessidades atuais de tradução. Em alguns casos, os fragmentos traduzidos eram constituídos por várias linhas de texto; e de futuro poderia ser necessário inserir *tags* HTML nos fragmentos. Assim, o autor criou um documento em XML na *skin*: o `locale.xml` deveria conter os termos localizados em todas as línguas necessárias. As *strings* seriam acedidas através de uma função utilitária, o que

provocou a necessidade de criar um novo documento XSLT, `myutils.xsl`, já que esta função (e outras, criadas posteriormente) deveriam estar presentes nos diferentes ficheiros XSLT.

O `locale.xml` segue o seguinte esquema (aqui indicado em Haml):

```
customLocales
    todo
        item
    notes
        note
    locales
        language
            strings
                string
```

—e este é o código XSLT que vai buscar o valor localizado, com controlo de erros:

```
<xsl:template name="localise">
  <xsl:param name="language" select="$siteLang"/>
  <xsl:param name="string"/>
  <xsl:variable name="stringnode" select="document('locale.xml')/customLocales
    /locales/language[@id=$language]/strings/string[@id=$string]"/>
  <xsl:choose>
    <xsl:when test="count($stringnode) > 0">
      <xsl:value-of select="$stringnode"/> <!--node exists-->
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="concat('[nostring:',$language,':',$string,']')"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

5.6.2 Formatação de listas

Figura 5.5: Consulta de notícias com 'style_hint'

O Daisy Wiki tem uma folha XSLT (`query-styling-html.xsl`) para a transformação de

consultas segundo uma opção `style_hint` indicada na cláusula DQL. Esta funcionalidade foi usada para produzir listas de notícias e eventos no *website*. Veja-se a seguinte consulta presente na página principal em todas as línguas:

```
SELECT name, language, Year($Date), Month($Date), DayOfMonth($Date),
%FeatureThumbnail.size, %SimpleDocumentContent.content
WHERE documentType = 'News' AND $Date <= CurrentDate() AND
InCollection('arocha')
AND branchID = ContextDoc(branchID) AND languageID = ContextDoc(languageID)
AND $NotOnMainPage = 'false' AND $NewsSubType = 'by-us'
ORDER BY $Date DESC
LIMIT 3 OPTION style_hint = 'news-yes-pic'
```

O código XSLT que produz a listagem é o seguinte:

```
<xsl:template match="d:searchResult[@styleHint='news']">
  <div class="news-box" style="border-top: 1px dotted gray">
    <xsl:for-each select="d:rows/d:row">
      <xsl:variable name="newsContent">
        <xsl:value-of select="substring(d:xmlValue, 1, 200)" />
      </xsl:variable>
      <xsl:variable name="date">
        <xsl:call-template name="localDate">
          <xsl:with-param name="format" select="'short'"/>
          <xsl:with-param name="language" select="d:value[2]"/>
          <xsl:with-param name="year" select="d:value[3]"/>
          <xsl:with-param name="month" select="d:value[4]"/>
          <xsl:with-param name="day" select="d:value[5]"/>
        </xsl:call-template>
      </xsl:variable>
      <div class="news-item" style="border-bottom: 1px dotted gray">
        <p>
          <strong><xsl:value-of select="d:value[1]"/></strong>
          <xsl:text> </xsl:text>
          <em style="color:#666"><xsl:value-of select="$date"/></em>
        </p>
        <p>
          <xsl:value-of select="concat($newsContent,'...')" />
          <xsl:text> (</xsl:text>
          <a href="{$searchResultBasePath}{@documentId}.html?
            branch={@branchId}&amp;language={@languageId}">
            <xsl:call-template name="localise">
              <xsl:with-param name="string" select="'read-more'"/>
              <xsl:with-param name="language" select="d:value[2]"/>
            </xsl:call-template>
          </a>
          <xsl:text>)</xsl:text>
        </p>
      </div>
    </xsl:for-each>
  </div>
</xsl:template>
```

5.6.3 Google Analytics

As métricas de utilização são essenciais para se ter um feedback do trabalho feito. Que páginas são mais visitadas? Quais os trajetos que os visitantes fazem dentro do website? De onde vêm quando cá chegam?

Os *websites* d'A Rocha usam o Google Analytics para recolher as métricas de utilização. Assim, foi necessário introduzir código JavaScript genérico em todas as páginas. O *beacon* só é enviado para o Google Analytics em páginas regulares, e não em páginas de administração ou de edição.

5.6.4 Adaptação da skin para árabe

As línguas escritas da direita para a esquerda têm uma lógica de apresentação diferente, pois a noção estética que temos quando analisamos não é 'à esquerda' ou 'à direita' em termos absolutos, mas sim 'na origem da escrita' ou 'no final da escrita' [TEXI05]. Assim, para que o *design* tenha o mesmo efeito, deverá ser espelhado.

O autor usou o seguinte método para mudar a localização dos elementos gráficos sem ter que recorrer a mais do que uma folha de estilos: indicar a direção de escrita da língua numa *tag* de nível superior no HTML. Note-se o efeito da posição espelhada dos elementos no *screenshot* abaixo: o logotipo à direita, submenu à direita, caixa de pesquisa à esquerda dão à página um *flow* em tudo idêntico ao *design* para línguas da esquerda para a direita.

Figura 5.6: Resultado da skin numa página em árabe

Foi em primeiro lugar necessário incluir informação sobre a direção de texto no documento de localização. Assim o valor de `document('locale.xml')/customLocales/locales/language[@id='ar']/strings/string[@id='text-direction']` é 'rtl' (é 'ltr' para todas as outras línguas presentemente utilizadas). Este valor é colocado no atributo `dir` da *tag* `<body>`. Depois bastou usar o seletor de atributos na folha de estilos CSS para posicionar os elementos segundo a direção do texto, p. ex.:

```
body[dir="ltr"] #menu { float:left; text-align: left; }
body[dir="rtl"] #menu { float:right; text-align: right; }
```

5.6.5 Alterações ao código sem impacto gráfico

Foram efetuadas algumas alterações ao código da *skin* que não tiveram repercussão no *layout* das páginas. Uma delas foi acrescentar à tag `<body>` (que anteriormente se apresentava sem atributos), para além do atributo `dir`, já explicado, também `id` e `class`; p. ex. para a página das Notícias em inglês, temos agora: `<body id="doc2183-DSY" dir="ltr" class="branch-4">`. Ambas as opções foram colocadas para flexibilizar o *skinning* em alterações futuras: a `class` permitiria mudar o design com base na área do *website* em que se está (mais sobre isto na secção 5.8.3, p. 70); e a definição de um `id` por página permitiria inclusivamente criar um design diferente para cada página. Esta última ideia foi inspirada pelo *web designer* Jason Santa Maria, com as suas experiências de *art direction online* (ver [JSM10], onde cada página tem um *design* que ajuda a contar a história do texto, como as páginas de uma revista).

A outra alteração ao código foi a consolidação de ficheiros JavaScript e CSS, de forma a reduzir o número de pedidos ao servidor e agilizar o carregamento da página. Foi igualmente feita a remoção de código morto dos mesmos ficheiros.

5.6.6 Elementos de design não utilizados

O autor finaliza esta secção com uma referência a elementos de design que criou, mas que não estão presentemente a ser utilizados no *website*.

1. *Navegação estrutural.* Um elemento importante para situar o visitante na estrutura geral de *websites* hierárquicos com uma navegação profunda. Um exemplo para uma página existente seria: *A Rocha* → *Our work* → *Conservation* → *Species profiles* → *Cork Oak*. Normalmente é colocado perto do título da página. Este elemento de navegação é criado em Daisy através da pesquisa na árvore de navegação do documento, para encontrar o elemento que tem o atributo `selected` definido, depois mostrando esse elemento e todos os seus antecessores.
2. *Botão 'Imprimir'.* Este elemento mostrava a página atual com um *layout* simplificado: sem menus ou rodapés, apenas o conteúdo principal. Esta função fazia uso das capacidades do Daisy para produção de diferentes resultados com base no mesmo conteúdo, com pequenas adaptações na *skin*.
3. *Botão 'Enviar por email'.* Este elemento despoletava a criação de um *email* na aplicação de *email* da pessoa, com o assunto e corpo já preenchidos, para o visitante poder sugerir a página atual a outra pessoa.

5.7 TIPOS DE DOCUMENTO

Os tipos de documento são um dos mecanismos mais potentes no Daisy, pela flexibilidade que trazem ao *layout*. Esta secção irá referir os tipos de documento criados de raiz ou customizados para o *website* `arocha.org`, indicando o seu objetivo, arquitetura e funcionalidade. A lista encontra-se ordenada por data de criação do *doctype*, e não inclui elementos triviais como o ID do documento e o nome da variante.

5.7.1 Image

O *doctype* 'Image' permite guardar qualquer imagem em formato *raster*. A implementação de

origem contém dois campos apenas de leitura, 'width' e 'height', onde são guardadas as dimensões da imagem. Estes campos são usados para preencher os atributos `width` e `height` na renderização da *tag* `<img>`. Guarda também duas versões da imagem em tamanho reduzido (125 e 250 *pixels* na maior dimensão), que são usadas para mostrar a imagem quando o tamanho desejado for 250 *pixels* ou menos.

Aquando da adoção do Daisy, o autor pensou em adicionar meta-informação aos *doctype* 'Image', de forma a se usar este CMS para organizar a biblioteca de *media* d'A Rocha. Seria uma boa aplicação das potencialidades de *faceted browsing*. Foram assim criados os seguintes campos: 'Descrição', 'Ano de criação', 'Assunto', 'Local', 'Autor' e 'Alteração de ordenação'. Os campos 'Assunto', 'Local' e 'Autor' estão limitados a valores predefinidos, de forma a uniformizar a escrita. O campo 'Alteração de ordenação' permite identificar imagens como "favoritas" e colocá-las no topo de resultados de pesquisa.

Estes campos foram também acrescentados aos *doctypes* 'SimpleDocument', 'MultiColumn' e 'Attachment', mas nunca foram usados. Um vetor de desenvolvimento futuro para o website seria implementar internamente a pesquisa facetada, de forma a facilitar a reutilização de conteúdos.

5.7.2 LiteralHtml

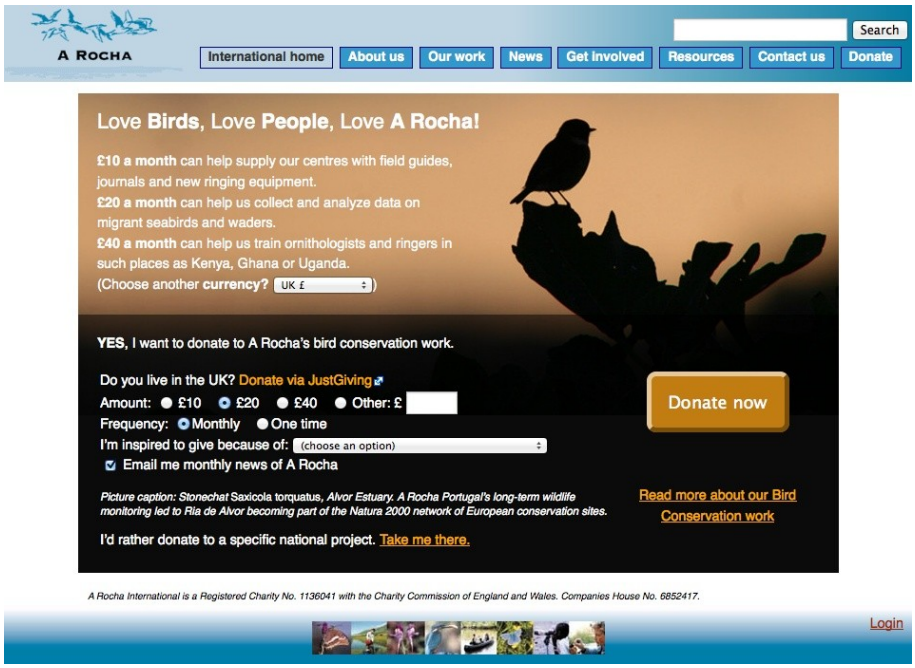

Figura 5.7: "Love Birds" – um exemplo de LiteralHtml

O LiteralHtml é o *doctype* mais flexível no seu uso, já que permite introduzir qualquer combinação de HTML. O autor cria uma página destas sempre que é necessário um *design* customizado ou recurso a JavaScript. Este método não é extensível aos outros editores do Daisy, que regra geral não sabem HTML, de forma que cabe ao autor desenvolver estas páginas, que são as mais aliciantes do ponto de vista conceptual, gráfico e de desenvolvimento.

A principal barreira na criação de documentos LiteralHtml é a transformação que o Daisy aplica a todos os atributos `id` no HTML inserido. O Daisy permite a inserção de

documentos dentro de outros, o que possibilitaria ter *tags* com o mesmo ID dentro de um documento final. De forma a evitar esta colisão de identificadores, o Daisy prefixa o conteúdo de cada `id` com o ID do documento de origem. Assim, se tivermos uma *tag* `<input id="amt1">` dentro de um documento cujo ID Daisy seja `11102-DSY`, ela será renderizada como `<input id="dsy11102-DSY_amt1">`. Esta mudança de ID's causa dificuldades na programação e *debug*.

O autor implementou a seguinte função auxiliar, que deteta se o documento está a ser servido a partir de um servidor HTTP. O uso de ID em JavaScript funciona tanto para documentos locais, como no servidor, bastando usar sempre esta função, p. ex. da seguinte forma: `document.getElementById(id(x))`.

