Pedro da Silveira Vieira da Silva

Uma infra-estrutura de suporte à adaptabilidade em sites web

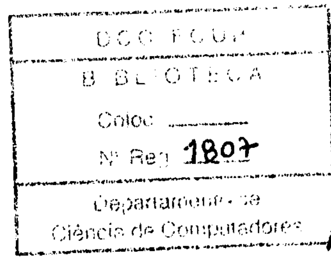
Departamento de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto

Outubro 2006

Pedro da Silveira Vieira da Silva

Uma infra-estrutura de suporte à adaptabilidade em sites web



Tese submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Mestre em Informática

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

Outubro 2006

Aos meus Pais

Agradecimentos

Gostaria de agradecer ao Professor José Paulo Leal por me ter dado a hipótese de realizar este trabalho e pelo apoio que me deu durante toda a dissertação. As suas ideias, conselhos, critícas e boa disposição foram uma mais valia para este trabalho e para mim. Quero também agradecer ao Professor Manuel Eduardo Correia que com os seus truques intermináveis ajudou na resolução de alguns problemas. Gostaria ainda de agradecer, a todos os elementos do NIADD que contribuíram com as suas ideias para as discussões que delinearam os nossos objectivos.

Por fim, obrigado à Liliana e ao Diogo que estiveram sempre disponíveis para fazer o trabalho chato e me cederam os meios para fazer os testes deste trabalho.

Resumo

Esta dissertação apresenta uma solução para a introdução de adaptabilidade em *sites web* existentes ou a desenvolver.

É proposta uma arquitectura distribuída que tem como características: ser independente da tecnologia, abstrair as operações de adaptação e permitir a expansibilidade dos módulos de adaptação. A arquitectura inclui o desenho dos componentes autónomos responsáveis pelas diferentes tarefas necessárias para adaptar uma página: recolha de dados, produção de adaptações e aplicação de transformações. Foi também desenhada uma linguagem para codificar a informação trocada ente os componentes que circula em mensagens e que permitem abstrair os processos a adaptação.

É também descrita a implementação de uma *framework* baseada na arquitectura proposta e um protótipo de uma aplicação que faz uso dessa *framework*. A *framework* é constituída por um módulo cliente, responsável por fazer a recolha de dados e transformações, e por um módulo intermediário, um *broker*, que regista um conjunto de informações e que faz a ligação entre clientes e processos de adaptação. A aplicação protótipo construída usa a *framework* para criar um visualizador *web* dos manuais do *Unix* com algum grau de adaptação.

Foi implementado um módulo cliente em Ajax e que permite a recolha, envio de dados e aplicação de transformações de modo assíncrono. Um meta-adaptador que permite agregar adaptadores em diferentes linguagens de *scripting* e que realiza as adaptações a apresentar na página. Um módulo central que gere as comunicações e regista várias informações.

Foram também realizados um conjunto de testes com a aplicação protótipo para

avaliar o seu comportamento. Estes testes permitiram validar a arquitectura proposta, verificar a aplicabilidade da *framework* desenvolvida e o cumprimentos dos objectivos inicialmente propostos.

Abstract

This thesis presents a solution for introducing adaptability in existing or to be developed web sites.
It is proposed a distributed architecture that is characterized by: technology independence, abstraction of adaption operations and allowing the expandability of the adaptation modules. The architecture includes the design of autonomous components that are responsible for each of the tasks required to get an adapted web page: data gathering, production of adapted results and application of transformations. It as also been design a language that codifies the exchanged information in messages between components and allows the abstraction of the adaption processes.
A framework implementation is also described, based on the proposed architecture, and an application prototype that makes use of that framework. The framework is made up of a client module, responsible for gathering data and make page transformations, and by an intermediary module, a broker, that registers a set of in formations and connects clients and adaption processes. The prototype application makes use of the framework to create a web Unix man pages visualizer with some degree of adaptation. A client module has been implemented with Ajax that allows the gathering of data, sending of data and use of transformations in an asynchronous way. A meta-adapter that allows the use of external adapters built with different scripting languages and produces the adaptions to be used on the pages. A central module that manages the communications and registers several in formations.
Several tests were conducted with the prototype application to assess its behavior. This tests allowed to validate the chosen architecture, verify the applicability of a

framework and the accomplishment of the initial proposed objectives.

Resumé

Cette dissertation présente une solution à l'introduction d'une adaptabilité sur des sites existants ou à développer.
Il vous est présenté une architecture distribuée qui a pour caractéristiques : indépendance de la technologie, abstraction des opérations d'adaptation et possibilité d'expansion des modules d'adaptation. L'architecture inclue le dessin des composants autonomes responsables des différentes opérations nécessaires à l'adaptation d'une page : recueil de données, production d'adaptations et application de transformations. Un langage a également été mis au point pour codifier l'information échangée entre les composants. Celui-ci circule sous forme de messages qui permettent d'abstraire les processus d'adaptation. On y décrit aussi l'implantation d'un framework basé sur l'architecture proposée et un prototype d'une application qui utilise ce même framework. Le framework est constituée par un module client, chargé de recueillir les données et les transformations et par un module intermédiaire, un broker, qui enregistre un ensemble d'informations et qui fait le lien entre les clients et les processus d'adaptation. L'application prototype construite utilise le framework pour créer un visualisateur web des manuels du Unix avec un certain degré d'adaptation.
Un module client a été mis en place en Ajax qui permet de recueillir et d'envoyer des données et d'appliquer des transformations de façon asynchrone. Un meta-adaptateur permet d'agréger des adaptateurs avec différents langages de scripting et réalise les adaptations à présenter sur la page. Un module central gère les communications et enregistre les diverses informations.
Afin d'évaluer le comportement de l'application prototype, un ensemble de tests a

également été réalisé. Ces tests ont permit de valider l'architecture proposée, de vérifier l'applicabilité du framework développé et de vérifier que les objectifs initiaux ont été atteints.

Conteúdo

Resumo **7**

Abstract **9**

Resumé **11**

Índice de Tabelas **17**

Índice de Figuras **20**

1 Introdução **21**

1.1 Adaptação de páginas *web* 22

1.1.1 Recolha de dados de adaptação 24

1.1.2 Integração de mecanismos de adaptação 26

1.1.3 Aplicação de transformações 27

1.1.4 Exemplo de adaptação 28

1.2 Abordagem 29

1.2.1 Objectivos de desenho 30

1.2.2 Arquitectura geral 31

1.2.3 Preparação de *sites* para adaptação 33

1.3 Cenário de utilização 34

1.4 Estrutura da tese e convenções 35

2 Estado da Arte **37**

2.1 Arquitectura orientada a serviços 37

2.2 XML 39

2.2.1 Validação 39

2.2.2 Transformação 40

2.3 *Web Services* 41

2.3.1 SOAP 42

2.3.2 WSDL 42

2.3.3 UDDI 43

2.3.4 WS-* 44

2.4 Ajax 45

2.5 *Browsers* e suporte Ajax 46

2.6 Tecnologia 47

3 Arquitectura do SOM **49**

3.1 Mensagens 51

3.1.1 Estrutura 52

3.1.2 *Namespace* e Validação 57

3.2 Cliente 58

3.3 *Broker* 61

3.4 Adaptador 64

3.5 Características SOA 66

4 Implementação do SOM **69**

4.1 Infra-estrutura de adaptação 70

4.2 Broker 72

4.2.1 *Proxying* 73

4.2.2 *Logging* 74

4.2.3 Implementação 75

4.3 Cliente 78

4.3.1 Utilização de Ajax 79

4.3.2 Preparação dos conteúdos a adaptar 81

4.3.3 Comunicação com o *Broker* 84

4.3.4 Biblioteca de adaptação JavaScript 86

4.4 Adaptador 87

4.4.1 Adaptadores internos 87

4.4.2 Adaptadores externos 88

4.4.3 Implementação 92

4.5 Mensagens 94

5 Avaliação e testes **97**

5.1 Testes de *performance* 100

5.2 Testes de carga 103

5.2.1 Avaliação qualitativa 105

6 Conclusão e trabalho futuro **109**

A Especificação das mensagens **115**

A.1 DTD 115

A.2 XSD 118

B Especificação dos adaptadores externos **123**

C Esquema das mensagens **127**

Bibliografia **130**

Lista de Tabelas

3.1 Valores para o atributo `action` do elemento `admin` 54

5.1 Especificações dos computadores usados para testes. 99

Lista de Figuras

1.1 Proposta para arquitectura geral. 32

3.1 Arquitectura geral de uma solução que faça uso da *framework*. 50

3.2 Elemento `message`. 52

3.3 Elemento `adapt`. 53

3.4 Elemento `notify`. 53

3.5 Elemento `admin`. 54

3.6 Elemento `user`. 55

3.7 Elemento `client`. 55

3.8 Elemento `page`. 55

3.9 Elemento `adaptor`. 55

3.10 Elemento `item`. 56

3.11 Elemento `event`. 56

3.12 Arquitectura da implementação a realizar no Cliente. 60

3.13 Diagrama de actividades dos processos de adaptação e notificação do Cliente. 61

3.14 Diagrama de casos de utilização do *Broker*. 62

4.1 Componentes da *framework* e respectivos "interfaces funcionais". 70

4.2 Estrutura e interfaces disponibilizados pela *framework*. 71

4.3 Esquema da base de dados. 74

4.4 Diagrama de classes do *Broker*. 77

4.5 Diagrama de classes do Cliente. 86

4.6 Diagrama de classes do Adaptador. 93

4.7 Diagrama de classes do interface `Message`. 95

5.1 *Screenshot* do segundo protótipo num momento de utilização. 99

5.2 Tempos médios de acesso, em milissegundos, para um utilizador com um e dois adaptadores. 103

5.3 Tempos médios em milissegundos, para 3 clientes e uma configuração de página com 2 adaptadores, com o número de pedidos a variar entre 100 e 1000. 105

5.4 Percentagens de processamento para cada componente. 106

Capítulo 1

Introdução

O Site-O-Matic[29] é um projecto de investigação do grupo de Inteligência Artificial e Análise de Dados[24] (NIADD) que tem como objectivo, o desenvolvimento de uma plataforma e uma metodologia para automatizar actividades relacionadas com a gestão de *sites*. Exemplos dessas actividades, são a gestão do conteúdo, a sua estruturação, recomendação e personalização. O Site-O-Matic propõe-se ainda a ter em conta o comportamento dos utilizadores e os objectivos do *site* na aquisição dos seus próprios objectivos.