```
function id(t) {
  return (document.URL.substring(0,5)=='http:') ?
    'dsy' + daisy.document.id + '_' + t : t;
}
```

Já no caso do uso do ID pelo HTML, não é possível contornar a situação; p. ex. no seguinte código a `label` ativa o `input` apenas na versão *online*:

```
<input type="radio" value="10" name="amt" id="amt1"/>
<label for="dsy11102-DSY_amt1" id="lblamt1">£10</label>
```

5.7.3 IframeWrapper

Um 'IframeWrapper' é um tipo de documento muito simples: consiste num URI a mostrar dentro de uma `iframe`, a altura dessa `iframe`, e dois blocos de conteúdo opcionais, um antes, outro depois da `iframe`. Foi o primeiro *doctype* criado pelo autor.

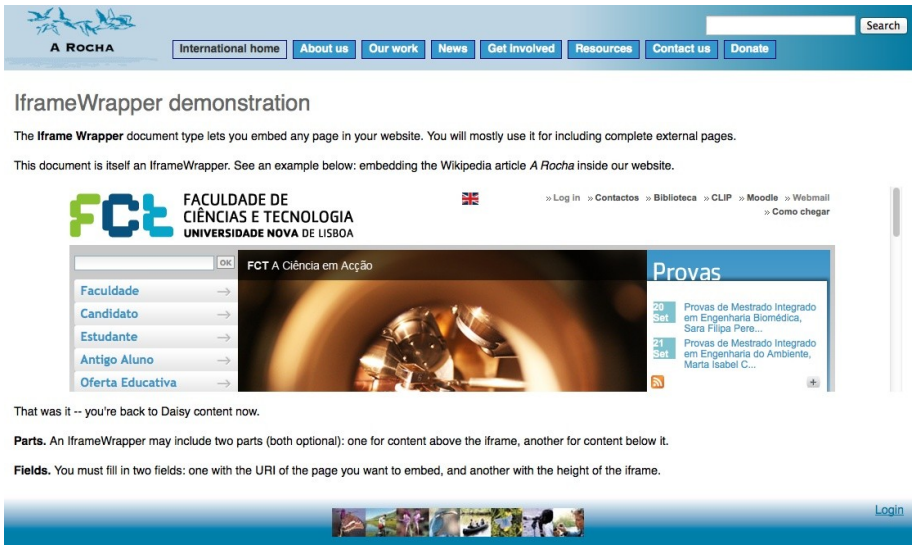

Figura 5.8: IframeWrapper correndo o website da FCT–UNL

5.7.4 News

Quando o *site* funcionava sob Carrelet, sempre que fosse dia 1 era necessário inserir a página com as notícias desse mês e depois editar a página principal do *website*, em todas as sete línguas deste. O autor criou o *doctype* 'News' para ser possível inserir as notícias e traduzi-las

em qualquer data, e as páginas principais serem atualizadas automaticamente no dia em que a notícia devesse ser lançada. Contém o seguinte:

- Corpo do texto (parte): onde se escreve a notícia propriamente dita
- Thumbnail (parte): uma imagem para ilustrar o evento na lista personalizada e na página do evento.
- Data da notícia (campo): obrigatório. Serve para ordenação das notícias. Normalmente a notícia não é mostrada nas listagens, se a data for posterior à data atual; notar que este campo não protege a notícia de ser publicada, pelo que esta se encontra visível através de pesquisa no *website* ou via *feed* RSS.
- Subtipo de notícias: se é uma notícia criada pel'A Rocha, ou se é sobre A Rocha.
- Link URI e Link text (campos): opcionais, para associar um URI à notícia.

5.7.5 MultiColumn

'MultiColumn' é constituído por cinco blocos de conteúdo, todos opcionais: três blocos para três colunas de texto lado a lado, um a toda a largura do topo e um a toda a largura por baixo das colunas. A *homepage* é um exemplo de uso do 'MultiColumn' com as três colunas preenchidas (ver figura 5.3, p. 53).

As partes que não estiverem preenchidas não são mostradas; é assim possível omitir as partes no topo ou em baixo, ou ter apenas duas colunas. Um documento 'MultiColumn' com apenas uma das partes preenchida tem o *layout* idêntico a um 'SimpleDocument'.

As larguras das colunas é configurável, tanto com um valor absoluto, como em percentagem da largura total. Para manter o *layout* fluido, não é possível indicar as larguras de todas as colunas que estejam a ser usadas; caso isso aconteça, a última largura indicada é ignorada.

O 'MultiColumn' coloca uma linha pontilhada entre as colunas existentes; permite também omitir o título da página, já que uma página em três colunas se presta à colocação de um título acima de cada uma das colunas, não necessitando nesse caso de um título geral.

5.7.6 FormMailer

A apresentação de formulários e o envio do seu conteúdo via *email* para um endereço dentro da organização é uma função bastante importante num website, já que tem a possibilidade de se adaptar a todo o tipo de fins e de dados. No caso particular de um formulário de contacto, ao evitar mostrar o endereço de *email* reduz a vulnerabilidade ao *spam*, que era na altura um problema considerável na organização.

É fácil construir um *back-end* CGI para enviar o conteúdo de um formulário para um endereço de *email* específico; mas como criar um mecanismo genérico que permitisse enviar resultados de formulários para diferentes endereços através de um único CGI? Incluir o endereço como um campo escondido apresentava dois tipos de vulnerabilidade aos *spammers*:

1. *Revelava um endereço A Rocha.* Um campo escondido não é renderizado, mas é visível no HTML; basta usar uma expressão regular para procurar o endereço.
2. *Permitia o uso como plataforma de envio de spam.* Os *spammers* poderiam criar um

pedido POST para o CGI substituindo o endereço de destino, e enviar *spam* através do servidor da ARI.

Este problema é passível de resposta através de uma extensão em Apache Cocoon; aliás, a Sr.ª Nik Hatta fê-lo, mas a solução desenvolvida não era localizável nem customizável. O autor gerou a sua própria solução, através de um *doctype*. O endereço vai ser preenchido aquando da criação do documento, no lado do Daisy, e vai ser posteriormente enviado de uma forma codificada para o CGI.

O 'FormMailer' é um documento constituído por HTML livre, no qual vão ser colocados os campos e a formatação do formulário; não é necessário colocar a tag `<form>`, pois esta é fornecida na transformação XSLT, que inclui também como campos escondidos a codificação do endereço. Os campos especiais são os seguintes:

1. *Email title.* Um título para o email a enviar com o conteúdo do formulário.
2. *Encoded address.* Apresenta uma lista de endereços de destino para o formulário, mostrada de forma clara mas guardada de forma codificada; p. ex. se o criador do documento selecionar a opção 'International webmaster: webmaster@arocha.org', o campo ficará preenchido com o valor 'int-web', que será passado para o CGI.
3. *Test address.* Aqui pode ser escrito um endereço de *email*. É possível indicar em 'Encoded address' que será usado este endereço, que é colocado no *form*. Esta função foi incluída para auxiliar ao *debug*: um *webmaster* local pode redirigir para si os resultados do formulário enquanto está a editar a página, e depois de se certificar que está tudo bem removerá o seu endereço de *email* e selecionará em 'Encoded address' o destinatário final do formulário.
4. *Next page.* Por defeito, depois de enviado o formulário com sucesso, o *website* apresentará uma mensagem dizendo simplesmente, "A sua mensagem foi enviada", com um *link* para voltar à página anterior. Esta opção permite indicar o URL de uma página de seguimento, de forma a customizar a confirmação de envio.

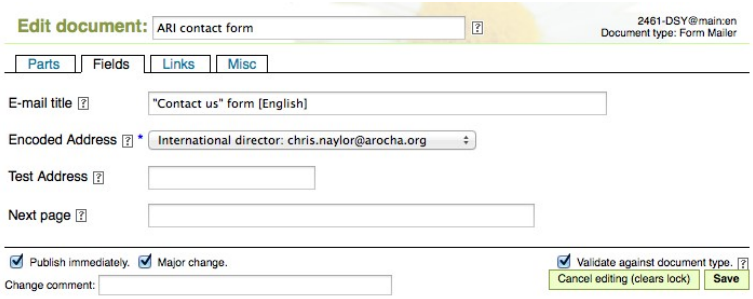

Figura 5.9: Campos de um documento FormMailer

Apesar de ser flexível e não revelar nenhum endereço de *email*, esta solução não protegia contra o envio do formulário com *spam* para o endereço A Rocha correspondente. Como exemplo, cerca de metade dos formulários de voluntariado vinham preenchidos com *spam*. Assim, numa segunda fase o autor decidiu implementar um *captcha* para deter os *spam bots*.

Um *captcha* é uma aplicação que protege *websites* dos *bots*, gerando e avaliando testes que os seres humanos conseguem passar, mas que os *bots* não. ([CMU10]). Um *captcha* envolve

o reconhecimento de padrões de uma forma que os computadores ainda não conseguem fazer: ler texto distorcido, reconhecer imagens, ou encontrar padrões em comum. O captcha escolhido foi o reCAPTCHA, desenvolvido inicialmente pela Carnegie Mellon University e atualmente propriedade da Google, que envolve o reconhecimento de texto distorcido proveniente de *scans* de livros antigos. O reCAPTCHA funciona da seguinte forma (figura cortesia de [HESS09]):

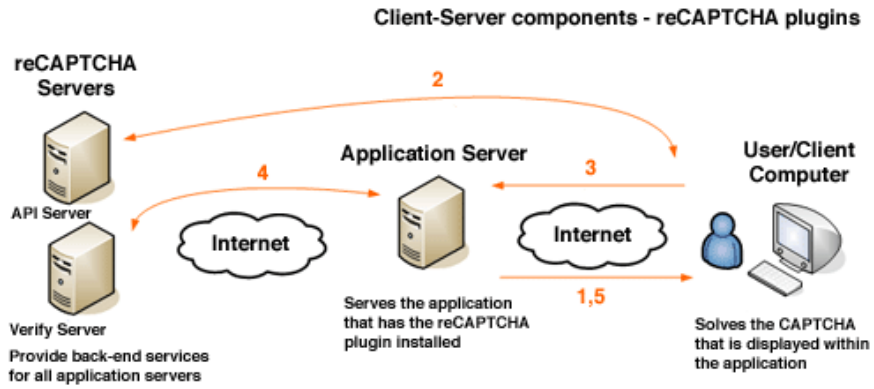

Figura 5.10: Diagrama de funcionamento da API reCAPTCHA

1. O utilizador carrega uma página *web* que inclui o código JavaScript do reCAPTCHA.
2. O *browser* do utilizador pede um desafio ao reCAPTCHA. O reCAPTCHA envia ao utilizador um desafio, e um *token* que identifica o desafio.
3. O utilizador preenche o formulário na página *web*, e submete o resultado ao servidor do *website*, juntamente com o *token*.
4. O servidor do *website* envia o *token* e a resposta do reCAPTCHA ao servidor de verificação do reCAPTCHA, que reponde ao servidor do *website* indicando se o teste passou ou falhou.
5. Cabe agora ao servidor do *website* lidar com o utilizador com base no resultado do teste: se passou deverá ter acesso à ação pretendida (p. ex. o formulário de contacto que preencheu deverá ser enviado); se falhou, verá uma mensagem de erro, e pode ser-lhe permitida uma nova tentativa.

A forma mais simples que o autor encontrou para incluir o *captcha* foi através da criação de uma *tag* específica `<captcha/>`. Essa *tag* deveria ser opcional, isto é, se o editor de uma página não a colocar, o teste não será executado.

Primeiramente foi necessário indicar ao HTMLCleaner, módulo responsável por limpar e formatar o HTML guardado pelo Daisy, que não removesse a *tag* `<captcha/>`, introduzindo essa informação em `daisywikidata/live-htmlcleaner-folder/htmlcleaner.xml`:

```
<element name="captcha"/>
```

Vejamos a parte em HTML de um 'FormMailer', onde o criador do formulário simplesmente colocou `<captcha/>` onde quis (o *tag* está realçado):

```
<html><body><table class="borderless"><tbody>
  <tr>
```

```
    <td>Name:</td>
    <td><input name="Name" type="text" size="40"/></td>
  </tr><tr>
    <td>E-mail:</td>
    <td><input name="Email" type="text" size="40"/></td>
  </tr><tr>
    <td>Town and country:</td>
    <td><input name="Location" type="text" size="40"/></td>
  </tr><tr>
    <td>Message:</td>
    <td><textarea cols="60" rows="6" name="Message"/></td>
  </tr><tr>
    <td/>
    <td><captcha/></td>
  </tr><tr>
    <td/>
    <td><input type="submit" value="Send"/></td>
  </tr>
</tbody></table></body></html>
```

Isso resultou na geração do seguinte HTML, parte modificado em `FormMailer.xsl`, outra parte em `document-to-html.xsl` (as modificações estão realçadas):

```
<form action="/cgi-bin/form-mailer.cgi" method="post">
  <input value="ARI contact form" name="pagename" type="hidden">
  <input value="en" name="language" type="hidden">
  <input value="&quot;Contact us&quot; form [English]" name="msgtitle"
    type="hidden">
  <input value="int-ops" name="addrcode" type="hidden">
  <input value="" name="addrtest" type="hidden">
  <input value="" name="nextpage" type="hidden">
  <table class="borderless">
    <tbody><tr>
      <td>Name:</td>
      <td><input name="Name" type="text" size="40"></td>
    </tr><tr>
      <td>E-mail:</td>
      <td><input name="Email" type="text" size="40"></td>
    </tr><tr>
      <td>Town and country:</td>
      <td><input name="Location" type="text" size="40"></td>
    </tr><tr>
      <td>Message:</td>
      <td><textarea cols="60" rows="6" name="Message"></textarea></td>
    </tr><tr>
      <td></td>
      <td>
        <!--reCAPTCHA begin-->
        <div class="captcha">
          <script type="text/javascript">
            var RecaptchaOptions = {lang:'en', theme:'white'};
```

```
          </script>
          <script src="http://www.google.com/recaptcha/api/challenge?
k=6Lculr4SAAAAAC8x4cXaVcaVIAVCUvIrDufcr" type="text/javascript"></script>
          <noscript>
            <iframe frameborder="0" width="500" height="300"
src="http://www.google.com/recaptcha/api/noscript?
k=6Lculr4SAAAAAC8x4cXaVcaVIAVCUvIrDufcr"></iframe>
            <br>
            <textarea cols="40" rows="3"
name="recaptcha_challenge_field"></textarea>
            <input value="manual_challenge" name="recaptcha_response_field"
type="hidden">
          </noscript>
        </div>
        <!--reCAPTCHA end-->
      </td>
    </tr><tr>
      <td></td>
      <td><input type="submit" value="Send"></td>
    </tr></tbody>
  </table>
</form>
```

Figura 5.11: FormMailer com reCAPTCHA

Finalmente, o autor fez as alterações ao `form-mailer.cgi` para contactar a API reCAPTCHA:

```
#!/usr/bin/perl -w
…
use Captcha::reCAPTCHA;
…
my %private_key = (
      'www.arocha.org' => '6Lculr4SAAAAAC8x4cXaVcaVIAVCUvIrDufcr',
      'www.riadealvor.org' => '6Lcvlr4SAAAAAMEWlCTaPFAjMTwrg-ZP11DNi',
      'www.tanariverdelta.org' => '6Lcwlr4SAAAAABrrzBiz-nWiIau0fUldQsRkr',
      'www.discovernewspecies.org' => '6Lcxlr4SAAAAAG9qn6BBe5hwo4ArqLwh-ooNm'
);
…
# Check CAPTCHA response
my $server = $ENV{'SERVER_NAME'};
my $c = Captcha::reCAPTCHA->new;
```

```
if (defined($vars{'recaptcha_response_field'})) {
  my $result = $c->check_answer(
    $private_key{$server}, $ENV{'REMOTE_ADDR'},
    $vars{'recaptcha_challenge_field'},
    $vars{'recaptcha_response_field'}
  );
  if ( !$result->{is_valid} ) {
    dopage 'CAPTCHA validation error', "The Captcha you provided was wrong.
      Your message was not sent. Please go back and try sending again.\n
      <!--$_[0]-->", $close_en, 1;
  }
}
#send form results now
…
```

O *spam* recebido através do 'FormMailer' baixou para metade com esta alteração.

5.7.7 GoogleMap

Figura 5.12: riadealvor.org com documento 'GoogleMap'

O *doctype* 'GoogleMap' serve para facilitar a colocação de mapas dinâmicos Google Maps nas páginas dos diferentes websites d'A Rocha. Lida com a transformação de atributos `id` nas *tags* de forma transparente, conforme detalhado na secção 5.7.2, p. 60.

É constituído por uma parte em JavaScript, que contém as instruções para desenhar o mapa, e dois blocos de conteúdo opcionais, para colocar antes e depois do mapa. Tem duas variáveis opcionais, a largura (default: 100% da largura disponível) e a altura (default: 400 px).

A figura anterior mostra um mapa que indica os limites do sítio Natura 2000 Ria de Alvor, criado com a introdução do seguinte código JavaScript:

```
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.setCenter(new GLatLng(37.13787, -8.61465), 14, G_HYBRID_MAP);
var polyline = new GPolyline([
  // a polyline está desenhada no sentido dos ponteiros do relógio
  new GLatLng(37.16236, -8.59731), // ponto de maior latitude
  new GLatLng(37.16161, -8.58907), // canto NE
  new GLatLng(37.14205, -8.59087),
  new GLatLng(37.13982, -8.59461),
  new GLatLng(37.13667, -8.59431),
  new GLatLng(37.13007, -8.59714), // Alvor
  new GLatLng(37.12254, -8.59628), // canto SE
  new GLatLng(37.12200, -8.61928),
  new GLatLng(37.11840, -8.64212), // canto SW
  new GLatLng(37.12015, -8.64370),
  new GLatLng(37.12425, -8.63409),
  new GLatLng(37.13185, -8.62838),
  new GLatLng(37.13876, -8.62607), // reentrância a W
  new GLatLng(37.14335, -8.63126),
  new GLatLng(37.14677, -8.63658),
  new GLatLng(37.15087, -8.65160), // canto NW
  new GLatLng(37.15689, -8.64212),
  new GLatLng(37.15888, -8.63499),
  new GLatLng(37.15744, -8.63006),
  new GLatLng(37.15422, -8.62701),
  new GLatLng(37.15227, -8.61941),
  new GLatLng(37.14916, -8.61697), // reentrância a N
  new GLatLng(37.15296, -8.61336),
  new GLatLng(37.15894, -8.59972),
  new GLatLng(37.16236, -8.59731)  //regresso ao ponto original
  ], "#00ff00", 10);
map.addOverlay(polyline);
```

O resultado na página HTML é o indicado abaixo. Notar que o JavaScript indicado acima não é inserido na página, mas é carregado diretamente da parte (código realçado).

```
<style type="text/css">
  #map_3779-DSY { margin-bottom: 10px; width: 100%; height: 600px }
  #map_3779-DSY img { background: none; }
</style>
<div id="map_3779-DSY"></div>
<script type="text/javascript" language="javascript">
// user's javascript file:
var user_script = '/3779-DSY/version/last/part/36/data';
// map object -- not checking if (GBrowserIsCompatible())
var map = new GMap2(document.getElementById("map_3779-DSY"));
// include_dom inserts and executes a javascript file
// kudos to Stoyan Stefanov @ http://www.phpied.com/javascript-include/
function include_dom(script_filename) {
```

```
  var html_doc = document.getElementsByTagName('head').item(0);
  var js = document.createElement('script');
  js.setAttribute('language', 'javascript');
  js.setAttribute('type', 'text/javascript');
  js.setAttribute('src', script_filename);
  html_doc.appendChild(js);
  return false;
}

function afterload(){
  include_dom(user_script);
}
window.onload = afterload;
```

Como área de desenvolvimento futuro deste *doctype*, o autor poderá modificar o código XSLT para tornar possível vários inserir vários documentos 'GoogleMap' na mesma página. Neste momento a execução iria falhar por pelo menos duas razões:

1. Todos os mapas se chamariam 'map', o que provocaria a existência de um único mapa.
2. O mecanismo de carregamento do JavaScript apenas carregaria um *script* de mapa, porque cada atribuição a `window.onload` iria substituir a anterior.

O autor iria também tentar inserir o JavaScript dos mapas diretamente no documento HTML, para diminuir o número de chamadas ao servidor e agilizar o carregamento da página.

5.7.8 Event

A criação do *doctype* 'Event' foi solicitada pel'A Rocha Reino Unido, como forma de publicitar os seus múltiplos eventos a diferentes níveis (nacional, regional e local). É bastante semelhante ao *doctype* 'News', e foi criado a partir deste com os seguintes acrescentos:

- Data final: opcional. Omitir se o evento ocorreu num único dia.
- Localização: onde o evento se vai realizar; campo em texto livre, para impressão.
- GeoTags: permite definir palavras-chave para filtrar os eventos posteriormente. P. ex. um evento com 'Location'='Cambridge', tem em 'GeoTags'='eastofengland, uk'. Assim é possível fazê-lo aparecer numa listagem regional (`WHERE $EventGeoTags LIKE '%EastofEngland%'`) ou nacional (`WHERE $EventGeoTags LIKE '%uk%'`).
- Tipo de evento: outro campo que poderá conter palavras-chave para filtrar o tipo de listagem em que aparece, da mesma forma que 'GeoTags'.

Algumas ideias de desenvolvimento futuro relacionadas com o *doctype* 'Event':

- EventList: um *doctype* parametrizável para listar documentos entre duas datas.
- EventCalendar: mostraria um calendário de eventos. Filtrável por GeoTags e por tipo de evento; possivelmente interativo.

5.8 OUTRAS SKINS EXECUTADAS

Esta secção indica algumas outras skins que o autor executou, ou ajudou a executar.

5.8.1 riadealvor.org

O website `riadealvor.org` [RDA] foi criado pelo autor para apoiar a campanha em defesa da Ria de Alvor—uma área classificada a nível europeu na rede Natura 2000, mas que está em risco de destruição dos seus valores naturais. Já foi incluído um *screenshot* (figura 5.12, p. 67). Esta *skin* apresenta um *layout* mais rígido que `arocha.org`, com uma largura máxima. Não tem um menu superior mas apenas um menu lateral, com os submenus a surgirem em *mouseover*.

5.8.2 tanariverdelta.org

O *website* tanariverdelta.org [TANA] surge numa situação idêntica a `riadealvor.org`. O rio Tana é o maior rio do Quénia, nasce nas montanhas no oeste do país, e flui através de terrenos semidesérticos até ao oceano Índico, espraiando-se num delta com uma área de 1 300 km² rico em espécies animais. Um projeto multinacional de produção de biocombustível ameaça a vida selvagem do delta, bem como o estilo de vida de dezenas de milhares de habitantes locais. A Rocha participa numa plataforma de organizações que lutam pela defesa do delta.

O autor fez esta *skin* baseado na *skin* anterior; ela é de facto idêntica, sendo modificadas apenas algumas cores e a imagem do topo, e retirados alguns elementos que eram específicos de `riadealvor.org`, tais como o *link* para alternar de língua entre inglês e português.

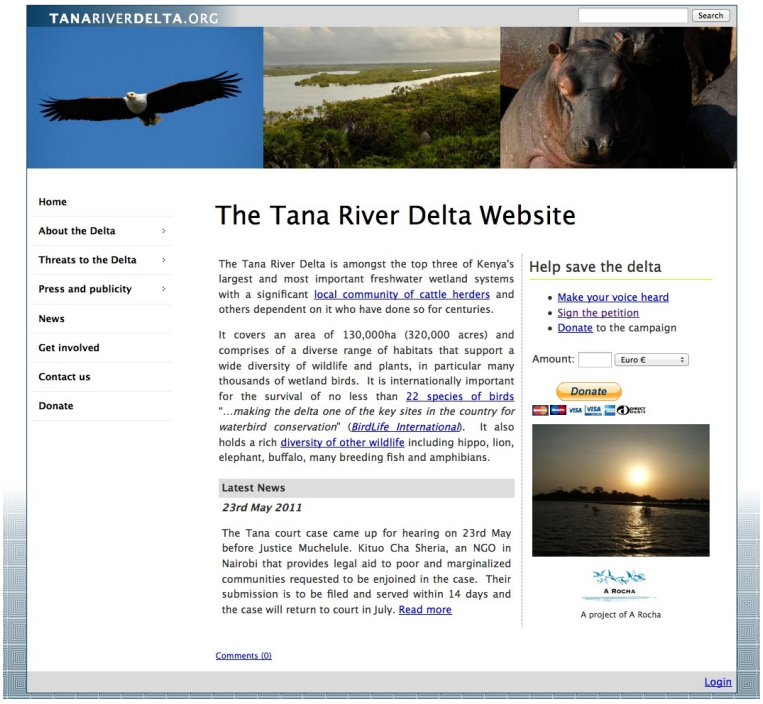

Figura 5.13: tanariverdelta.org

5.8.3 empreintedenature.ch

Esta *skin* foi desenvolvida por Steve Tanner, diretor d'A Rocha Suíça, com base na *skin* `riadealvor`, para um projeto que ainda não se encontra ao vivo. O Steve pretendia aplicar diferentes *designs* às diferentes áreas do website, e não estava a conseguir. Então comunicou essa dificuldade ao autor. Como conseguir esta variação em Daisy?

A solução que o autor apresentou foi calcular o "ramo" do documento de navegação em que se está. O cálculo foi feito da seguinte forma:

```
<xsl:variable name="branchIndex">
  <xsl:value-of select="count($navigationTree/*[@selected][1]/
    preceding-sibling::*) + 1"/>
</xsl:variable>
  <body dir="{$textDirection}" id="doc{$documentId}" class="branch-
{$branchIndex}>
    <xsl:attribute name="class">
      <xsl:value-of select="concat('branch-',$branchIndex)"/>
    </xsl:attribute>
```

Figura 5.14: empreintedenature.ch: designs diferentes em cada área

Passo por passo, a variável `branchIndex` é preenchida da seguinte forma:

1. procuramos o elemento do primeiro nível da árvore de navegação cujo descendente está selecionado: `$navigationTree/*[@selected][1]`...
2. geramos uma lista dos elementos do mesmo nível (portanto de primeiro nível) que ficam antes deste: .../`preceding-sibling::*`
3. contamos quantos são esses elementos e somamos 1, de forma à numeração dos ramos da árvore começar em 1: `count(...) + 1`

Agora que já temos algo como `<body id="doc2183-DSY" dir="ltr" class="branch-4">`, basta usar classes no CSS para dar estilo segundo a área do website:

```
body.branch-1 {
  background: white url('../images/bienvenue/top.png') center -1px no-repeat;
}
body.branch-1 a {
  color: rgb(1,95,44);
}
…
```

```
body.branch-2 {
  background: white url('../images/activities/top.png') center -1px no-repeat;
}
body.branch-2 a{
  color: rgb(147,93,41);
}
…
```

5.9 *BUGS* DESCOBERTOS

Durante a implementação do sistema em Daisy, foram descobertos alguns *bugs* no *software*. Vamos indicar apenas um deles, que embora pequeno é crítico para o desempenho do CMS.

O menu Daisy é gerado dinamicamente, e mostra apenas as ações permitidas pelo *role* ou *roles* assumidos atualmente pelo utilizador. Aquando da definição de *roles* por língua, o autor descobriu que o sistema não permitia a criação de variantes se o utilizador não tivesse permissões de edição sobre a variante que se estava a visualizar. Ou seja, se um tradutor para francês, com permissões para criar documentos em francês, mas não em inglês, estivesse a ver um documento em inglês, o sistema não lhe permitia criar a variante em francês.

A correção era relativamente simples: no ponto de `documentlayout.xsl` em que se define o elemento de menu "Add variant…"—

```
<xsl:if test="$isEditor">
 <createVariant
  href="{concat($documentPath, '/createVariant', $variantQueryString)}"/>
</xsl:if>
```

—era necessário substituir o teste por `<xsl:if test="not($onlyGuestRole)">` de forma a permitir que todos os utilizadores tivessem acesso à opção de criar uma variante. Fica assim adiada a verificação para o momento da gravação da variante, no qual o sistema testa se o utilizador tem permissões de criação.

Este *bug* foi reportado, juntamente com o *patch* necessário (ver [REIS08b]).

5.10 CONCLUSÃO

O processo de implementação de uma solução em Daisy para `arocha.org` (e posteriormente outros websites das organizações A Rocha) foi um pouco limitado pela falta de recursos humanos da parte da ARI. Essa falta de recursos humanos faz com que quaisquer alterações levem bastante tempo, o que compromete um pouco o usufruto das potencialidades do CMS. No entanto, foi possível cobrir bastante terreno nestes três anos desde a adoção do Daisy.

6. VALIDAÇÃO

Neste capítulo, o autor irá *validar* o processo de seleção e implementação do CMS Daisy, e o seu desempenho; ou seja, refletir sobre o método e critérios que levaram à escolha do CMS Daisy, analisar a implementação da solução geral, e também as formas em que o CMS Daisy se revelou mais, ou menos, adequado à tarefa proposta de gerir o website `arocha.org`.

Em alguns casos são dadas sugestões para ultrapassar o problema verificado.

6.1 SELEÇÃO

Quatro anos mais tarde, é possível olhar para o processo de decisão que levou à adoção do CMS Daisy e tecer alguns comentários críticos sobre o mesmo.

Houve no processo de seleção alguma *falta de distanciamento em relação ao CMS anterior*. Após quatro anos com o CMS Carrelet, este foi usado como termo de comparação para o novo sistema. É natural e compreensível que se use a experiência atual como ponto de partida para avaliar a aplicação de *software* que a vem substituir. No entanto, há que evitar cair em extremos, procurando no novo sistema tudo aquilo que o anterior não era.

6.1.1 Favorecimento da complexidade

Um exemplo do que foi dito é o favorecimento da complexidade. O CMS anterior era extremamente simples de usar, o que é uma caraterística a seu favor; no entanto, visto esse sistema ser limitado em muitos aspetos, a sua simplicidade foi subvalorizada. Os problemas introduzidos com a escolha de um sistema complexo foram desconsiderados:

- *Dificuldade pelos programadores em compreender o sistema.* O CMS Daisy é bastante fácil de usar no que diz respeito a fazer *upload* de uma imagem ou a criar uma página, mas em termos de programação é relativamente complexo.
- *Potencialmente mais bugs.* Sendo iguais os outros fatores, em princípio deverão estar escondidos mais bugs em 100 000 linhas de código do que em 10 000.
- *Subaproveitamento do sistema.* Um sistema complexo demonstra um grande conjunto de funcionalidades, do qual apenas um subconjunto nos interessa. É fácil escolher-se um sistema complexo por causa da variedade de opções que nos apresenta, sem ter em consideração a quantidade de opções que nunca usaremos, e por vezes até sem verificar se tem todas as funcionalidades que pretendemos (o sistema parece tão completo que se descura essa verificação).
- *Falta de recursos humanos.* Um sistema complexo tem tipicamente uma menor *pool* de

pessoas qualificadas nele, o que pode gerar um círculo vicioso difícil de quebrar.

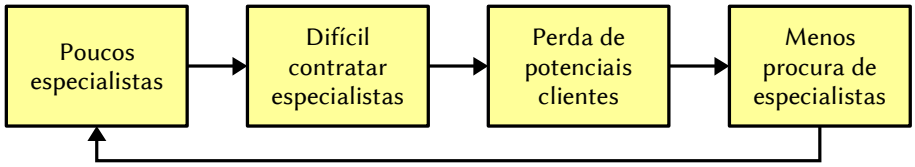

Figura 6.1: Círculo vicioso de falta de recursos humanos

Todas as questões indicadas se fizeram sentir pelo menos um pouco com o Daisy.

6.1.2 Especificações pouco equilibradas

A criação da lista de especificações para o novo CMS foi um ponto positivo do processo: forneceu critérios concretos contra os quais foi possível comparar dois CMS, e determinar qual o melhor. A lista de especificações poderia ter sido mais bem elaborada se tivesse seguido uma metodologia mais rigorosa, e não apenas considerado as necessidades do momento e reagido aos pontos fracos do CMS anterior.

As seguintes especificações, constantes da lista, seriam desnecessárias mesmo em 2007, visto todos os CMS no mercado as cobrirem:

- Autenticação
- Sistema de permissões incluindo permissão de leitura
- *Upload* de ficheiros de qualquer tipo
- Atualização de documentos
- Gestão de documentos e de páginas *web*

A presença destes critérios indica que o CMS existente dominou as considerações para o novo CMS.

6.2 IMPLEMENTAÇÃO

A implementação do *website* `arocha.org` em Daisy levou cerca de 4 meses, com um esforço particular em dois desses meses. No entanto, a implementação levou muito tempo a arrancar; só começou realmente quando a Sr.ª Nik Hatta foi contratada, até porque o autor não recebeu nenhum tipo de formação em XSLT, Cocoon, Daisy ou JavaScript antes da adoção do sistema.

A falta de conhecimentos que o autor então tinha nas tecnologias utilizadas no desenvolvimento fez com que este não pudesse controlar o código da programadora externa. Não há nada a apontar à Sr.ª Nik Hatta, que era boa programadora, como se veio a verificar pela adequação das adaptações que fez à *skin*; mas se não fosse este o caso, o autor não teria forma de o verificar, pois estava a aprender as tecnologias à medida que desenvolvia a solução.

6.3 DESEMPENHO

Esta secção compara o desempenho do sistema face às expetativas geradas durante a fase de seleção do CMS. O autor irá referir-se aos 25 itens definidos nas especificações do CMS desejado para a Rocha (ver seção 2.6, p. 9), bem como aos 3 itens que foram considerados

"vantagens adicionais" do Daisy sobre o outro finalista, Gadara (ver seção 3.4.2, p. 27).

Em relação aos seguintes itens não há nada a indicar, pois são triviais:

- Atualização de documentos
- *Upload* de ficheiros
- Gestão de documentos e de páginas *web*
- Criação de subpáginas
- Compatível com Unicode

Para cada uma das outras caraterísticas, o autor indica como é que o CMS Daisy variou face ao esperado, para melhor ou para pior.

6.3.1 Estabilidade

O Daisy apresenta um problema cuja causa ainda não foi possível determinar. Por vezes, a Daisy Wiki perde ligação ao servidor-repositório. Em alguns casos, o processo está parado; em outros, a ferramenta de teste do Daisy indica que o repositório está em funcionamento.

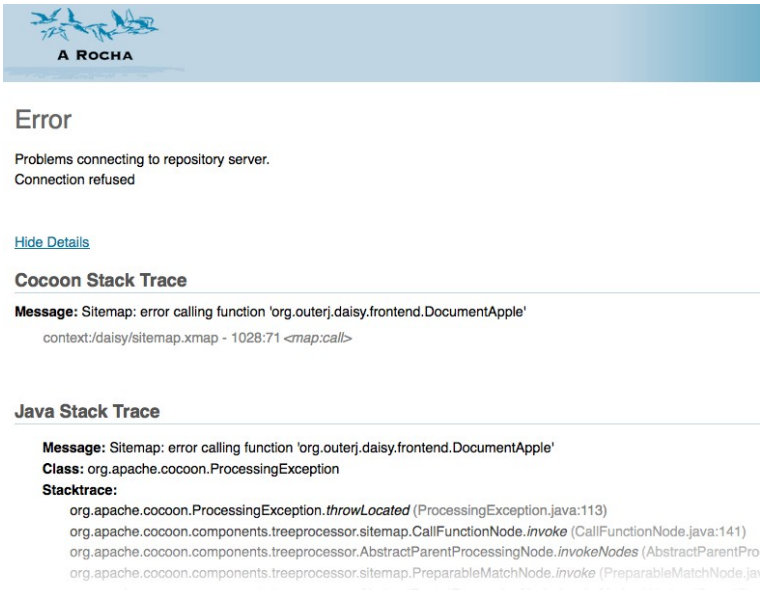

Figura 6.2: Problemas na ligação com o servidor-repositório

A solução para colocar o *website* a funcionar novamente é trivial—basta reiniciar o Daisy—mas esta situação é preocupante porque, embora não aconteça com muita frequência, quando acontece provoca *downtime* e exige uma intervenção manual. Já sucedeu A Rocha ficar sem o *website* p. ex. durante um fim de semana inteiro.

6.3.2 Segurança

Quanto à segurança do sistema, verificam-se problemas de intrusão, embora pouco preocupantes. Todas as semanas são criadas, em média, 1–2 contas de utilizador por *bots*. O sistema envia um email de confirmação, e por vezes essas contas chegam a ser confirmadas. A *skin* tem os comentários removidos para utilizadores com role 'Guest', que é o *role* por defeito, mas no entanto é preciso eliminar essas contas.

Cada vez que um *bot* se regista, o Daisy escreve um registo na base de dados; essa operação poderia ser explorada para fazer um ataque DoS sobre `arocha.org` e os outros *websites* servidos a partir deste repositório.

Uma solução possível para este problema seria incluir um reCAPTCHA na página de registo de novo utilizador. Isso implicaria alterar a página de registo (via XSLT), a componente de tratamento do formulário (em Java) e a página de resultados do registo.

O *login* dos utilizadores é efetuado em HTTP sem segurança, o que significa que a informação de autenticação viaja pela Internet como texto não encriptado. Isso torna-a vulnerável a *eavesdropping* e a ataques *man-in-the-middle*. Seria preferível que o *login* fosse efetuado sob HTTPS. Ainda não se verificou a intrusão no sistema através da conta de um utilizador legítimo.

6.3.3 Escalabilidade

Um *setup* Daisy apresenta algumas condições de escalabilidade; p. ex. o servidor-repositório, o Daisy Wiki e a base de dados podem residir em servidores diferentes. Portanto, em caso de previsão de aumento de tráfego, pode aumentar-se a capacidade do *setup* com facilidade.

A renderização de páginas não se encontra, no entanto, otimizada. Cada página é gerada no momento em que é pedida; não há nenhum tipo de *caching*, seja do pedido à BD, seja da geração da página, o que significa que um pedido repetido da mesma página é gerado novamente, como se se tratasse do primeiro. Seria relativamente fácil programar o sistema de forma a fazer *caching* dos pedidos à BD; seria igualmente possível guardar algum do HTML gerado—pelo menos no caso do utilizador 'Guest', que é o utilizador não autenticado, através do qual são feitos praticamente todos os pedidos.

Já foi referida a necessidade de contratar mais RAM, poucos meses após o início de funcionamento do *website*. A VM Java enchia por completo os 256 MB de RAM inicialmente contratados, e causava o bloqueio do sistema: primeiro fazendo o carregamento de uma página demorar mais de um minuto, e depois fazendo mesmo o Daisy ir abaixo. Após o *upgrade* para 768 MB não tem havido problemas, mas a carga de pedidos sobre o servidor é relativamente baixa: desde o início do ano, a hora de maior tráfego para `arocha.org` contou apenas 1 600 *pageviews*, o que dá uma média de 2,2 segundos entre *pageviews*. Portanto, falta ainda ter uma situação de carga elevada sobre o servidor.

Uma CDN com nós colocados o mais perto possível dos locais onde o conteúdo é pedido pode acelerar consideravelmente a visualização de um *website*. Visto que tudo é servido dinamicamente no Daisy, este sistema não se presta a distribuição via CDN. Um CMS que servisse algum conteúdo estático teria aqui bastante vantagem.

A *skin default* do Daisy não vem preparada para agilizar o carregamento das páginas; em termos de CSS, contém nada menos do que 13 folhas de estilo. É certo que nem todas as folhas de estilo são carregadas nas mesmas páginas, mas bastaria dividir as regras CSS em dois ficheiros, no máximo: um para páginas administrativas, e outro para páginas de conteúdo.

6.3.4 Extensibilidade

O Daisy é efetivamente extensível: através de *sitemaps* Cocoon, de tipos de documento, de

editores de tipos de documento, e do mecanismo de herança de *skins*. Não há nada a apontar.

6.3.5 Roadmap

O desenvolvimento do CMS Daisy está um pouco parado: em três anos o CMS passou da versão 2.2 para a 2.4.2 ([SFOR]). Não há grandes desenvolvimentos a relatar, de forma que A Rocha permanece com a versão 2.2. A Outerthought, empresa que desenvolve o Daisy, está a concentrar os seus esforços em dois outros projetos:

- Kauri: um *framework* Java de desenvolvimento de aplicações e serviços para a *web*, que segue o estilo de arquitetura de *software* REST ([KAUR])
- Lily: um repositório de dados NOSQL construído sobre Apache Hadoop ([LILY])

É óbvio que a Outerthought está interessada em contratos de assistência, e que o desenvolvimento de *software open source* é um meio de atingir esse fim. Se não houver clientes pagadores para o Daisy, o desenvolvimento pára. Talvez nem pudesse ser de outra forma.

6.3.6 Autenticação

O sistema não guarda um histórico de tentativas de acesso, pelo que não é possível aferir através do Daisy se este tem sido alvo de algum ataque.

6.3.7 Sistema de permissões incluindo permissão de leitura

O ACL é dos mecanismos mais poderosos e mais flexíveis do Daisy, simples de usar mas poderoso, com uma granularidade ao nível da variante de documento. O seu funcionamento é impecável.

6.3.8 Versionamento

Também aqui não há nada a apontar ao Daisy; funciona de forma ideal. Até faz *diff*s entre versões, e como normaliza o código HTML antes de o guardar, as *diff*s são independentes do tipo de formatação preferido pelo editor da página. (A normalização do código é customizável através do ficheiro de configuração `htmlcleaner.xml`.)

6.3.9 Histórico para auditoria

O Daisy guarda apenas um histórico de alterações feitas às páginas. Este registo está agregado por variante de documento, portanto não permite ver que páginas foram alteradas por um utilizador. É possível, através uma consulta DQL, saber quais as páginas cujo último editor foi um determinado utilizador, mas esta listagem é demasiado incompleta para servir de controlo de edições de um utilizador.

Seria útil que o Daisy guardasse um histórico de atividade contendo as seguintes operações:

- Tentativa de entrada, entrada e saída do sistema
- Criação, alteração, retirada ou eliminação de documentos
- Todas as operações administrativas, tais como: criação, alteração e eliminação de tipos de documento, partes e campos; criação, alteração e eliminação de utilizadores; alterações ao ACL etc.

Este histórico deveria ser filtrável por utilizador e por tipo de operação, de forma a se poder ter resposta a perguntas como "o que tem feito o utilizador X", ou "quando foram feitas as últimas alterações ao ACL, e por quem".

6.3.10 URL estáveis

A estabilidade dos URL depende da estabilidade da árvore de navegação: uma alteração aí, e o URL muda. Pode no entanto dizer-se que o mecanismo de definição de URL do Daisy tem um desempenho satisfatório.

O facto de haver múltiplos *paths* para a mesma página não é tão bom, pois cria redundância na informação. O autor já teve que guardar o *website* num CD, para demonstração em eventos onde não havia ligação à Internet, e é frustrante ver que o mesmo documento pode surgir diversas vezes; p. ex. os seguintes endereços correspondem todos ao mesmo recurso: a *homepage* da ARI em língua portuguesa.