O Site-O-Matic debruça-se sobre três aspectos para atingir os seus objectivos:

- Definir uma plataforma flexível para *sites web* que permita a aquisição de dados de qualidade, assim como transformações e adaptações *online* da estrutura e interface dos *sites*.

- Desenvolver técnicas para realizar adaptações *web* usando *data mining*.

- Obter conteúdos focados em tópicos, sendo essa obtenção guiada por objectivos bem definidos e monitorização de valores de *performance* sobre os dados da utilização.

Para atingir estes objectivos este projecto dividiu-se em várias tarefas, uma das quais deu origem a esta dissertação. Esta tarefa tem por objectivo uma metodologia e uma infra-estrutura que facilite a introdução de adaptabilidade em *sites web*. A infra-estrutura deve permitir a modificação automática de páginas *web* usando mecanismos de adaptação que funcionem sobre dados de utilização. Deve também recolher esses dados necessários à adaptação.

1.1 Adaptação de páginas *web*

Para realizar adaptações sobre páginas *web*, qualquer solução tem de reunir a capacidade de fazer um conjunto de tarefas que vão produzir a adaptação:

- Recolha de dados de adaptação;
- Integração mecanismos de adaptação;
- Aplicação de transformações.

A adaptação em função do utilizador implica saber algo sobre este e com esse conhecimento disponível criar páginas com conteúdo relevante. Para isso, é necessário recolher dados para adaptação, dados que dêem a conhecer algo sobre o utilizador e que possam ser levados em conta na adaptação.

Os dados sobre o utilizador, têm um papel preponderante para os processos de adaptação. Os mecanismos que produzem adaptações usam-nos para moldar os seus resultados, conseguindo assim um refinamento dos resultados e uma maior orientação ao utilizador.

Depois de obtidos os resultados é necessário transformá-los em visualizações para o utilizador, por forma a reflectirem-se de alguma maneira na página que o utilizador está a consultar.

Estas tarefas são comuns e podem ser observadas em alguns *sites*, como é o caso da conhecida livraria *online* Amazon^{TM}.

Quando consultamos o *site*, sem que nunca tenhamos feito uma compra, ao seleccionar um determinado livro são-nos apresentadas outras sugestões de compra, baseadas noutros utilizadores que comprarsam esse mesmo livro ou outros livros dentro do mesmo tema. Aqui, é feita uma adaptação com base em dados recolhidos nas compras de outros utilizadores. Se começarmos a fazer compras, e cada vez em maior número, vemos que as sugestões começam a estreitar o seu domínio ficando cada vez mais perto dos nossos gostos. Isto resulta da recolha de dados sobre as nossas próprias compras que indicam mais sobre as nossas preferências.

Estas sugestões são por exemplo compiladas em listas de sugestões com ordens de preferência, mostrando uma diversificação crescente de gostos à medida que se percorre a lista. Esta é uma das formas de disponibilizar ao utilizador o resultado das adaptações feitas.

Independentemente destas tarefas que são necessárias realizar, existem duas visões distintas de adaptação:

Recomendação Adaptações em que são inseridos novos conteúdos.

Reformatação Adaptações em que o conteúdo não é alterado mas é reorganizado, seja do ponto de vista da sua ordem relativa ou da formatação.

A recomendação está relacionada com a disponibilização ao utilizador de novas informações, como a lista de sugestões de compras. Este tipo de adaptação é uma maneira de introduzir conteúdo orientado ao utilizador num *site*. Há um acréscimo de conteúdo e de informação que não existia inicialmente e que passa a estar disponível.

A reformatação está mais dirigida para a manipulação das informações já existentes no *site*. A alteração de estilos, dando mais destaque a determinadas informações, ou mesmo a alteração da disposição de conteúdos podem ser exemplos desta visão. Nesta visão não há um acréscimo de informação mas um melhoramento, para o utilizador, da informação existente.

Em ambos os casos o objectivo é sempre o de melhorar a experiência do utilizador. Esta dualidade de ideias procura responder a todas as necessidades, não só para

corresponder às expectativas do projecto mas também para manter em aberto a possibilidade de introduzir versatilidade no processo. Se resolvêssemos optar por um aspecto em particular estaríamos à partida a limitar o nosso domínio de intervenção, podendo resultar daí um desinteresse.

1.1.1 Recolha de dados de adaptação

Para recolher informações sobre o utilizador que melhorem os resultados da adaptação é necessário responder a duas perguntas: O que recolher? E onde fazer essa recolha? Respondendo primeiro à segunda questão, existem três opções onde fazer essa recolha de dados:

- Sistemas de informação;
- *Logs* de servidores;
- Interacção do utilizador com a página.

As duas primeiras e mais tradicionais são as usadas pelos *sites* com algum grau de adaptação, caso da Amazon™.
A primeira opção, trata-se de usar os registos dos sistemas de informação para obter dados que permitam fornecer informações sobre o utilizador. Registos de compras, *wish lists*, sugestões e críticas são usadas para sugerir novas compras ou novas publicações.
A utilização destes dados apresentam algumas dificuldades, principalmente porque apenas existem quando é feito algum tipo de transacções como é o caso das compras. Com estes dados há uma dependência da aplicação perdendo-se a capacidade de dar uma resposta genérica de adaptação utilizável a qualquer outra aplicação. Outro inconveniente é para utilizadores que apenas naveguem no *site* e que não ficam abrangidos por estes registos, logo, não beneficiando das adaptações.
Outra hipótese é a utilização dos *logs* do servidor que regista os pedidos feitos pelo

cliente. Com a informação aí presente pode-se perceber algumas das actividades do utilizador, principalmente a navegação entre páginas e daí extrair os itens ou assuntos visualizados. Contudo, os *logs* foram criados com o objectivo de gerir o servidor e não representar dados resultantes das acções do utilizador. Mesmo usando algumas extensões é possível acrescentar alguma informação complementar aos *logs* mas continuam a existir limitações sobre os dados recolhidos. Podemos obter indirectamente a localização geográfica do utilizador, valores de *cookies* ou os *referers* usados (*links* na mesma ou noutras páginas a partir dos quais se chegou à página corrente, apontadores) que permitem relacionar acessos entre sites. No entanto, continuamos limitados a um registo por pedido ou a intervalos de tempo entre acessos que não permitem perceber o que está a acontecer. Por exemplo, se o utilizador está ou não a ler a página.

A terceira alternativa é a recolha de dados junto da fonte, neste caso o cliente *web*. Ao recolher dados de interacção no próprio *browser* ganhamos, quantidade e especificidade. Quantidade porque em vez de um registo por pedido podemos ter vários registos por pedido e especificidade, porque em vez de informação relativa apenas ao pedido podemos ter acesso a dados sobre acontecimentos. Podemos ter dados entre pedidos, algo que não é possível nas duas primeiras opções.

A forma para fazer esta recolha de dados é colocar no cliente, integrado com os conteúdos, ferramentas que possam recolher eventos gerados pelo utilizador. A possibilidade de processar eventos já existe no cliente e apenas é feito um aproveitamento desta grande quantidade de dados, canalizando-a para um registo que mais tarde pode ser usado para refinar a adaptação.

Alguns eventos disponíveis nos *browsers* e a que podemos recorrer são:

click Quando um objecto é seleccionado com um clique de rato;

focus Quando um campo de edição na página recebe destaque ou atenção (*focus);*

scroll Quando a página ou um elemento é deslocado para visualização;

select Quando texto ou outro elemento é seleccionado.

Estas informações permitem aprender muito sobre o comportamento do utilizador e dar algum contexto à sua navegação, sendo possível fazer um registo detalhado das suas acções. Estes eventos podem ser conjugados de forma a produzir **meta-eventos** que traduzam conceitos de utilização e que não era possível perceber até agora. Por exemplo, se existem n cliques numa determinada secção da página num pequeno intervalo de tempo, podemos definir que o utilizador revelou interesse sobre aquela secção. O mesmo pode ser dito de um selecção de texto ou *scroll* para ver determinada parte de uma página oculta.
Podemos também considerar o inverso, se um utilizador estiver *idle*, podemos interpretar como desinteresse ou até ausência.

1.1.2 Integração de mecanismos de adaptação

Para gerar as adaptações é necessário que existam processos que as produzam. Os processos podem ser simples ou complexos, mas de alguma maneira e independentemente da sua complexidade têm de existir.
Por norma estes processos trabalham um conjunto de dados, relacionados ou não com o utilizador dependendo do grau de optimização que se deseja e o fim a que se destinam, para gerar adaptações. Por exemplo, processos de *data-mining* recorrendo a algoritmos de análise de dados com implementações específicas para lidar com conjuntos de dados e relacioná-los, são frequentemente usados para este objectivo.
A utilização de processos deste tipo requer por vezes algumas condições especiais. Condições essas que podem ser por exemplo de *hardware* ou *software*. Podemos ter processos de *data-mining* que tem grandes requisitos de *hardware*, por exemplo, estando distribuídos em *clusters* com acesso a bases de dados de elevado desempenho. Podemos também ter processos que estão implementados em tecnologias diversificadas, recorrendo até a *software* proprietário.
Por outro lado, podemos também ter processos cuja função é realizar adaptações, mas que são mais específicos, com poucos requisitos e de pequena dimensão.
No fundo existe uma grande diversidade e disponibilidade para utilizar mecanismos que

produzem como resultado dos seus processamentos adaptações, sendo difícil apontar características comuns. Isto leva-nos a considerar que os mecanismos a que uma solução de adaptação em *sites web* recorra, poderão ter diferentes perfis e poderão estar distribuídos por diversas localizações. Podemos pensar em colocar este mecanismos como parte integrante dos gestores de conteúdos ou alternativamente separados. O facto de integrar estes processos com o gestor de conteúdos implica fazer uma implementação dedicada limitando as possibilidades de expansão. Para além do facto de que o gestor de conteúdos deixaria de se preocupar apenas com os conteúdos para se começar a preocupar com os mecanismos de adaptação.

1.1.3 Aplicação de transformações

A aplicação de transformações serve fundamentalmente para reproduzir as adaptações feitas nas páginas que o utilizador está a visualizar. Quando uma adaptação é feita é necessário transmiti-la ao utilizador para que possa usufruir da mesma.
O conteúdo e anotações da página conferem-lhe uma estrutura e formatação, e é sobre eles que as transformações vão incidir.
As operações a realizar podem ser várias, passando pela inserção, remoção ou modificação. Cada uma delas podendo ser efectuada sobre o conteúdo ou sobre a estrutura da página. Com as primeiras podemos fornecer conteúdos orientados a um determinado utilizador através da inserção de novos textos, da sua remoção ou alterando-os de forma a complementar as informações existentes. Com as operações sobre a estrutura podemos reformatar a informação existente de forma a adequá-la a um determinado utilizador. Podemos dar destaque a alguns elementos, esconder outros, fazer determinadas operações que de alguma forma transformam a estrutura existente para a adequar ao utilizador.
Para consegui-lo existem duas hipóteses, introduzir estas adaptações de forma estática no próprio gestor de conteúdos, ou de forma dinâmica no próprio *browser*. Quer uma quer outra são soluções aceitáveis e permitem atingir o objectivo de adaptação, contudo a primeira trás limitações para a aplicabilidade da solução. A escolha de realizar as

transformações no gestor de conteúdos, que vão ser depois apresentadas ao utilizador, implica alguma dependência. Ou seja, outros gestores de conteúdos vão requerer novas implementações e novas soluções. Com a solução dinâmica do lado do cliente liberta-se o gestor de conteúdos para a sua função específica e permite-se a aplicabilidade sobre diversas plataformas.

1.1.4 Exemplo de adaptação

Imaginemos uma página *web* que fornece um leitor de notícias RSS (semelhante a um *news aggregator*) e que foi configurado para realizar adaptações. Esta página agrega *feeds* RSS de outros *sites* e organiza-os em função do assunto. Quando um determinado utilizador acede ao *site* pela primeira vez, depara-se com uma página onde lhe são apresentadas as diferentes notícias organizadas por uma determinada ordem definida pelo criador do *site*.

Se introduzirmos um mecanismo de colecção de eventos sobre a página podemos obter informações sobre os utilizadores, nomeadamente, como interagem com as diferentes secções de notícias. Ao consultar a página o utilizador necessita de clicar nas notícias em que estiver interessado para ler o seu texto na íntegra, se esta acção for registada, indicando que o utilizador leu uma notícia duma determinada secção, podemos interpretá-la como interesse. Se juntarmos os vários registos gerados pela utilização do *site*, ao fim de alguns minutos temos um registo de todas as acções feitas pelo utilizador desde o início da sua navegação e quais os diferentes temas que lhe suscitaram interesse. Realizando uma operação simples de contagem sobre os dados, obtemos uma lista, que pode ser ordenada por ordem decrescente, das secções mais vistas, isto é, secções com mais interesse para o utilizador.

Esta contagem ordenada pode ser usada, para alterar a forma como as secções são apresentadas a esse utilizador. Ao aceder novamente à página as secções aparecerão pela ordem de maior interesse, as mais vistas no topo da página e as menos vistas no fundo da página. Esta adaptação simples baseia-se apenas na contagem de interacção com a página, mas permite oferecer ao utilizador em próximas visitas a informação de

uma maneira mais inteligente e personalizada.

Neste caso o utilizador acaba por ser o próprio motor de adaptação, pois com a sua navegação estará a moldar a página de acordo com os seus interesses.

1.2 Abordagem

Tendo definido as tarefas que uma solução de adaptação *web* requer, podemos definir uma arquitectura que se enquadre nestes requisitos.

Para desenhar uma solução que responda às necessidades que vimos nos pontos anteriores existem três alternativas:

- Criação de um gestor de conteúdos com as características necessárias à realização de adaptações;
- Criação de uma biblioteca de adaptação a ser usada por um gestor de conteúdos;
- Criação de uma *framework* que permita introduzir mecanismos de adaptação.

A primeira hipótese tem como vantagens o facto de se produzir uma ferramenta que nos dê a solução de forma optimizada e especializada. A criação de um gestor de conteúdos com capacidade de realizar adaptações aumentaria a facilidade com que os conteúdos adaptados seriam produzidos, aumentando também a *performance* da construção das respectivas páginas adaptadas. Contudo, esta solução apresenta outros problemas relacionados com os outros dois aspectos que vimos serem necessários à adaptação: a recolha de dados e os mecanismos de adaptação. Apesar de para estes últimos a limitação não ser tão óbvia, é verdade que o próprio gestor pode incorporar mecanismos de adaptação, nos casos em que esses mecanismos são mais complexos e têm maiores requisitos a complexidade do gestor aumenta. Por outro lado, a recolha de dados para adaptação fica um pouco limitada. Os dados a que o gestor tem acesso são os tradicionais *logs* de servidor e o sistema de informação, que poderá ou não existir. Isto não acrescenta muito à qualidade e quantidade de dados que usualmente já se obtêm.

A segunda hipótese elimina um dos problemas permitindo que os mecanismos de adaptação residam fora do gestor. Ainda assim, o problema da recolha de dados mantêm-se e acima de tudo continuamos a ter alguma dependência com o gestor usado, nem que seja pelo facto de a biblioteca usada ter necessariamente de ser suportada por este. Isto faz com que para gestores diferentes tenhamos bibliotecas diferentes.

Com a terceira solução conseguimos responder aos problemas anteriores. Colocando a responsabilidade de cada uma das tarefas num componente autónomo ganhamos independência e portabilidade. Podemos ter um componente responsável por realizar transformações, independentemente do gestor de conteúdos usado, podemos ter uma solução para recolha de dados que pode estar em qualquer ponto da *framework*, nomeadamente, e mais interessante, junto do utilizador. Podemos ainda ter componentes especializados na criação de adaptações e sem a necessidade de serem implementados especificamente para uma determinada aplicação. O último componente que é necessário acrescentar é um ponto de ligação entre todos os componentes formando assim uma solução distribuída.

Esta solução apresenta alguns problemas, como a perda de alguma especialização ou a perda de alguma *performance* devido à diversidade de suporte, mas é também a única que permite responder a todos as tarefas de adaptação de forma consistente.

1.2.1 Objectivos de desenho

Partindo de uma abordagem que define uma ***framework*** como solução para introduzir capacidades de adaptação em *sites*, fazendo a ponte entre o cliente (*browser*) e as ferramentas usadas para produzir adaptações, definimos um conjunto de objectivos de desenho:

Independência das tecnologias *web* A *framework* dever ser capaz de ser usada por diferentes tecnologias de forma a se adaptar às principais tecnologias de formatação *web* (HTML, Flash, Applets Java). A aplicação da *framework* em diversos *sites*, existentes ou a desenvolver, deve ser possível sem que seja neces-

sário uma re-implementação ou se criem limitações.

Abstracção das operações de adaptação Ao ser um meio para a adaptação a *framework* deve abstrair as operações de adaptação, isto é, não necessita de saber nada sobre os processos de adaptação. Apenas deve ser capaz de fornecer essas adaptações, para isso contribui a escolha de uma linguagem de comunicação que seja capaz de representar e transmitir os dados de adaptação.

Expansibilidade dos módulos de adaptação A *framework* deve ser capaz de suportar novos módulos de adaptação, isto é, deve ser possível adicionar, remover ou alterar os métodos que produzem adaptações sem que haja perda de funcionalidades ou que se entre num estado de inutilização.

Minimização do impacto causado pela solução A inclusão de uma solução como esta num *site* não deve, do ponto de vista do utilizador final, penalizar a sua experiência de navegação, bem como impedir que o *site* funcione sem esta.

1.2.2 Arquitectura geral

A arquitectura distribuída esquematizada na figura 1.1 divide-se em quatro componentes distintos: **Clientes**, ***Broker***, **Conteúdo** e **Adaptadores**, sendo o *Broker* o elo de ligação entre eles. O módulo Conteúdo corresponde a um elemento externo, e que existe sempre mesmo numa utilização que não recorra à utilização da *framework*. É o repositório dos conteúdos, podendo ser um sistema de ficheiros, um gestor de conteúdos ou outro qualquer mecanismo de fornecimento de conteúdos.

O módulo Cliente é na realidade uma extensão a incluir na página visualizada (conteúdos), fornecendo os métodos necessários a realizar uma adaptação da página e também a recolha de dados de adaptação. Portanto, quando nos referimos ao Cliente referimo-nos a essa extensão, dado que existe um *browser* e uma página que existem e existirão sempre, mesmo que a *framework* não seja usada.