```
http://www.arocha.org/int-en/2-DSY.html?branch=main&language=pt
http://www.arocha.org/int-pt/index.html
http://www.arocha.org/int-pt/2
```

6.3.11 Localização e internacionalização

Igualmente nada a apontar aqui; o sistema de localização do Daisy funciona perfeitamente, embora o autor não saiba se alguém usa o interface sem ser em língua inglesa.

6.3.12 Gestão de documentos multilingue

É bastante positivo que o Daisy trate traduções de um mesmo conteúdo como diferentes variantes de uma só página, em vez de obrigar à criação de documentos separados para o que são conceptualmente versões do mesmo tipo de informação.

No entanto, a interface de traduções deixa um pouco a desejar. Quando se cria uma variante com base noutra, o editor de documentos apresenta uma só caixa de edição, com o texto na língua de origem dentro da caixa de edição. Facilitaria mais o trabalho de tradução se se apresentasse sempre o texto da língua de origem numa caixa de texto separada, tal como faz o interface de tradução do Plone.

Figura 6.3: Interface de tradução do Plone

6.3.13 *Design* customizável através de *templates*

O sistema de *skins* Daisy é bastante flexível, pois graças ao XSLT permite apresentar os conteúdos do repositório da forma que se entender. No entanto, as *skins* não são exportáveis por diversas razões, nomeadamente:

- O mecanismo de herança de recursos entre *skins* provoca dificuldade em definir que documentos pertencem à *skin*. Podemos ter uma *skin* `B` que deriva da *skin* `A`, com um único documento na pasta `css`. Sabemos que qualquer *skin* contém 13 ficheiros CSS, portanto os outros podem estar na pasta `skins/A/css`. Imaginemos que a *skin* `A` deriva da *skin* `default`; neste caso os ficheiros estão guardados num local completamente diferente da estrutura de ficheiros. E embora se aconselhe vivamente a não alterar a *skin* `default`, mas a criar uma *skin* derivada, para termos a certeza que a *skin* `B` vai funcionar conforme desejado é necessário copiar os ficheiros da `default`. Até porque pode ser a *skin* `default` da versão 2.1 e alguém a querer utilizar no Daisy 2.4.
- Em relação aos tipos de documento, a skin define apenas a sua transformação, e não a sua constituição. A informação sobre campos, partes, tipos de dados e outras caraterísticas é guardada na BD.

Quando andava a investigar os diferentes CMS, o autor não compreendeu porque é que havia galerias com centenas de *skins* para Drupal, e nem uma para Daisy. A existência de *skins* gratuitas na *web* é sempre um bom argumento a favor de um CMS. A não-exportabilidade das *skins* Daisy explica esta omissão.

Seria preferível que o Daisy guardasse toda informação sobre os tipos de documento em diretórios, de forma a tornar as *skins* exportáveis, e abandonasse o mecanismo de herança de recursos das *skins*.

Já agora, as *skins* não são compatíveis entre diferentes versões do Daisy. O XSLT incluído nas *skins* não tem efeito puramente visual, mas altera a lógica de funcionamento do Daisy. Assim, um *upgrade* de versão Daisy implica comparar a *skin* `default` das duas versões, e tentar ver como é que essas alterações devem ser repercutidas nas nossas *skins* customizadas.

6.3.14 Agendamento de conteúdos

O agendamento de conteúdos simplesmente não existe. Independentemente do que era publicitado pelos programadores do Daisy, não há nenhum local onde seja possível indicar uma data para publicação futura de uma variante. Não há muito mais a dizer.

6.3.15 Pesquisa

Tanto a operação de indexação, quanto a pesquisa, são bastante rápidas. O problema principal é a página de resultados de pesquisa, que é extremamente complexa. Dá acesso a uma "pesquisa avançada" que contém uma série de campos bastante intimidadores. Nenhum utilizador que não seja editor do *website* deverá ter que sequer se cruzar com palavras como "Lucene", "collection" e "branch". Este formulário nunca deveria ser mostrado.

Para além disso, a página de resultados de pesquisa é internacionalizável de acordo com a língua de interface definida para o utilizador atual. Ou seja, para visitantes não registados, que usam o utilizador 'Guest', será sempre a mesma língua: o inglês.

Está em curso a substituição da pesquisa no *website* do motor Lucene, que vem com o Daisy, para Google Search.

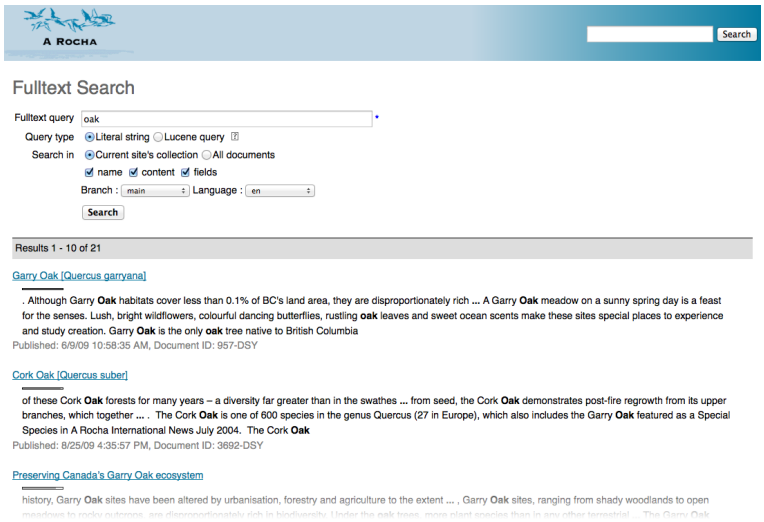

Figura 6.4: Página de resultados de pesquisa

6.3.16 Interatividade

No capítulo da interatividade, o Daisy ficou aquém das expetativas. A única componente de interatividade é o sistema de comentários em páginas, que é extremamente limitado:

- Os comentários são colocados de forma linear, um a seguir ao outro. Não é possível indicar que um comentário é resposta ao comentário de outra pessoa.
- Não permite a inserção de formatação, apenas texto simples.
- O autor de um comentário não o pode alterar, apenas apagá-lo.

Esta limitação no capítulo da interatividade acabou por não ser muito determinante até agora, pois não tem havido nem recursos humanos para lidarem com a interatividade, nem uma estratégia que a contemple. No entanto, essa situação está a mudar: a ARI tem página no Facebook desde 2009, e conta no Twitter desde maio de 2011, e está a elaborar uma estratégia de marketing que indica a necessidade do envolvimento mais próximo com os seus apoiantes e com o público em geral. Portanto, as limitações do Daisy em matéria de interação com os visitantes do website vão sentir-se de forma mais aguda de agora em diante, e isso pode mesmo ditar a mudança para outro CMS.

6.3.17 Formulários

Dizer que o Daisy "tem formulários" é um abuso de linguagem; é melhor dizer que o Daisy "permite formulários". Tendo um sistema flexível de *plug-ins*, e permitindo páginas em HTML livre e tendo também a ARI acesso a um servidor com CGI, isso torna possível que se programem formulários. Não é, no entanto, fácil: são necessários conhecimentos de HTML para se criar um formulário.

Poder-se-á dizer com propriedade que um CMS "tem" formulários se este incluir uma ferramenta acessível, que permita a criação de formulários por leigos em informática.

6.3.18 Curva de aprendizagem suave

De facto é suave a curva de aprendizagem do Daisy, se analisada do ponto de vista do editor de

páginas, que era a intenção do autor.

6.3.19 Migração de conteúdos

O Daisy não tinha nenhum assistente de importação de dados: todo o conteúdo teve que ser inserido manualmente. No entanto, duas componentes do Daisy tornaram a tarefa de inserção de páginas vindas do *site* antigo bastante facilitada:

1. O editor WYSIWYG, HTMLArea, porque aceita conteúdo formatado, por isso foi apenas uma questão de selecionar o conteúdo da página antiga, copiar, colar no editor, e guardar.
2. O módulo `htmlcleaner`, que tratou de remover qualquer má formatação do texto copiado (p. ex. formatação direta com atributos `style` ou tags `<font>`).

O Daisy não permite exportação de dados em formato utilizável por outro CMS, pelo que em princípio teremos o mesmo problema de reinserção manual de documentos se quisermos sair para outro CMS. No entanto, dada a flexibilidade do mecanismo de *skinning* em XSLT, deverá ser possível criar uma *skin* que, em vez de formatar as variantes na BD como HTML, faça o *output* do seu conteúdo segundo um *schema* XML aceitável pelo CMS de destino para importação em massa. Será necessário acautelar a preservação dos *links* internos, ou pelo menos assinalá-los para correção manual.

6.3.20 Suporte técnico e documentação

O Daisy tem documentação escrita, o que ajudou bastante à compreensão do sistema; embora a documentação pudesse ser mais clara, nomeadamente quanto à ordem em que as diferentes folhas XSLT são aplicadas.

O autor inscreveu-se na *mailing list* da comunidade Daisy logo que o servidor foi contratado. De facto, as pessoas ali são bastante prestáveis, tanto os programadores do Daisy na Outerthought, como as outras pessoas que um pouco por todo o mundo têm instalado este CMS. Não há nada a apontar.

A ARI não procurou suporte técnico pago durante estes anos, mas o autor tem resolvido as questões que têm surgido (à exceção da falha misteriosa de ligação com o servidor-repositório).

6.4 OUTROS ASPETOS

Nesta secção serão avaliados alguns aspetos que não foram contemplados na secção anterior por não fazerem parte dos parâmetros de avaliação do Daisy, mas que são ainda assim importantes.

6.4.1 Reação ao Daisy

As pessoas da ARI e das ARNO tiveram uma reação mista ao Daisy. Para algumas, a mudança foi uma complicação: o sistema não funcionava como desejavam, e era mais complicado do que necessário, particularmente em matéria de customização. A não-interatividade deste CMS também pesou negativamente para algumas pessoas.

Para outras, o Daisy teve uma apreciação bastante positiva. Estas pessoas eram

particularmente as que beneficiavam de algum dos aspetos do sistema, como sejam: o suporte para árabe (e posteriormente chinês); o editor visual; a possibilidade de traduzir diretamente no *website*; os diferentes tipos de documento; a inserção automática de conteúdo via consultas; as páginas mais elaboradas, feitas em HTML; e a sensação de que com o Daisy, muitas coisas difíceis eram agora possíveis.

6.4.2 Upgrade de versão

Quando foi feita a atualização da versão 2.1 para a 2.2, o Daisy apresentou alguns problemas com as *skins*. Havia ficheiros que estavam *hard-coded* ao diretório `daisy-2.1`, e que foram mudados depois de se dar conta dos problemas. Os erros não foram imediatamente óbvios, pois não abrangiam todas as páginas.

Como já foi referido, o *upgrade* provoca problemas com as *skins* customizadas, pois espera funcionalidades diferentes da parte do XSLT, particularmente em `layout.xsl` e `document-to-html.xsl`. Essa dificuldade fez com que não fosse feito mais nenhum *upgrade*.

6.4.3 Redimensionamento de imagens

Conforme já referido na secção 5.7.1, p. 59, o doctype 'Image' guarda para além da própria imagem no tamanho que foi inserida, a imagem em dois outros tamanhos: 'thumbnail' (125 px na sua maior dimensão) e 'preview' (250 px). Estes três tamanhos são os únicos que são servidos no *website*. O Daisy nunca redimensiona as imagens quando são pedidas em outros tamanhos; o que o sistema faz é:

1. Se o tamanho especificado para a dimensão maior da imagem for exatamente 125 px, o Daisy serve a parte 'thumbnail';
2. se for 250 px, o Daisy serve a parte 'preview';
3. em todos os outros casos serve o ficheiro de imagem carregado originalmente.

Isto provoca um efeito visual menos cuidado, já que uma imagem redimensionada no *browser* tem menos qualidade do que se for redimensionada num editor de imagem. Mais grave ainda é o desperdício de largura de banda que esta opção acarreta, pois se a imagem fosse transmitida depois de redimensionada certamente ocuparia menos largura de banda.

O autor já pensou numa solução para este problema: criar uma *cache* de imagens fora do Daisy, e servi-las de lá. Em traços largos, a solução seria assim:

1. Criar um diretório e servi-lo estaticamente, p. ex. `www.arocha.org/static/img`
2. Definir uma convenção para passar toda a informação necessária sobre uma imagem no seu nome. Um exemplo seria: `<id>|<branch>|<language>|<width>|<height>`, o que poderia dar exemplos como `11458-DSY|main|default|200|` (neste caso, a altura da imagem não foi indicada).
3. Modificar a *skin* para, em vez de servir as imagens, gerar um pedido de imagem ao diretório estático usando o formato definido; usando o exemplo acima, o pedido seria `www.arocha.org/static/img/11458-DSY|main|default|200|`.
4. Configurar o Apache para correr um *script* (p. ex. `/cgi-bin/serveimg`) sempre que uma imagem seja solicitada e não encontrada em `/static/img`.

5. Criar o *script* (bash ou perl) que faz o seguinte: a) descodifica os diferentes parâmetros do pedido de imagem; b) solicita a imagem ao servidor-repositório; c) redimensiona-a e guarda-a com o nome de ficheiro necessário; d) renova a *cache*, começando pelos ficheiros de imagem mais antigos, apagando aqueles cujo *timestamp* esteja além da idade máxima; e) verifica se se ultrapassou a quota de tamanho da cache de imagens, e se sim elimina ficheiros, começando pelos mais antigos, até a dimensão *cache* de imagens ficar dentro do limite aceitável; f) serve o ficheiro recém-criado, tendo o cuidado de passar o *response code* correto (200 e não 404).

Este desafio interessante fica para quando houver oportunidade. Apresenta ainda a vantagem de fornecer ficheiros de imagem que poderiam ser servidos através de um CDN, o que representaria uma dupla aceleração do sistema.

A outra opção para resolver o problema do tratamento de imagens é mudar de CMS!

6.5 CONCLUSÃO

O Daisy recebe no geral uma avaliação positiva: é potente, flexível e tem servido as necessidades da ARI até ao momento. O processo de decisão, apesar de alguns problemas estruturais, também decorreu de forma bastante positiva.

No entanto, o Daisy tem alguns problemas que muito provavelmente farão com que seja substituído por outro CMS a curto ou médio prazo:

- Instabilidade da ligação com o servidor-repositório
- Dificuldade em encontrar especialistas em Daisy
- Impossibilidade de separação de conteúdos estáticos para serem servidos via CDN
- Impossibilidade de importar ou exportar *skins*
- Histórico para auditoria bastante incompleto
- Falta de ferramentas de interatividade
- Tratamento de imagens

Esta página foi intencionalmente deixada em branco.

7. CONCLUSÕES FINAIS

O último capítulo desta tese de mestrado em Engenharia Informática, inserido no programa "Ser Mestre", aponta para o futuro: considerações sobre um próximo CMS para a ARI, e também áreas de estudo e desenvolvimento dentro da gestão de conteúdos *online*.

7.1 CONSIDERAÇÕES PARA O PRÓXIMO CMS

7.1.1 Opções estratégicas para a presença online

Antes de optarmos por algum CMS, devemos definir primeiro uma estratégia para comunicações; e depois, que objetivos devem ser cumpridos pelo *website*.

A estratégia de comunicações que tem vindo a ser delineada e colocada em prática aponta para a valorização da interatividade e aumento do envolvimento *online*. Assim, o novo CMS deverá permitir a interatividade.

7.1.2 Vida útil da solução

Deverá ser definida a "shelf-life" do CMS; idealmente ciclos de quatro ou cinco anos, porque:

1. *A tecnologia muda*, e vão sendo desenvolvidas soluções melhores para comunicação.
2. *As pessoas mudam*, e a forma como interagem com os *websites* também.
3. *As organizações mudam*, e muda a forma como se apresentam e comunicam.

7.1.3 Definição rigorosa de especificações

A lista de especificações para o próximo CMS deverá seguir uma metodologia mais rigorosa, de forma a assegurar que esta abrange tudo aquilo que será requerido do sistema. Uma forma simples e rápida de estruturar a lista de especificações seria começar por procurar um *website* com uma lista de caraterísticas de CMS, divididas por categorias. P. ex. [MATR] indica as seguintes categorias:

1. *Requerimentos do sistema*: linguagem de programação, OS, servidor, custos etc.
2. *Segurança*: privilégios granulares, versionamento, fluxo de aprovação de conteúdos etc.
3. *Suporte*: formação, manuais, suporte profissional, API etc.
4. *Facilidade de utilização*: edição visual, *templates*, *upload* em massa etc.
5. *Performance*: *caching*, *load balancing* etc.
6. *Gestão*: administração *online*, gestão de traduções, agendamento de conteúdos etc.
7. *Interoperabilidade*: UTF-8, XHTML, CSS, WAI etc.

8. *Flexibilidade*: customização de URL, gestão integrada de conteúdo multilingue etc.
9. *Aplicações incluídas*: fórum de discussão, *blog*, galeria de fotos, gestão de eventos etc.
10. *Comércio eletrónico*: carrinho de compras, subscrições, gestão de inventário etc.

Este passo assegurará que se ponderam todas as categorias, para evitar p. ex. que por se ter um sistema onde o desempenho sob carga não é um problema, se esquece de verificar a *performance* sob carga do novo sistema.

O passo seguinte será registar numa folha de cálculo todas as caraterísticas e as respetivas categorias. Depois analisa-se cada característica e atribui-se-lhe uma ponderação numérica relativamente à sua relevância face às necessidades da ARI, p. ex.: 0=irrelevante; 1=pouco importante; 2=muito importante; 3=crítico.

Cada CMS em estudo será avaliado quantitativamente face a cada requisito (excluindo os requisitos irrelevantes), p. ex. com 0=muito mau; 1=fraco; 2=médio; 3=bom; 4=excelente. Podem definir-se classificações mínimas para determinadas caraterísticas, e chumbar-se liminarmente um CMS se não passar esse mínimo: digamos, ter "médio" a "versionamento". A pontuação total T de um CMS i será calculada pela seguinte fórmula:

$$T_i = 100\% \times \frac{\sum_{j=1}^{n} S_{ij} \times W_j}{\sum_{j=1}^{n} S_{max} \times W_j}$$, em que:

W_j : peso da caraterística j

S_{ij} : pontuação do CMS i na caraterística j

S_{max} : pontuação máxima em todas as categorias

7.1.4 Lista de especificações revistas

Os próximos critérios de escolha de um CMS deverão apresentar uma ênfase mais equilibrada entre as diferentes categorias. O autor indica de seguida alguns critérios por categorias, que serão particularmente valorizados:

- Segurança: Histórico de operações.
- Facilidade de utilização: *Upload* de documentos em massa.
- Performance: Escalabilidade; *Caching*; Uso de CDN.
- Gestão: Agendamento de conteúdos; *Workflow* de publicação.
- Interoperabilidade: Suporte para *metatags* e RDF.
- Flexibilidade: Gestão integrada de traduções.
- Aplicações incluídas: Fórum de discussão, e outras aplicações com interatividade.

7.2 UMA PRÓXIMA IMPLEMENTAÇÃO DE CMS

O próximo processo de mudança de CMS deverá exibir maior celeridade entre a tomada da decisão e a implementação do CMS, até porque provavelmente só se vai decidir sobre um CMS quando estiver aprovado um orçamento que contemple a sua implementação. Assim, o autor prevê que para a próxima vez, entre a proposta de novo CMS e o início da implementação deverão mediar uns dois meses, no máximo.

Paradoxalmente, a próxima implementação deverá levar *mais tempo* a desenvolver do que a atual. A implementação de `arocha.org` em Daisy ocupou na realidade uns 3–4 meses, mas o *design* foi copiado da implementação anterior. O próximo ciclo de CMS deverá incluir um *redesign* completo do *website*, incluindo uma revisão de objetivos e o recurso a técnicas de usabilidade para decidir funcionalidades e suas implementações; p. ex. o uso de *paper prototyping* como forma de avaliar as opções de *design*, antes mesmo de serem codificadas.

De forma ao desenvolvimento decorrer com maior celeridade, o autor deverá ter formação específica nas tecnologias *web* utilizadas no novo CMS, antes da fase de implementação. A falta de formação atrasa o desenvolvimento, introduz *bugs* e, caso seja contratada ajuda externa para a programação da solução, impede o autor de controlar a qualidade do código produzido pelos programadores contratados.

A contratação de ajuda externa deverá ser feita sob parâmetros bastante rigorosos, através da assinatura de um contrato de prestação de serviços que defina etapas no desenvolvimento, os *deliverables*, os prazos e as *tranches* dos pagamentos, defina um prazo de garantia, após o qual será concluído o pagamento, e indique claramente o que sucede em casos de atrasos ou outros incumprimentos.

Deverá ser usada uma CDN com o próximo CMS, se este o suportar. Uma análise de *round-trip times* (p. ex. com recurso a [JPING]) e das estatísticas de acesso ao *website*, sugere que um nó do CDN deveria estar colocado em São Paulo, e outro em Vancouver.

7.3 CANDIDATOS JÁ ELIMINADOS

Os seguintes CMS são considerados neste momento fora da corrida para plataforma de conteúdos de `arocha.org`.

7.3.1 Business Catalyst

O *Adobe Business Catalyst* é uma plataforma *all-in-one* para desenvolvimento de *websites*: gestão de conteúdos, comércio *online*, CRM, e *marketing* via *email* ([ADOB11]).

É usado pel'A Rocha EUA [ARUS] há dois anos. A organização reporta que este sistema é demasiado complexo e pouco intuitivo ("as coisas não estão onde logicamente deveriam estar"), e difícil de usar fora das formas estipuladas no produto: p. ex. é difícil fazer os formulários enviarem os dados para um URL fora do sistema.

7.3.2 Drupal

O *Drupal* ([DRUP]) provavelmente ficará na lista de candidatos eliminados, devido a problemas experimentados por outros *webmasters* com a arquitetura de *plug-ins*.

Muita da funcionalidade do sistema é conseguida através de *plug-ins* (p. ex. galerias de fotos), mas em alguns casos estes *plug-ins* são incompatíveis entre si, ou têm erros de programação que comprometem a estabilidade de todo o sistema.

7.3.3 Joomla!

O CMS Joomla! ([JOOM]), rejeitado em 2007, provavelmente continuará rejeitado na próxima escolha de plataforma de publicação online.

A linguagem PHP continua a não inspirar confiança ao autor. Aparentemente, cada atualização de *software* tapa *bugs* das atualizações anteriores, e cria *bugs* para serem corrigidos em atualizações futuras; p. ex. a *release* 5.3.8 foi lançada cinco dias depois da 5.3.7; corrigiu um *bug* introduzido na 5.3.7, e reverteu um "melhoramento" feito nessa versão ([PHP538]).

7.4 FUTUROS CANDIDATOS A CMS PARA AROCHA.ORG

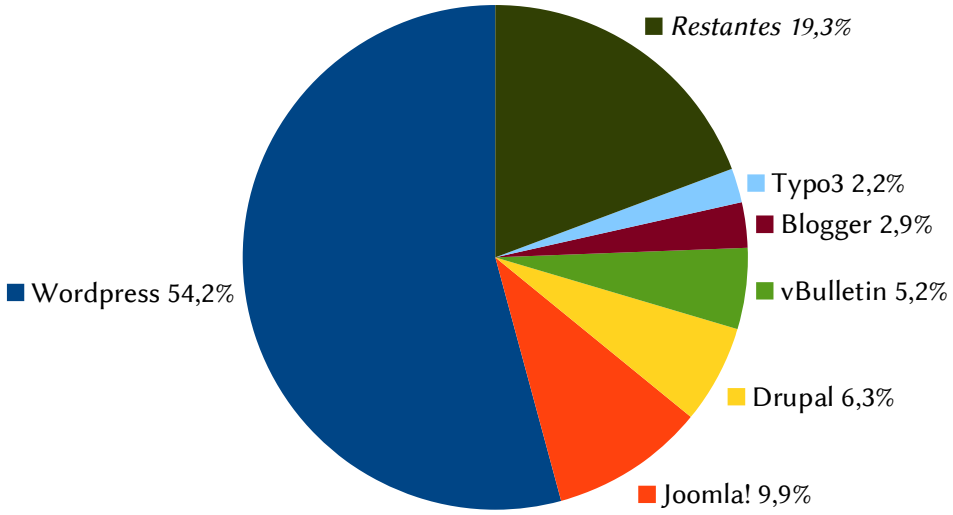

Figura 7.1: Quota de mercado de CMS, setembro de 2011

O autor considera a quota de mercado como sendo apenas uma caraterística de um CMS; não deve ser ignorada, mas não determina a sua aplicabilidade à solução pretendida.

No presente ano de 2011, os seguintes CMS parecem bons candidatos para a edição do website `arocha.org`.

7.4.1 BlueBream

Caso se opte pelo desenvolvimento de uma aplicação *web* para a totalidade ou para parte do *website*, BlueBream parece ser uma boa opção de *framework*. Conhecido como Zope 3 até janeiro de 2010, conta com as seguintes vantagens ([ZOPE10]):

- Escrito em Python, uma linguagem fácil de aprender e de ler, se comparada p. ex. com perl (muito embora neste momento o autor tenha mais prática em perl do que em Python).
- BlueBream segue o paradigma de programação MVC. A separação de funções imposta exigirá à partida um maior rigor na programação, e facilitará a produção de código mais fiável e mais fácil de manter.