O *Broker* é o responsável pela interligação dos componentes, agindo como um *proxy*

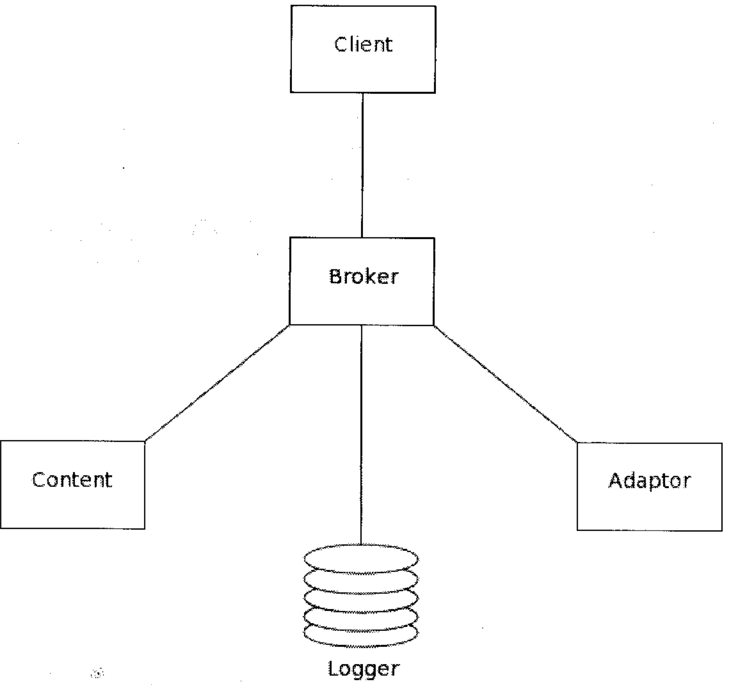


Figura 1.1: Proposta para arquitectura geral.

e servindo de *gateway* a toda a comunicação. É também responsável pelo registo de determinadas informações relevantes para o processo de adaptação. O *Broker* fornece um ponto de acesso único e compatível com os requisitos de segurança, de validar as mensagens de adaptação, e de registar informação necessária a futuras adaptações.
Um Adaptador terá o trabalho de processar as informações passadas pelas mensagens, podendo analisar os registos que o *Broker* criou e devolver dados através de mensagens de resposta para o Cliente.
Os Adaptadores podem estar em servidores ou locais diferentes. O facto de existir esta possibilidade dá à *framework* uma capacidade ainda maior de portabilidade e diversificação, podendo coexistir em sistemas e localizações diferentes. Permite também que seja possível, a quem desenvolve adaptadores, escolher a forma, a linguagem e o método que mais lhe convém para criar os seus adaptadores.
Este factor é particularmente importante na fase de desenvolvimento, pois dá aos investigadores a possibilidade de experimentar o sistema com grande facilidade no seu ambiente favorito e até eventualmente aproveitar outros trabalhos já existentes.

A comunicação entre os diferentes componentes é baseada em **mensagens**. Estas levam do Cliente ao *Broker* toda a informação necessária aos pedidos de adaptação e às notificações de eventos, que por sua vez retira delas a informação necessária aos registos da base de dados e as passa aos Adaptadores para que possam processar a informação. Os Adaptadores depois de trabalharem os dados, constroem mensagens de resposta baseadas nos mesmos formatos, que são devolvidas ao *Broker* e novamente ao Cliente.

A definição de um esquema para as mensagens permite abstrair o que é adaptação, passando a estar subjacente nos dados e na estrutura que compõem as mensagens, libertando a *framework* da necessidade de compreender o que é a adaptação. Isto acentua ainda mais o papel de transportador de adaptações que a *framework* tem.

Para além da vertente principal de passagem de informação entre componentes, as mensagens possuem ainda outro papel no sistema, o de administração e o de reportar erros. Com a definição de mensagens criada, não só é passada informação da qual o sistema nada sabe, como é possível administrar o sistema ou ainda receber informação de controle, como erros que ocorrem em determinado ponto. Esta funcionalidade tornou-se importante pelas características do sistema, que ao poder estar distribuído e ser controlado por diferentes utilizadores, necessitava de poder dar alguma informação de retorno para ser possível compreender alguns acontecimentos, nomeadamente erros.

A possibilidade de administração prende-se com o facto de ser necessário realizar a qualquer momento operações sobre alguns componentes, mais concretamente sobre os Adaptadores, que podem estar em máquinas remotas.

1.2.3 Preparação de *sites* para adaptação

Um ponto essencial e que não pode ser dissociado da utilização da *framework* é a preparação dos *sites* para a adaptação, seja alterando *sites* já existentes para que se interliguem com a *framework*, seja prevendo no desenho de novos *sites* pontos de ligação. Em qualquer um dos casos algum tipo de intervenção é necessária.

É importante referir que com este modelo não há restrições às tecnologias a usar

na implementação, isto é, podem ser usadas ferramentas diversas na elaboração de páginas: Flash, HTML, JavaScript, ASP entre outras. Apenas existem requisitos como a necessidade de comunicar assincronamente sem forçar a actualização da interface *web*. Desde que sejam respeitados estes requisitos, qualquer tecnologia está habilitada a utilizar a *framework*. No entanto, a facilidade de implementação e a adequação de cada uma delas fica ao critério dos criadores. Isto é importante devido à heterogeneidade das ferramentas utilizadas para a criação de *sites*. Actualmente existem diversas tecnologias (AJAX[22][15], Flash, VBScript e Applets Java) usadas no desenvolvimento de *sites*, o que dificulta e torna inviável, a obrigatoriedade de utilização da ferramenta A ou B. Isto inclusive limitaria logo à partida o sucesso da *framework*, além de que uma arquitectura deste género, como iremos ver mais à frente, tem precisamente como objectivos a liberdade de utilização de ferramentas e especificações.

1.3 Cenário de utilização

Retomando o exemplo da página com as *feeds RSS*, podemos ilustrar como este trabalho seria aplicado. O primeiro passo seria a colocação num servidor aplicacional do *Broker* e a configuração de um acesso a algum tipo de base de dados para o registo de informações. A seguir deve ser configurado um ou mais Adaptadores a que o *Broker* terá acesso e para onde enviará os pedidos de adaptação. Este Adaptador poderá estar no mesmo servidor local ou remoto que o *Broker*. Poderiam também ser Adaptadores a serem já utilizados por outros sistemas de adaptação em produção.

Por último e para que o cliente possa ter acesso às funções que permitem realizar adaptações e notificações, o *webmaster* colocaria *online* uma versão do *site* modificada de forma a que a extensão ao cliente, responsável por enviar mensagens com dados específicos para o *Broker*, ficasse disponível.

Com estes passos a *framework* estaria pronta a funcionar, os acessos dos utilizadores começariam a ser registados criando um registo das várias acções sobre a página, o

que a cada pedido novo irá produzir resultados de adaptação diferentes.

Com esta configuração temos um *site* com capacidades adaptativas. O Cliente compila informação sobre a página e envia-a ao *Broker*, que por sua vez envia essa informação a um determinado Adaptador para realizar a adaptação. Após o processo estar concluído uma mensagem de resposta é enviada para o *Broker* que a faz chegar ao Cliente, onde este insere na página o resultado da adaptação.

A par disto o Cliente envia ainda informação ao *Broker* dados relativos às interacções do utilizador com a página para que este as registe.

O resultado final é uma página com *feeds* RSS que se adapta a cada utilizador mostrando de cada vez que o utilizador acede à página uma configuração de acordo com a sua interacção, seja colocando as *feeds* por ordem da sua preferência, seja removendo *feeds* que não têm relevância. Ou seja, dando mais destaque usando alterações de estilo sobre as *feeds* que o utilizador mostrou mais interesse.

1.4 Estrutura da tese e convenções

Nos restantes capítulos desta tese vamos ver os vários passos que levaram à criação da infra-estrutura de adatptação *web*. Vamos começar por ver no capítulo 2 as diferentes tecnologias relacionadas com este trabalho. No capítulo 3 vamos abordar a arquitectura escolhida para a *framework*, focando os aspectos funcionais de cada componente, e a estrutura das mensagens. No capítulo 4 vamos ver em detalhe a implementação da *framework* focando as escolhas técnicas para a implementação. No capítulo 5 vamos ver os resultados obtidos com um protótipo da *framework* em termos de eficiência e robustez. Por fim, no capítulo 6, vamos reflectir sobre o trabalho feito e apontar metas para o trabalho futuro a desenvolver.

Na escrita deste trabalho foram feitas algumas convenções sobre o texto, nomeadamente, para representação de conceitos escolheu-se o **negrito**, para transcrições de dados de programa ou código a fonte `typewriter` e para nomes ou designações em alguma língua que não o português o *itálico*. Nas imagens ao longo deste trabalho todos

os textos de suporte usados estão em inglês por uma questão de uniformização com o código produzido, siglas e denominações usuais de tecnologia que são habitualmente usadas.

Capítulo 2

Estado da Arte

Ao longo da história os modelos usados nas tecnologias de informação (TI) foram evoluindo. Começando com aplicações ***stand-alone*** que mais tarde deram lugar aos modelos **cliente-servidor** e mais recentemente aos modelos **distribuídos**.
Neste capítulo vamos falar um pouco sobre os modelos, especificações, *standards* e tecnologias existentes que contribuiram de alguma maneira para este trabalho.

2.1 Arquitectura orientada a serviços

A arquitectura orientada a serviços ou SOA - Service Oriented Architecture[8], é um conjunto de indicações ou directivas, que servem como linhas orientadoras para o desenho de aplicações, modelos de negócio ou qualquer outro contexto que se veja reflectido nas indicações introduzidas por esta.
A arquitectura SOA assenta em alguns princípios básicos:

Emparelhamento flexível Os serviços mantêm uma relação que minimiza as dependências, bastando apenas que tenham consciência da sua existência.

Serviços contratuais Os serviços acordam mutuamente num tipo de comunicação, sendo esta definida por uma ou mais descrições dos serviços.

Autonomia Os serviços têm controlo sobre a sua lógica de operação

Abstracção Para além daquilo que é dado a conhecer e é acordado entre serviços, estes escondem a sua lógica de funcionamento do exterior.

Reutilização Divide-se em vários serviços as diferentes lógicas de operação de modo a poderem ser reutilizados.

Composição Vários serviços podem aglomerar-se para formar serviços compostos.

Ausência de estado Os serviços retêm o mínimo de informação que é específica a uma actividade.

Capacidade de descoberta Os serviços devem ser desenhados de forma descritiva para que possam ser descobertos e utilizados usando apenas ferramentas de descoberta.

Este princípios definem uma arquitectura SOA e bastam para delinear estratégias para o desenho de qualquer aplicação que queira retirar proveitos destes conceitos. No entanto, como a evolução tecnológica e do mercado obrigou a algumas adaptações um novo conceito surgiu, o SOA comtemporâneo[1]. Este novo modelo é apenas a extensão do modelo base com a inclusão de necessidades de segurança, fiabilidade, qualidade de serviço e outras características comuns nas aplicações e negócios actuais. Mesmo com estas adições os princípios base da SOA mantêm-se, servindo de base para o desenho e implementação de aplicações orientadas a serviços.
Para colocar em prática estas linhas orientadores existem algumas tecnologias que pelas suas características ganharam notoriedade e contribuíram para a massificação do modelo SOA.

[1]Esta conceito está bem descrito no livro *Service Oriented Architecture*[8]

2.2 XML

O XML[25][36] é uma especificação de uma linguagem para representação de dados e documentos. O modo como o XML está definido permite criar documentos estruturados em que os dados possuem contexto e algum controle sobre os mesmos. O XML é hoje uma das tecnologias mais utilizadas, principalmente em aplicações *web.*

Com as suas características descritivas o XML deu às aplicações algo mais do que o transporte de dados, permitiu-lhes perceber os dados e extrair informações relativas a estes de uma forma automática, pois os documentos não têm limites no que toca á quantidade de informação que a eles pode estar associada.

A sua forma estruturada permite realizar tarefas de *parsing* de forma eficiente e rápida. Isto permite que as aplicações o possam usar para guardar e gerir a informação que por elas circula. Com o XML as aplicações podem aceder facilmente a valores contidos nos documentos, criar novos documentos e com a sua estrutura definir contextos. Uma das vantagens é ser muito descritivo, o que dá aos documentos um carácter auto-explicativo.

Associado ao XML estão as API's a estes documentos, de que é exemplo a DOM[6][5]. O Document Object Model é uma API que através de métodos e constantes permite aceder e ler conteúdos de documentos. O que esta API faz, é representar em memória um documento através duma árvore pela qual é possível navegar. Existem funções para navegar entre os diversos nós, para a frente, para trás ou para os lados, a estas navegações correspondem o nó pai (*parentNode*), os nós filhos (*childNodes*) e os nós irmãos (*siblingNodes*). Permite também aceder aleatoriamente a um determinado nó da árvore e fazer a validação de documentos recorrendo a documentos próprios para esse efeito.

2.2.1 Validação

O Document Type Definition (DTD), é um documento com uma linguagem própria que define explicitamente a estrutura de um documento XML. A vantagem da

utilização do DTD é a de permitir perceber rapidamente a sua estrutura devido à sua simplicidade. No DTD são indicados os nomes dos elementos e os respectivos atributos. Para cada elemento e atributo pode ser indicado o seu tipo. Estas regras permitem controlar o modo como são construídos os documentos baseados nesta definição, servindo de guia para novos documentos.
O DTD tem algumas limitações, os seus tipos são muito básicos e não permitem responder a configurações mais complexas de documentos. Para responder a isto foi criado o XML Schema Definition[38][7] (XSD).
O XSD é um documento XML que tal como o DTD tem uma estrutura muito bem definida e que serve para indicar como um outro documento XML se pode construir. O XSD tem uma configuração mais complexa e permite indicar tipos simples e complexos e inclusive criar novos tipos.
Apesar de o DTD ter sido a primeira solução para a verificação de documentos as suas limitações fizeram com que o XSD rapidamente se tornasse dominante, actualmente o XSD é usado para fazer a validação dos documentos XML construídos mediante as regras de um determinado XSD. As aplicações que fizerem uso de documentos XML e que tenham acesso ao XSD que os define, podem usá-lo para realizar a verificação dos documentos.

2.2.2 Transformação

Uma das funcionalidades a que muitas vezes se recorre, quando estamos a lidar com documentos XML, é a sua transformação. Isto é, a produção de novos documentos através da manipulação dos documentos XML existentes.
Para isso, é normalmente usado uma classe de linguagens denominadas XSL - XML Stylesheet Language. Desta classe fazem parte as linguagens XSLt e XPath que servem respectivamente para:

- transformação de documentos noutros documentos;
- navegação na árvore do documento, obtenção de valores e realização de testes

sobre os elementos da árvore.

Com o XPath podemos obter de um documento os valores de um determinado nó, encontrar um caminho na árvore, ou até realizar operações de teste sobre os valores contidos no documento. Aliado ao XSLt, pode ser usado para construir novos documentos através da utilização das transformações que o XSLt oferece e as operações que o XPath fornece. Um exemplo comum desta utilização é a conversão de documentos XML, que contêm por exemplo a definição de uma página HTML, num verdadeiro documento HTML.

2.3 *Web Services*

Os *web services* são um conjunto de especificações que têm como base a comunicação entre pontos dando apenas a conhecer o seu interface público. Os serviços conhecem-se mas não conhecem os seus detalhes, sendo apenas capazes de comunicar entre si dados de acordo com um protocolo previamente estipulado. Para que tudo se articule, a comunicação entre serviços é assegurada por mensagens, mensagens essas em XML que permitem a circulação de dados de forma autónoma e sem dependências. Nos *web services* as mensagens são autónomas e autocontroladas. Quando uma mensagem circula no sistema é um elemento único sem qualquer ligação ao serviço de onde partiu e com as suas próprias estruturas de controle.

Isto torna-os independentes da tecnologia, pois as suas implementações são desconhecidas das aplicações. Ao ter apenas a comunicação como ponto de ligação entre serviços, estes podem ser reutilizados e compostos com outros serviços. Os *web services* criam camadas de abstracção, as aplicações conhecem as suas funcionalidades mas nada sabem sobre a sua implementação. Os *web services* dispõem ainda de protocolos que permitem aos serviços descobrirem outros serviços e comunicar com eles.

Estas características fazem com que os *web services* sejam a ferramenta ideal para a implementação de arquitecturas SOA, aliás para muita da indústria TI é um dado

adquirido que ao falar de SOA estamos obrigatoriamente a falar de *web services.*

2.3.1 SOAP

O SOAP[28] - Simple Object Access Protocol - que inicialmente permitia comunicação de mensagens e emular comunicações do tipo RPC, foi sendo transformado e actualmente deixou de ser um acrónimo para ser um *standard* próprio. Este protocolo é usado pelos *web services* e dá às mensagens as características necessárias para circularem numa arquitectura SOA.

O SOAP é um protocolo de comunicação que permite fazer a ponte entre os interfaces de um serviço definidos num WSDL, através da construção e envio de mensagens. Cada mensagem é um documento XML com uma estrutura bem definida e dividida em três elementos: envelope, cabeçalho e corpo. O envelope corresponde à raiz do documento e funciona como contentor para o cabeçalho e o corpo.

O cabeçalho tem como função passar informação de controle e pode também servir como meio para estabelecer novas extensões, introduzindo novas funcionalidades no protocolo. Este elemento não é obrigatório na construção da mensagem SOAP.

O corpo por sua vez é obrigatório e serve como contentor para mensagens XML bem formatadas. Enquanto os cabeçalhos de uma mensagem têm um papel mais activo durante a transmissão de mensagens SOAP, pois podem ser processados pelas aplicações para obter informações, o corpo é normalmente ignorado sendo apenas utilizado num momento final para receber o seu conteúdo.

2.3.2 WSDL

O WSDL é uma linguagem para descrição de serviços web - Web Service Definition Language. Nele estão definidas um conjunto de informações que definem um serviço permitindo às aplicações que comuniquem de maneira correcta com ele, identificando o seu endereço físico e o modo como os dados devem ser enviados. O WSDL é constituído

por dois tipos de definições: abstractas e concretas.

As definições abstractas definem as características do interface do *web service* sem que haja compromisso com a tecnologia usada para gerar o *web service*. Isto permite que o serviço possa manter a sua definição mesmo que as tecnologias usadas para o implementar e suportar sejam alteradas. Existem três partes nas definições abstractas: *portType*, *operation* e *message*, cada uma delas contida na anterior. O *portType* define um conjunto de operações (*operation*) que um serviço disponibiliza para receber determinadas mensagens. Cada uma desta operações corresponde a uma tarefa específica do serviço. Como a comunicação dos *web services* se baseia em mensagens, cada operação define um conjunto de mensagens (*messages*) que pode receber e enviar.

As definições concretas fazem a ligação entre as definições abstractas a implementação real do serviço. Para isso, define três partes: *binding*, *port* e *service*. A primeira descreve quais os requisitos de um serviço para estabelecer conexões físicas ou conexões com o serviço. No fundo indica uma tecnologia de transporte a ser usada para a comunicação, como por exemplo o SOAP. Para estabelecer a comunicação é necessário definir um endereço físico (*port*) através do qual são trocadas as mensagens. Por fim a definição *service* é usada para referir um grupo de interfaces relacionados.

Uma particularidade do WSDL é que em conjugação com um XSD, permite definir os documentos XML contidos no corpo das mensagens SOAP.

2.3.3 UDDI

A proliferação dos *web services* levou à necessidade de criar um mecanismo que permitisse identificar os *web services* existentes. O Universal Description Discovery and Integration (UDDI) é uma especificação que permite manter um registo central dos *web services* existentes fornecendo um serviço de registo e pesquisa. Esta solução permite às aplicações e utilizadores conhecerem rapidamente os serviços que estão disponíveis para utilização e as suas especificações. O UDDI permite definir diferentes contextos como registos públicos ou privados.

Esta diferenciação aumenta o leque de possibilidades da utilização dos registos a um

nível global como a Internet ou mais local como o ambiente de uma empresa. Os registos públicos são compostos por servidores dedicados que estão articulados por forma a sincronizarem os seus registos. Os registos privados são servidores locais que servem uma determinada empresa e onde o acesso à informação existente é limitada a utilizadores registados. Esta especificação não tem qualquer relação com o WSDL, mas os apontadores para os interfaces dos serviços podem ser o WSDL que define o serviço.

O UDDI ainda não é *standard* totalmente aceite, mas os seus benefícios são considerados extremamente importantes.

2.3.4 WS-*

A utilização da sigla "WS-*"tornou-se a maneira de referir a segunda geração de especificação de *web services*. Representa as extensões que extendem as definições base dos primeiros *web services*, SOAP e UDDI. A sigla surge do nome dado às extensões que usam na sua nomenclatura o prefixo *WS*.

A vantagem da introdução destas extensões foi o aumento das capacidades associadas à primeira geração de *web services* dando novas possibilidades ao processos computacionais. Algumas extensões fundamentais são:

WS-Adressing Acrescenta um conjunto de elementos no cabeçalho da mensagem relacionados com o seu trajecto: local de origem, destinatário ou acção em caso de falha na entrega.