- A informação é guardada numa base de dados transacional de objetos, extremamente poderosa e fácil de usar—muito mais fácil do que p. ex. transformar os objetos em registos para inserção numa base de dados relacional.

7.4.2 Daisy

Apesar das dificuldades sentidas com a implementação do CMS Daisy em `arocha.org`, a presente plataforma de publicação usada pela ARI será novamente considerada para continuar por mais um ciclo de quatro anos. As seguintes considerações pesam a favor do CMS Daisy:

1. *Conteúdo.* Deixará de ser necessário exportar o conteúdo para outro CMS. Isto representa uma poupança de tempo da ordem das centenas de horas. Claro que a poupança relativa depende de se querer mover todo o conteúdo existente para uma nova versão do *website*; ver secção 7.1.1, p. 85.
2. *Transição suave.* `arocha.org` pode mudar de estratégia aos poucos, e ir adaptando o seu *website*. Esta forma de renovação da sua presença *online* tem menos visibilidade junto do público, mas permite poupar tempo e dinheiro, e é mais flexível a mudanças do que um corte completo com o CMS anterior.
3. *Know-how.* O *webmaster* já sabe trabalhar com o Daisy, e já conhece as suas vantagens e inconvenientes. Os outros candidatos são à partida desconhecidos—por muito que se leia sobre um CMS, só se conhece realmente o software que já se experimentou pessoalmente.
4. *Simplificação de objetivos.* `arocha.org` provavelmente já não será um *website* para diversas ARNO, mas apenas para a ARI. Assim, bastará atender às necessidades de uma só organização, e não será necessário ter em consideração a capacidade de outros *webmasters* de usarem o CMS.
5. *O Daisy tem mais para dar.* Não foram exploradas todas as potencialidades do CMS atual. Em particular, o mecanismo de navegação facetada parece promissor para áreas do *website* que contenham bastante informação; p. ex. um repositório de documentos sobre a investigação científica feita pela organização, ou uma galeria de imagens para uso interno.

Continuam a fazer-se sentir alguns dos problemas com o CMS Daisy, como a falta de recursos humanos competentes neste produto, pelo que permanecer com o Daisy pode não ser uma solução ideal.

7.4.3 Plone 4

A versão 3 do CMS Plone já foi tratada em detalhe na secção 3.2.6, p. 22. A última versão à data de escrita desta dissertação é a 4.1, que saiu em agosto de 2011 ([PLON11]). As principais diferenças face ao Plone 3 são as seguintes:

1. Um novo tema gráfico, com ficheiros CSS simplificados para facilitar a sua alteração
2. Melhor desempenho (50% mais rápido)
3. Mudança para o editor visual TinyMCE, mais intuitivo
4. Mais opções na página de pesquisa (na realidade será de ver se é possível *retirar* opções da página de pesquisa, porque os problemas de usabilidade aumentam com a complexidade das mesmas; ver [ROBE11])
5. Indexação de texto melhorada

6. Facilidade em customizar páginas para categorias de utilizadores (p. ex. poder-se-ia oferecer perfis para membros dos media, para cientistas, para educadores; a pessoa iria definir o seu perfil quando se registasse no *website*; e depois veria apenas os conteúdos que considerássemos mais relevantes para ela)
7. Documentos grandes guardados no sistema de ficheiros em vez de num *blob* na base de dados (melhora bastante a performance ao evitar p. ex. obter ficheiros áudio através de consultas à base de dados)
8. Integração com o *framework* JavaScript jQuery
9. Usa em média menos 20% de RAM
10. Suporte para versões mais avançadas de Python (2.6) e Zope (2.12; 2.13 no Plone 4.1)

O Plone 4 é assim um sério candidato a ser o próximo CMS para `arocha.org`.

7.4.4 Ruby + Sinatra + Haml + Sass

Caso se opte pelo desenvolvimento de uma aplicação *web* para a totalidade ou para parte do *website*, Sinatra parece ser uma boa opção de *framework* para aplicações *web*.

Sinatra é uma linguagem de domínio específico (DSL) escrita em Ruby. Tem uma estrutura mínima. Não segue a arquitetura de desenvolvimento MVC como p. ex. Ruby on Rails, tendo sido desenhada para criar rapidamente aplicações *web* em Ruby com o mínimo de esforço ([SINA]).

Haml é uma DSL que permite gerar HTML e XHTML de forma elegante, com um mínimo de repetição ([WEIZ11]). Está escrita em Ruby, pelo que o módulo `haml` está acessível ao *framework* Sinatra.

Sass é uma metalinguagem de CSS: uma linguagem que permite gerar folhas de estilos de forma mais eficiente, com recurso a elementos que do ponto de vista da programação "faltam" ao caráter puramente declarativo do CSS, tais como variáveis, blocos e herança de seletores ([CATL11]). Está escrita em Ruby, pelo que o módulo `sass` está acessível ao *framework* Sinatra.

O *framework* na sua totalidade aparenta ser bastante adequado. Para além disso, há já um bom conjunto de programadores em Ruby que podem prestar os seus serviços no desenvolvimento de uma solução.

7.4.5 WordPress

WordPress é uma ferramenta de *blogging* e de publicação de conteúdos escrita em PHP sobre base de dados MySQL ([WORD]). Segundo [W3TE11], tem mais de 50% de quota de mercado de CMS, pelo que merece uma vista de olhos.

Este CMS parece ser facilmente otimizado para velocidade através do *plug-in* WP Super Cache, que guarda cópias estáticas do HTML das páginas. Sempre que possível, a cópia estática é servida; noutros casos o *plug-in* corre apenas o PHP que gera a página, evitando o acesso à base de dados ([OCAO]). Aparenta ser bastante estável, e é um caso em que um excelente *plug-in* pode ser uma boa razão para se instalar um CMS.

Outras vantagens do WordPress incluem ([KINN11]):

1. Uma comunidade de utilizadores extremamente alargada;
2. *Layout* e *design* das páginas fácil de customizar, o que permite dar a um *site* WordPress um ar fresco e original;
3. Grande número e variedade de *plug-ins*, aparentemente programados sobre uma arquitetura sólida;
4. Grande quantidade de programadores de boa qualidade;
5. Ciclo de desenvolvimento rápido, com melhoramentos reais todos os anos.

Entre as desvantagens, contamos as seguintes:

1. O código *core* é um pouco fraco, embora melhore de qualidade todos os anos;
2. Grande quantidade de programadores sem qualidade, pelo que pode ser difícil distinguir os bons dos maus;
3. Um série de *plug-ins* muito populares mas extremamente mal programados, que provocam sérios problemas de performance;
4. O mesmo se passa com os alguns temas: muito populares, mal programados, comprometem a possibilidade de se atualizar para as versões seguintes do CMS;
5. Ciclo de *upgrades* demasiado rápido, por diversas razões (incluindo problemas de segurança), obrigando a atualizações frequentes do CMS, com a consequente verificação se os *plug-ins*, temas e outras customizações ainda são compatíveis com a nova versão.

O WordPress está desde junho de 2011 a ser usado pel'A Rocha Canadá para o seu *website* (ver [ARCA]). É muito cedo para se ter uma opinião sobre a *performance* do WordPress, a sua flexibilidade, ou a sua adequação ao tipo de *site* que a ARI pretende. No entanto, os resultados d'A Rocha Canadá serão certamente partilhados com a ARI.

7.5 VETORES PARA INVESTIGAÇÃO FUTURA

Os CMS são de existência muito recente: os mais antigos datam da década de 1990. Esta ferramenta, e o conjunto de tecnologias que usa, encontram-se em constante mutação, pelo que há muitas possibilidades de investigação futura, de forma a aumentar a maturidade do produto CMS, p. ex. com evoluções das tecnologias subjacentes, como XSLT, RDF, HTML 5, AJAX.

Uma área de investigação é o **desenvolvimento de *standards* de interoperabilidade semântica**. Já existem *standards* de interoperabilidade sintática, como o XML, que asseguram que os *websites* usem as mesmas letras para comunicar; mas a interoperabilidade sintática iria tentar que falassem a mesma língua. Presentemente é muito difícil (muitas vezes impossível) exportar dados entre CMS. Isto provoca um *lock-in* no produto que prejudica os *webmasters*, e distorce a concorrência entre CMS; gera dificuldades no processo de escolha e *deployment*; e acarreta muitas vezes perdas de informação pelo conteúdo que fica para trás.

A criação de um *schema* XML que cobrisse a exportação de *roles*, utilizadores, tipos de documentos, documentos, versões e línguas entre CMS (ou uma subseção destes) seria uma inovação bem vinda.

Esta página foi intencionalmente deixada em branco.

8. BIBLIOGRAFIA

[ADOB11] Adobe Systems Inc.. *Adobe Business Catalyst*. Consultado on-line em 18 de agosto de 2011 a partir de http://businesscatalyst.com/. Data da página: 2011.

[APAC08] Apache Software Foundation. *Apache Cocoon 2.2: Overview*. Consultado on-line em 18 de agosto de 2011 a partir de http://cocoon.apache.org/2.2/. Data da página: 2008.

[AR08] Assembleia da República. "Resolução da Assembleia da República n.º 35/2008", in *Diário da República n.º 145/2008, Série I*. Lisboa, 29 de julho de 2008: INCM.

[ARCA] A Rocha Canada. *A Rocha: Environmental Stewardship*. Consultado on-line em 18 de agosto de 2011 a partir de http://www.arocha.ca/. Data da página: 2011.

[ARI] A Rocha International. *A Rocha: Christians in Conservation*. Consultado on-line em 20 de setembro de 2011 a partir de http://www.arocha.org/. Data da página: 16 de setembro de 2011.

[ARUS] A Rocha USA. *A Rocha—Christians in Conservation*. Consultado on-line em 18 de agosto de 2011 a partir de http://www.arocha-usa.org/. Data da página: s/d.

[BERR73] Berry, Wendell. "Manifesto: The Mad Farmer Liberation Front", in *The Country of Marriage*. New York, 1973: Harcourt, Brace, Jovanovich.

[BOS11] Bos, Bert et al. (eds.); World Wide Web Consortium. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. Consultado on-line em 4 de agosto de 2011 a partir de http://www.w3.org/TR/2011/REC-CSS2-20110607. Data da página: 7 de junho de 2011.

[BRAY96] Bray, Tim et al. (eds.); World Wide Web Consortium. *Extensible Markup Language (XML) 1.1 (Second Edition)*. Consultado on-line em 4 de agosto de 2011 a partir de http://www.w3.org/TR/2006/REC-xml11-20060816/. Data da página: 29 de setembro de 1996.

[BURN87] Burns, Robert. "Verses Written With A Pencil Over the Chimney-piece in the Parlour of the Inn at Kenmore, Taymouth [1787]", in *The Harvard Classics, vol. 6: Robert Burns (1759–1796): Poems and Songs*. Cambridge, Mass., 1914: Harvard University.

[CATL11] Catlin, Hampton; Nathan Weizenbaum; Chris Eppstein. *Sass – Syntactically Awesome Stylesheets*. Consultado on-line em 24 de agosto de 2011 a partir de http://sass-lang.com/. Data da página: 2011.

[CC11] Conselho Científico da FCT–UNL. *Despacho CC–2/2011: Normas para Formatação e Apresentação de Dissertações de Mestrado e de Doutoramento*. Caparica, fevereiro de 2011: FCT–UNL.

[CHAL04] Challenger, James R.; Paul Dantzig; Arun Iyengar; Mark S. Squillante; Li Zhang. "Efficiently Serving Dynamic Data at Highly Accessed Web Sites", in *IEEE/ACM Transactions on Networking, vol. 12, no. 2.* , abril de 2004: .

[CLAR99a] Clark, James. *XSL Transformations (XSLT) Version 1.0*. Consultado on-line em 18 de agosto de 2011 a partir de http://www.w3.org/TR/xslt. Data da página: 16 de novembro de 1999.