WS-ReliableMessaging Introduz um conjunto de elementos que controlam o processamento sequencial das mensagens e respectivas notificações de sucesso ou insucesso na entrega. Contribui para a qualidade de serviço.

WS-Policy Estabelece um conjunto de politicas que permitem adicionar uma camada de integridade. Permite aos serviços especificar regras e preferências em relação à segurança, processamento ou conteúdo das mensagens. É composta por outras

extensões WS-*.

WS-MetadataExchange Fornece um método *standard* através do qual podem ser pedidos ou fornecidos documentos de descrição dos serviços. Contribuí para a interoperabilidade e qualidade de serviço.

WS-Security Introduz extensões que permitem adicionar segurança ao nível das mensagens. Protegem o conteúdo das mensagens durante o transporte e durante o processamento por intermediários, permitindo por exemplo autenticação. É normal usar outras especificações no contexto da segurança como: XML-Encryption ou XML-Signature.

2.4 Ajax

O Ajax - Assynchronous JavaScript and XML[21], é uma designação comum para um conjunto de tecnologias que permitem a realização de pedidos HTTP e alterações na estrutura e conteúdos de páginas HTML sem necessidade de as redesenhar na totalidade.

Para realizar os pedidos o Ajax usa um objecto JavaScript, o `XMLHttpRequest`, que permite realizar pedidos de forma síncrona ou assíncrona. Os pedidos assíncronos tem a particularidade de permitir que o processamento do código onde este objecto está não seja bloqueado pelo aguardar da resposta. Isto faz com que uma página possa ser carregada na totalidade e simultaneamente realize pedidos HTTP, sem que o utilizador se aperceba. Com este método, fica facilitada a troca de informações que sirvam de complemento a um *site*.

Um dos exemplos de utilização desta tecnologia é o Google Suggest[11], onde é apresentado ao utilizador, à medida que insere o texto de pesquisa, uma lista de possíveis pesquisas e respectivos número de resultados. Isto é feito transparentemente e o utilizador não nota qualquer tipo de alteração na página nem na sua navegação. O que acontece é que quando um utilizador começa a escrever um texto na *searchbox* é feito

um pedido HTTP extra, pedindo uma lista de possíveis palavras ou frases a colocar nessa *searchbox*, mas que comecem pela letra ou prefixo que o utilizador escreveu. Desta maneira, o utilizador é complementado com informação sem haver necessidade de fazer um pedido de uma nova página, ou seja, o utilizador nem se apercebe que foi realizado um ou mais pedidos novos, pois a página em que se encontra não foi redesenhada.
A utilização do Ajax traz algumas vantagens adicionais, em particular a diminuição da largura de banda usada. Como os pedidos feitos com recurso ao Ajax transportam apenas a informação necessária contribuem para um descongestionamento das vias de comunicação e dos servidores.

2.5 *Browsers* e suporte Ajax

Uma das dificuldades de utilizar Ajax, actualmente, é o suporte dado pelos *browsers* ao JavaScript. Os diferentes *browsers* têm implementações próprias das especificações e introduzem diferenças de comportamento no modo como lidam com o código Ajax. Uma das principais diferenças está no suporte do objecto `XMLHttpRequest` que só recentemente começou a ser suportado nativamente. Este tipo de problemas vai sendo resolvido pela introdução de bilbiotecas JavaScript adicionais, mas mesmo estas apresentam algumas limitações.
Outra dificuldade está relacionada com a DOM. Muitos *browsers* fazem representações da estrutura dos documentos com pequenas diferenças, seja por opção ou simplesmente por falhas da implementação. É este tipo de problemas que explica as diferenças que podemos encontrar numa mesma página quando vista com *browsers* diferentes. Isto reflecte-se na dificuldade de implementar uma aplicação que tenha comportamento idêntico em *browsers* diferentes. Como é óbvio não queremos limitar a escolha do *browser* por parte do utilizador, por isso é necessário criar mecanismos que permitam uma utilização diversificada. Isto é conseguido normalmente através da criação de código específico e que é activado consoante o *browser* que estiver a ser usado.

Apesar de se notar uma tendência para a convergência no suporte que cada *browser* fornece, mas é impossível afirmar que estas diferenças deixaram de exstir.

2.6 Tecnologia

Para este trabalho e tendo em conta a abordagem que definimos escolhemos um conjunto de tecnologias que permitiram implementar a nossa solução. As tecnologias escolhidas permitiram a utilização dos *standards* e especificações que vimos nos pontos anteriores, com a particularidade de todos serem livres.

O **Tomcat**[30] é um servidor aplicacional que serve *JSP* e *Servlets*. Este servidor desenvolvido pela Apache Software Foundation[2] permite ainda extender as suas funcionalidades através da inclusão de extensões que estão disponíveis na *web*. O Tomcat é frequentemente usada para aplicações implementadas em Java.

O **Axis**[3] é um motor SOAP desenvolvido pela Apache Software Foundation[2] que permite extender o Tomcat e implementar em Java, clientes, servidores ou *gateways* SOAP. A sua API[4] permite aos programas escritos em Java criarem serviços, estabelecer as ligações e enviar os dados de uma forma simples e autónoma.

Como linguagem de implementação foi escolhido o Java. O Java é uma linguagem orientada a objectos (OO) com uma API vastíssima e com inúmeras classes para a implementação de *web services*, manipulação de XML e comunicações *web*.

Uma das classes que neste trabalho teve maior importância foi o pacote `javax.xml` que implementa as funcionalidades da especificação XML.

Estas foram as tecnologias com maior relevância na produção deste trabalho.

Capítulo 3

Arquitectura do SOM

A arquitectura da *framework* está directamente relacionada com as funcionalidades que se pretendem, nomeadamente a necessidade de extrair dados do cliente, realizar adaptações com eles e introduzir alterações no cliente como resultado dessa adaptação. Isto implica que haja capacidade no cliente para processar informação, capacidade de armazenamento e processos de adaptação. Atendendo a estas tarefas e ao objectivo de utilização por várias aplicações é necessário criar independência entre os processos por forma a não fazer depender das aplicações a arquitectura. A divisão em componentes específicos na sua função permite criar uma relação genérica entre todos. Um componente lida com a necessidade dos processos a nível do cliente: extracção de dados, aquisição de eventos e transformações. Outro componente lida com o registo da informação obtida por forma a ser reutilizada e outro componente fica responsável pelos processo de adaptação, isto é, os algoritmos que realmente processam os dados e geram dados adaptados.

Estes componentes ligados entre si permitem responder às necessidades para a introdução de adaptação em *sites web*, mas para que a comunicação entre eles resulte e se processe de forma segura há um elemento mediador que canaliza toda a comunicação, como se pode observar no diagrama da figura 3.1. O facto de estarmos a lidar com uma aplicação *web* limita de alguma maneira os acessos que podemos fazer a endereços para

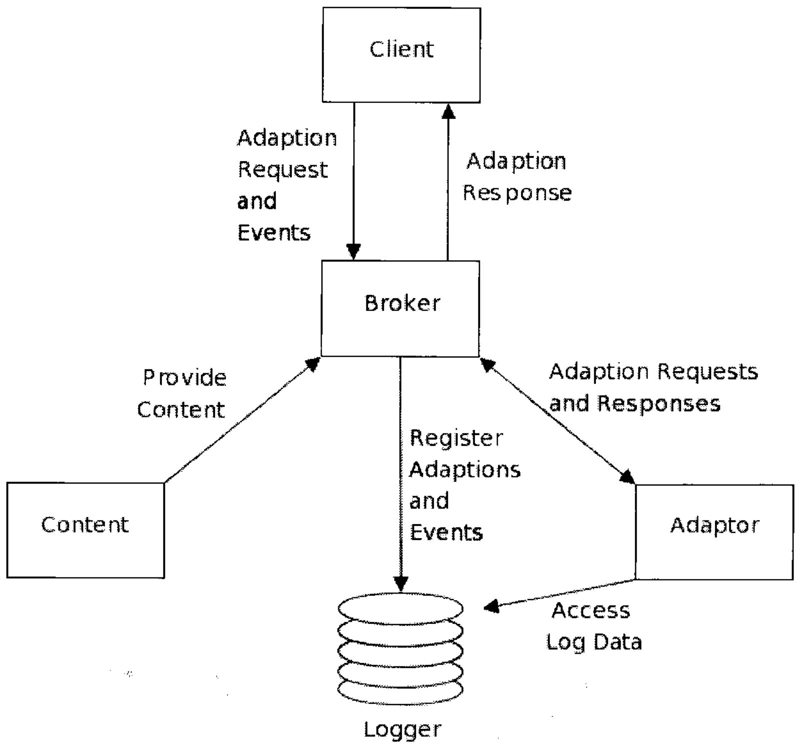


Figura 3.1: Arquitectura geral de uma solução que faça uso da *framework*.

estabelecer comunicações. Com esta unidade central essa dificuldade fica resolvida aumentando a segurança e ganhando um ponto de controle.

Assim, podemos identificar como componentes com um papel fundamental e activo na arquitectura: o **Cliente**, o ***Broker*** e o **Adaptador**. Circulando por todos estes componentes e suportadas pelos diferentes protocolos de comunicação estão as mensagens. As mensagens correspondem a pedidos e respostas de adaptação e a eventos que são gerados pelo utilizador. Uma mensagem permite dar contexto, transmitir dados, comandos ou mesmo erros. As mensagens têm uma estrutura bem definida, genérica e detalhada ao mesmo tempo. São genéricas, pois o seu desenho permite diversas funções, como veremos, mas ao mesmo tempo suficientemente detalhadas ao ponto de permitirem fornecer dados com grande precisão. As mensagens possuem ainda uma sintaxe que fornece contexto, para que cada componente possa perceber ou fornecer determinadas informações sobre as diferentes comunicações.