[CLAR99b] Clark, James et al. (eds.); World Wide Web Consortium. *XML Path Language (XPath) Version 1.0*. Consultado on-line em 4 de agosto de 2011 a partir de http://www.w3.org/TR/1999/REC-xpath-19991116/. Data da página: 16 de novembro de 1999.

[CMU10] Carnegie Mellon University. *CAPTCHA: Telling Humans and Computers Apart Automatically*. Consultado on-line em 17 de setembro de 2011 a partir de http://www.captcha.net/. Data da página: 2010.

[COCO] CocoonDev. *Faceted Browser*. Consultado on-line em 10 de setembro de 2011 a partir de http://cocoondev.org/main/facetedBrowser/default. Data da página: s/d.

[DI11a] Departamento de Informática da FCT–UNL. *Mestrado em Engenharia Informática: Dissertação via Relatório: Guião para a sua Elaboração*. Caparica, maio de 2011: FCT–UNL.

[DI11b] Departamento de Informática da FCT–UNL. *MEI – Dissertações*. Consultado on-line em 1 de agosto de 2011 a partir de http://www.di.fct.unl.pt/ensino/mestrado-em-engenharia-informatica/2010-2011/mei-dissertacoes. Data da página: 2011.

[DOCF11] Docforge.com. *Content management system*. Consultado on-line em 1 de setembro de 2011 a partir de http://docforge.com/wiki/Content_management_system. Data da página: 24 de agosto de 2011.

[DRUP] Drupal.org. *Drupal—Open Source CMS*. Consultado on-line em 18 de agosto de 2011 a partir de http://drupal.org/. Data da página: s/d.

[DRUP-CORE] drupal.org. *Core modules*. Consultado on-line em 4 de setembro de 2011 a partir de http://web.archive.org/web/20070324092005/http://drupal.org/handbook/modules. Data da página: 24 de março de 2007.

[DRUP-MOD] drupal.org. *Download & Extend*. Consultado on-line em 3 de setembro de 2011 a partir de http://drupal.org/project/modules. Data da página: s/d.

[HESS09] Hess, Ron. *Adding CAPTCHA to Force.com Sites*. Consultado on-line em 17 de setembro de 2011 a partir de http://wiki.developerforce.com/index.php/Adding_CAPTCHA_to_Force.com_Sites. Data da página: 15 de junho de 2009.

[ISO8879] International Organization for Standardization. *ISO 8879:1986: Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*. Consultado on-line em 4 de agosto de 2011 a partir de http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=28557. Data da página: outubro de 1986.

[JOHN08] Johnston, Sam. *Cloud Computing Sample Architecture*. Consultado on-line em 14 de setembro de 2011 a partir de http://commons.wikimedia.org/wiki/File:CloudComputingSampleArchitecture.svg. Data da página: 20 de agosto de 2008.

[JONE11] Bob Jones. *Comentário em "include()"*. Consultado on-line em 17 de agosto de 2011 a partir de http://www.php.net/manual/en/function.include.php#103011. Data da página: 20 de março de 2011.

[JOOM] Open Source Matters Inc. *What is Joomla?*. Consultado on-line em 18 de agosto de 2011 a partir de http://www.joomla.org/about-joomla.html. Data da página: s/d.

[JPING] Watchmouse Website Monitoring. *Online web-based ping: Free online ping from 50 locations worldwide*. Consultado on-line em 19 de setembro de 2011 a partir de http://just-ping.com/. Data da página: s/d.

[JSM10] Santa Maria, Jason. *CSS3 for web designers*. Consultado on-line em 17 de setembro de 2011 a partir de http://v4.jasonsantamaria.com/articles/css3-for-web-designers/. Data da página: 16 de novembro de 2010.

[KAUR] Outerthought bvba. *Kauri*. Consultado on-line em 19 de setembro de 2011 a partir de http://www.kauriproject.org/. Data da página: s/d.

[KINN11] Kinnear, Alec. *Drupal vs Joomla/Mambo vs WordPress: An Experienced Developer's Perspective*. Consultado on-line em 4 de setembro de 2011 a partir de http://foliovision.com/2011/04/02/drupal-vs-joomla-mambo-vs-wordpress. Data da página: 2 de abril de 2011.

[LILY] Outerthought bvba. *Lily – Smart data, at scale, made easy*. Consultado on-line em 19 de setembro de 2011 a partir de http://www.lilyproject.org/. Data da página: s/d.

[MARC5] Bíblia Online. *Marcos 5*. Consultado on-line em 4 de setembro de 2011 a partir de http://www.bibliaonline.com.br/acf/41/5. Data da página: s/d.

[MATR] Plain Black Corporation. *The CMS Matrix: The Content Management Comparison Tool*. Consultado on-line em 25 de agosto de 2011 a partir de http://www.cmsmatrix.org/. Data da página: s/d.

[MCCA10] McCarron, Shane et al. (eds.); World Wide Web Consortium. *XHTML™ 1.1 – Module-based XHTML – Second Edition*. Consultado on-line em 4 de agosto de 2011 a partir de http://www.w3.org/TR/2010/REC-xhtml11-20101123. Data da página: 23 de novembro de 2010.

[MNE10] Ministério dos Negócios Estrangeiros. "Aviso n.º 255/2010", in *Diário da República n.º 182/2010, Série I*. Lisboa, 17 de setembro de 2010: INCM.

[NERU70] Neruda, Pablo; Nathaniel Tarn (ed.). "Juegas Todos los Días...", in *Selected Poems: A Bilingual Edition*. New York, 1970: Dell.

[NIEL98] Nielsen, Jakob. *Web Pages Must Live Forever*. Consultado on-line em 16 de setembro de 2011 a partir de http://www.useit.com/alertbox/981129.html. Data da página: 29 de novembro de 1998.

[NOEL10] Noels, Steven. *Re: A few questions about v2.4*. Consultado on-line em 18 de agosto de

2011 a partir de http://permalink.gmane.org/gmane.comp.java.daisy.general/9925. Data da página: 20 de junho de 2010.

[OCAO] O Caoimh, Donncha. *WordPress Super Cache*. Consultado on-line em 4 de setembro de 2011 a partir de http://ocaoimh.ie/wp-super-cache/. Data da página: s/d.

[OT-LIVE] Outerthought. *Live Sites*. Consultado on-line em 3 de setembro de 2011 a partir de http://docs.outerthought.org/main/286-daisy.html. Data da página: 15 de julho de 2010.

[OUTE10] Outerthought bvba. *Documentation home [Daisy 2.2]*. Consultado on-line em 8 de setembro de 2011 a partir de http://docs.outerthought.org/daisy-docs-2_2/154-daisy.html. Data da página: 15 de julho de 2010.

[PATL] Cambia. *Patent Lens*. Consultado on-line em 3 de setembro de 2011 a partir de http://www.patentlens.net/. Data da página: 2011.

[PHP06] PHP Group. *A simple tutorial: Something Useful*. Consultado on-line em 17 de agosto de 2011 a partir de http://pt2.php.net/manual/en/tutorial.useful.php. Data da página: 20 de dezembro de 2006.

[PHP538] The PHP Group. *PHP 5.3.8 Released!*. Consultado on-line em 24 de agosto de 2011 a partir de http://www.php.net/archive/2011.php#id2011-08-23-1. Data da página: 23 de agosto de 2011.

[PLON-NAME] Plone Foundation. *What does Plone mean? How is it pronounced?*. Consultado on-line em 5 de setembro de 2011 a partir de http://plone.org/documentation/faq/name. Data da página: s/d.

[PLON-SITE] Plone Foundation. *Plone Sites*. Consultado on-line em 5 de setembro de 2011 a partir de http://plone.org/support/sites/sites_listing. Data da página: s/d.

[PLON07] Plone Foundation. *Plone 3.0 (Aug 21, 2007)*. Consultado on-line em 5 de setembro de 2011 a partir de http://plone.org/products/plone/releases/3.0. Data da página: 21 de agosto de 2007.

[PLON11] Plone Foundation. *Plone 4.1 (Aug 08, 2011)*. Consultado on-line em 5 de setembro de 2011 a partir de http://plone.org/products/plone/releases/4.1. Data da página: 8 de agosto de 2011.

[PROG] Progress Software Corporation. *OpenEdge: Accelerate Your Application Development Process*. Consultado on-line em 27 de julho de 2011 a partir de http://web.progress.com/en/openedge/index.html. Data da página: (s/d).

[RAGG99] Raggett, Dave et al. (eds.); World Wide Web Consortium. *HTML 4.01 Specification*. Consultado on-line em 4 de agosto de 2011 a partir de http://www.w3.org/TR/1999/REC-html401-19991224/. Data da página: 24 de dezembro de 1999.

[RDA] A Rocha. *Ria de Alvor*. Consultado on-line em 17 de setembro de 2011 a partir de http://www.riadealvor.org/. Data da página: 2008.

[REIS04] Reis, Júlio. *Castelos de Portugal*. Consultado on-line em 4 de agosto de 2011 a partir de http://www.tintazul.com.pt/castelos/. Data da página: 16 de dezembro de 2004.

[REIS07a] Reis, Júlio. *Puzzled Puffin: A new site and CMS for A Rocha*. Consultado on-line em

24 de agosto de 2011 a partir de http://scratchpad.wikia.com/wiki/Puzzled_Puffin. Data da página: 2 de abril de 2007.

[REIS07b] Reis, Júlio. *A New Content Management System*. A Rocha, 2007: documento interno, não publicado.

[REIS08a] Reis, Júlio. *Homegrown Daisy Documentation for Previous Users of Carrelet*. A Rocha, janeiro de 2008: documento interno, não publicado.

[REIS08b] Reis, Júlio. *Ticket #674: "Add variant" not available without write permissions to current variant*. Consultado on-line em 16 de setembro de 2011 a partir de http://dev.outerthought.org/trac/outerthought_daisy/ticket/674. Data da página: 4 de novembro de 2008.

[REIS97] Reis, Júlio. *Relatório de Projecto Final de Curso: Sistema de Apoio Técnico Aplicacional*. Monte da Caparica, julho de 1997: Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

[ROBE11] Robertson, James. *Simplify the search user experience*. Consultado on-line em 6 de setembro de 2011 a partir de http://www.steptwo.com.au/papers/cmb_simplifysearch/index.html. Data da página: 11 de setembro de 2008.

[RUCK11] Rucker, JD. *Open source wars: Wordpress vs Drupal vs Joomla*. Consultado on-line em 4 de setembro de 2011 a partir de http://www.techi.com/2011/07/open-source-wars-wordpress-vs-drupal-vs-joomla/. Data da página: 25 de julho de 2011.

[SFOR] Sourceforge. *The Daisy CMS project*. Consultado on-line em 19 de setembro de 2011 a partir de http://sourceforge.net/projects/daisycms/files/daisycms/. Data da página: 21 de fevereiro de 2011.

[SINA] The Sinatra Crew. *Sinatra: README*. Consultado on-line em 23 de agosto de 2011 a partir de http://www.sinatrarb.com/intro. Data da página: 7 de agosto de 2011.

[SPIP] SPIP. *SPIP*. Consultado on-line em 6 de setembro de 2011 a partir de http://www.spip.net/rubrique91.html. Data da página: 1 de setembro de 2011.

[STPX] St Pixels. *Internet church – sacred space online*. Consultado on-line em 4 de setembro de 2011 a partir de http://www.stpixels.com/headline-news. Data da página: 28 de agosto de 2011.

[TANA] A Rocha. *The Tana River Delta Website*. Consultado on-line em 17 de setembro de 2011 a partir de http://www.tanariverdelta.org/. Data da página: 23 de maio de 2011.

[TEXI05] Texin, Tex. *User Interfaces For Right-To-Left Languages*. Consultado on-line em 17 de setembro de 2011 a partir de http://www.i18nguy.com/MiddleEastUI.html. Data da página: 23 de outubro de 2005.

[W3TE11] W³Techs. *Usage of content management systems for websites, August 2011*. Consultado on-line em 18 de agosto de 2011 a partir de http://w3techs.com/technologies/overview/content_management/all. Data da página: agosto de 2011.

[WALS11] Walsh, Michael. *Gov 2.0 guide to Plone*. Consultado on-line em 5 de setembro de

2011 a partir de http://govfresh.com/2011/03/gov-2-0-guide-to-plone/. Data da página: 11 de março de 2011.

[WEIZ11] Weizenbaum, Nathan. *Haml Documentation*. Consultado on-line em 24 de agosto de 2011 a partir de http://haml-lang.com/docs/yardoc/. Data da página: 29 de junho de 2011.

[WORD] WordPress.com. *WordPress › About*. Consultado on-line em 18 de agosto de 2011 a partir de http://wordpress.org/about/. Data da página: s/d.

[ZOPE10] Zope Foundation. *What is BlueBream?*. Consultado on-line em 25 de agosto de 2011 a partir de http://bluebream.zope.org/about/whatis.html. Data da página: 2010.