3.1 Mensagens

Como as mensagens necessitam de representar a adaptação, é necessário que a sua estrutura seja verbosa ao ponto de poder transmitir conceitos e contexto nos seus dados. Assim, a escolha do XML para construir as mensagens deveu-se às suas características de representação e estruturação de dados e documentos. A flexibilidade que o XML permite, aliado às capacidades de validação permitem dar ás mensagens características necessárias para transportar os dados por adaptar e adaptados, de forma a que Adaptadores sejam capazes de os compreender e os Clientes sejam capazes de os ler e representar. O XML é a linguagem frequentemente usada em aplicações que tenham requisitos de estrutura e controle sobre os dados que se querem transmitir ou processar. No caso da *framework* é exactamente isso que se pretende, troca de informação entre componentes com uma grande qualidade de forma altamente estruturada.

As mensagens são um ficheiro XML estruturado de forma a representar informação. A definição da estrutura das mensagens teve por base as necessidades, quer do Cliente quer dos Adaptadores, ou seja, é representada a informação para adaptação e a informação resultante da adaptação. Para além destas, e num segundo plano, apesar de não haver uma alteração à estrutura base, as mensagens representam ainda comandos e erros. Aliás, pode-se dizer que a menos de alguma informação de controle nas mensagens, não existem diferenças entre umas e outras. Se observarmos as mensagens que circulam num determinado instante iremos ver que não existem diferenças de fundo na sua estrutura.

As mensagens são criadas dinamicamente sempre por Clientes e Adaptadores, o *Broker*, apesar de ter acesso a elas e inclusive poder receber comandos através das mensagens, não assume o mesmo papel no que diz respeito à sua construção, onde Cliente e Adaptador estão dependentes destas para o seu funcionamento.

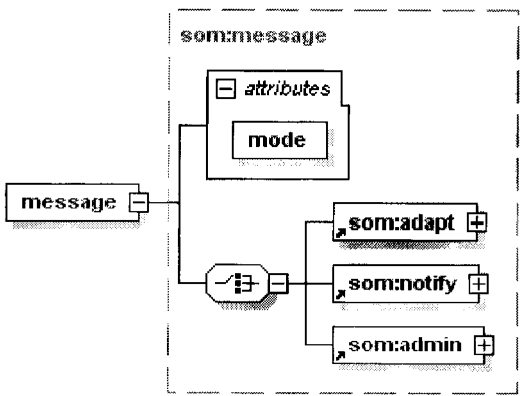


Figura 3.2: Elemento message.

3.1.1 Estrutura

A estrutura das mensagens pode ser facilmente explicada com recurso a uma representação gráfica[1]. Usando uma estrutura simples, semelhante a uma árvore podemos perceber como são definidos cada um dos elementos que compõem as mensagens que circularão na *framework*. Vejamos então, recorrendo a esses esquemas, como se define uma mensagem e quais os contextos da sua utilização.

Como as mensagens são construídas em XML, todas começam por um elemento de topo `message` (Fig. 3.2) que vai ter no seu interior a mensagem propriamente dita. São três os tipos de mensagens que este elemento pode conter:

- `adapt` - mensagens de pedidos e respostas de adaptação do Cliente para um determinado Adaptador e vice-versa;
- `notify` - mensagens de notificação de eventos do Cliente para o *Broker*;
- `admin` - mensagens de administração do sistema.

Na definição do elemento `message`, existe, para além da indicação dos seus possíveis sub-elementos, a indicação de um atributo de uso obrigatório para especificar o modo

[1]Esta representação foi feita recorrendo à versão de demonstração do Altova XMLSpy[37] e usando um XML Schema Definition.

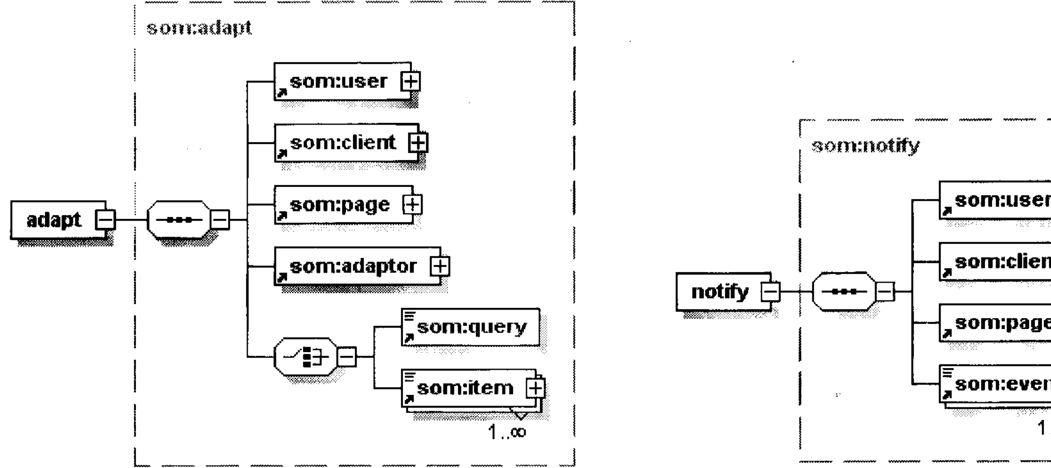


Figura 3.3: Elemento `adapt`.

Figura 3.4: Elemento `notify`.

da mensagem. Este modo é importante para se perceber qual o contexto da mensagem, dado que o mesmo tipo de mensagens é usado para pedidos e respostas (*request e reply* respectivamente). Esta é a maneira, por exemplo, de distinguir uma mensagem que vem de um Cliente e de um Adaptador. As mensagens que têm origem nos Clientes terão o atributo `mode` igual a `request` e as de repostas, vindas dos adaptadores igual a `reply`.

Este elemento de topo pode conter um de três tipos de elementos, cada um deles tendo sub-elementos comuns, por isso vejamos primeiramente cada um deles.

As mensagens de adaptação , que são as mensagens que saem do Cliente pedindo adaptações e são as mensagens que chegam dos Adaptadores com conteúdos ou informações adaptadas, podem-se definir da forma que se vê na figura 3.3. Cada mensagem de adaptação deverá conter os elementos: `user`, `client`, `page`, `adaptor`, `query` ou `item`.

Como se pode ver na figura 3.3, pela presença do símbolo interruptor, os últimos dois elementos são exclusivos. O elemento `item` poderá ocorrer mais do que uma vez.

No tipo seguinte de mensagens, as de notificação, que são as que saem exclusivamente do Cliente para o *Broker*, a definição é a da figura 3.4. As mensagens de notificação contêm os elementos: `user`, `client`, `page` e `event`.

É importante salientar que nesta definição há dois elementos em comum com a defini-

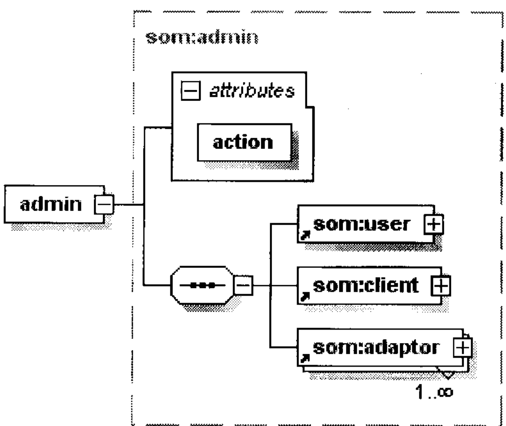


Figura 3.5: Elemento admin.

Valor	Descrição
list	usado para obter uma listagem dos adaptadores existentes
register	usado para registar um novo adaptador
status	usado bara obter o estado de um ou vários adaptadores
enable	usado para activar um adaptador
disable	usado para desactivar um adaptador

Tabela 3.1: Valores para o atributo action do elemento admin

ção anterior e dois novos. De notar também, que o elemento event poderá ocorrer mais do que uma vez. Isto permite que sejam agrupados mais do que um evento diminuindo a necessidade de comunicações com o *Broker* para fazer o respectivo registo.

Por fim, temos as mensagens de administração que apresentam a definição da figura 3.5. Estas mensagens são compostas pelos elementos: user, client e adaptor.

Este elemento, admin, que compõe as mensagens de administração apresenta diferenças relativamente aos dois elementos que vimos antes. Nomeadamente, existe um atributo associado, o atributo action que pode conter um dos valores da tabela 3.1, indicando a acção que se deseja realizar. As mensagens com elemento admin podem ser usadas para funções de administração quer no *Broker* quer no Adaptador.

Estas três definições compõem as mensagens tipo que circulam na *framework*, todas

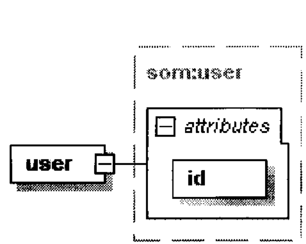


Figura 3.6: Elemento `user`.

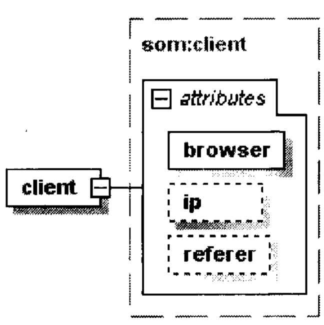


Figura 3.7: Elemento `client`.

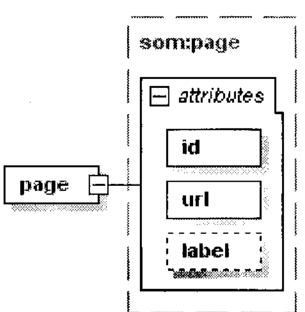


Figura 3.8: Elemento `page`.

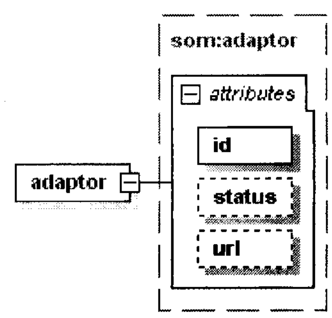


Figura 3.9: Elemento `adaptor`.

elas constituídas por sub-elementos que contêm os dados propriamente ditos.

Cada um destes sub-elementos mesmo quando incluídos em dois tipos diferentes de mensagens representam os mesmos dados. Isto é, o elemento `user` define-se do mesmo modo quando incluído numa mensagem de adaptação, `adapt`, ou quando incluído numa mensagem de notificação, `notify`. O elemento `user` (Fig. 3.6) é usado para guardar a identificação do utilizador do cliente em questão, para isso é usado um atributo `id`.

O elemento `client` (Fig. 3.7) guarda a informação sobre o cliente, o *browser* que está a usar, o seu IP e página de origem. Estes dados são guardados nos atributos `browser`, `ip` e `referer` respectivamente, os dois últimos opcionais.

O elemento `page` (Fig. 3.8) guarda informações sobre a página visitada, nomeadamente, a sua identificação única em `id`, o URL da página em `url` e uma string associada em `label` que é opcional.

O elemento `adaptor` (Fig. 3.9) guarda as informações relativas ao Adaptador a usar

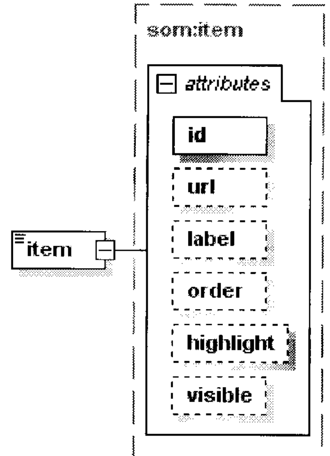


Figura 3.10: Elemento item.

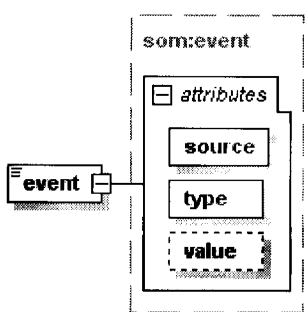


Figura 3.11: Elemento **event**.

para uma adaptação, podendo também ser usado para comunicar o estado e o endereço de um Adaptador nas mensagens de administração.

Os respectivos atributos são o `id`, **`status`** e `url`, sendo que os dois últimos são opcionais, pois a sua utilização depende do objectivo da mensagem.

Existe também um elemento `query` que é usado para guardar dados que irão ser passados ao Adaptador. Esta informação da qual falaremos mais à frente, é definida previamente no cliente. Este elemento é extremamente simples funcionando exclusivamente como um contentor.

O elemento `item` (Fig. 3.10) guarda uma série de informações relativas a elementos que são passados aos Adaptadores e que chegam dos Adaptadores. Com a excepção do atributo `id`, que guarda a identificação de cada elemento, todos os restantes são opcionais. Os restantes atributos: `url`, `label`, `order`, **`highlight`** e **`visible`** guardam valores sobre os elementos, como a sua ordem, destaque ou visibilidade. Este elemento pode ainda conter texto adicional.

Por fim, o elemento `event` (Fig. 3.11) tem como objectivo guardar toda a informação relativa aos eventos gerados no *browser*, guarda o local onde o evento foi gerado, o tipo de evento e um valor associado, este último opcional. Este elemento tal como o anterior pode também conter texto adicional, por exemplo o texto de uma selecção.

Com as mensagens bem definidas é agora possível ligar todos os componentes, fazendo

passar por eles toda a informação que esperam. A definição das mensagens, teve em conta a necessidade de passar a informação suficiente e essencial, representando de modo útil e funcional os dados, de modo a que os adaptadores pudessem ser alimentados com qualidade. A sua forma simples traz algumas vantagens, seja a nível de processamento, pois as mensagens do sistema nunca serão muito grandes, seja a nível da legibilidade, pois são auto-explicativas.

3.1.2 *Namespace* e Validação

A definição das mensagens, que vimos em detalhe, está descrita exaustivamente num XSD. É o XSD que é usado na *framework* para realizar validações das mensagens. Como forma de tornar a especificação mais robusta, as mensagens que a *framework* utiliza fazem uso de um *namespace*[35]. A utilização de um *namespace* serve para evitar ambiguidades a nível dos elementos e atributos que compõem as mensagens. Os *namespaces* são compostos por um URI que fica associado a cada elemento e atributo, desambiguando-os dos restantes elementos. O URI usado não tem necessariamente que existir, mas é normal usar-se endereços válidos que apontam por exemplo para as especificações do documento.
O *namesapce* usado na *framework* para as mensagens foi:

```
xmlns="http://www.niaad.liacc.up.pt/SOM"
```

Para facilitar a utilização do *namespace* é normal usar um prefixo que represente o *namespace* evitando a reescrita do *namespace* que pode ser extenso. No nosso caso o prefixo escolhido para o *namespace* foi `som`. Quer o *namespace* quer o prefixo foram excolhidos de forma a manter a ligação ao nome do projecto. Com a utilização de um prefixo o *namespace* passa a ser representado da seguinte forma:

```
xmlns:som="http://www.niaad.liacc.up.pt/SOM"
```

A inclusão do *namespace* nas mensagens faz com que os elementos tomem a seguinte forma:

```
<som:message mode="adapt"
          xmlns:som="http://www.niaad.liacc.up.pt/SOM>
 <som:adapt ...>
  ..........
 </som:adapt>
</som:message>
```

Esta nomenclatura assegura a ausência de conflitos de nomes que possam existir em elementos ou atributos, passando esta linguagem a ser de exclusiva utilização da *framework*.
A especificação das mensagens recorrendo a um XSD permite validar as mensagens em qualquer ponto do sistema. Deste modo, temos um mecanismo de controle sobre a correcção das mensagens a nível da sua estrutura. Um efeito deste procedimento é a introdução de uma componente de segurança, pelo menos no que diz respeito às mensagens, evitando que mensagens que não correspondam ao esperado sejam utilizadas com o objectivo de interferir no funcionamento da *framework*.

3.2 Cliente

Nesta arquitectura o *browser* mais utilizador, tem um papel de **cliente de adaptação**, através de funções que são adicionadas à página. Essas funções permitirão comunicar com o *Broker* passando-lhe as informações necessárias. Para isso, deverá construir mensagens que depois enviará ao *Broker* pelos canais de comunicação adequados. O Cliente, extensão a introduzir na página, tem várias responsabilidades, visto que depende de si a interacção com o utilizador. É ele que deve processar a página em que estão identificados os elementos de adaptação, construir a mensagem que irá ser enviada com a informação recolhida e, depois de receber uma resposta e processar a

mensagem, deverá ainda proceder ás alterações necessárias na página para reflectir o processo de adaptação. O Cliente possui ainda outra função que é tão importante quanto a adaptação, a notificação.

A notificação permite registar um grande número de interacções do utilizador com a página, por exemplo, os eventos gerados pelo rato, onde se pode perceber que o utilizador clicou ou seleccionou determinado elemento, ou os deslocamentos de página, que o utilizador fez para ver determinada informação que se encontrava fora do ecrã. Estes e outros eventos permitem registar um elevado número de comportamentos do utilizador, para mais tarde alimentar os Adaptadores.

Com estas funções o Cliente pode ser estruturado com base em três acções distintas:

- Processamento de pedidos de adaptação e notificações
- Manipulação de mensagens
- Manipulação de páginas

Uma possível representação para esta estrutura é a do diagrama da figura 3.12. O Cliente pode assim ser definido por três elementos, associando a cada um deles as funções que vimos atrás. Tomando os nomes da figura temos os seguintes objectos:

Message Constrói e processa as mensagens do Cliente, sejam elas as mensagens de adaptação ou notificação. Dado ser o elemento responsável pelas mensagens poderá ainda realizar operações sobre as mensagens que sejam importantes para o funcionamento do processo do lado do Cliente, como por exemplo a serialização.

Template Processa uma página podendo realizar duas acções: retira os dados necessários para formar a mensagem a enviar para realizar uma adaptação ou, realiza alguma alteração na página, baseada na informação contida na mensagem devolvida pelo *Broker*.

Adapt Inicia os processos de adaptação e notificação interagindo com os objectos anteriores. Usando o primeiro, constrói mensagens de acordo com os dados

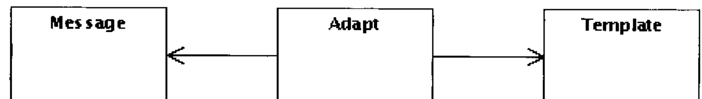


Figura 3.12: Arquitectura da implementação a realizar no Cliente.

indicados e obtidos usando o segundo. Responsabiliza-se pela comunicação enviando e recebendo as mensagens para e do *Broker*.

Estas acções são despoletadas, por exemplo, quando um utilizador está perante uma página e clica num determinado *link* que aponta para essa mesma página, por exemplo num menu de um *site*. Seguindo a arquitectura do Cliente, a página seria processada para obter uma lista com os vários menus e seria compilada numa mensagem. A mensagem seria enviada ao *Broker* que se encarregaria de outras operações, quando a mensagem de resposta chegasse seria processada. Imaginemos, por exemplo, que o mesmo menu era devolvido mas numa ordem diferente, essa ordem passaria a ser a nova ordem no menu da página em questão. O Cliente trataria então de reordenar a página da forma indicada.

Analisando a figura 3.13, podemos perceber melhor como se articulam os objectos para realizar os dois tipos de processos de que o Cliente é responsável: a adaptação e a notificação. Os dois distinguem-se bem pelo facto de na notificação não haver resposta do *Broker*, apenas há um envio de informação. Exceptuando isso, os dois processos são sensivelmente idênticos até ao envio da mensagem. Ambos obtêm dados, constroem uma mensagem e enviam-na para o *Broker*. A partir daqui apenas a adaptação tem continuidade, a mensagem de resposta com os resultados da adaptação chega e dá-se o processo inverso, a mensagem é processada e consoante a sua informação a página é alterada. As notificações apesar de serem bastante mais elementares são fundamentais no processo, pois permitem a recolha de informações sobre a interacção do utilizador. A movimentação do rato, o clique num determinado elemento ou o seleccionar são informações que o Cliente pode recolher através da observação de eventos. Sempre que um evento é detectado uma mensagem é construída e é enviada para o *Broker*,

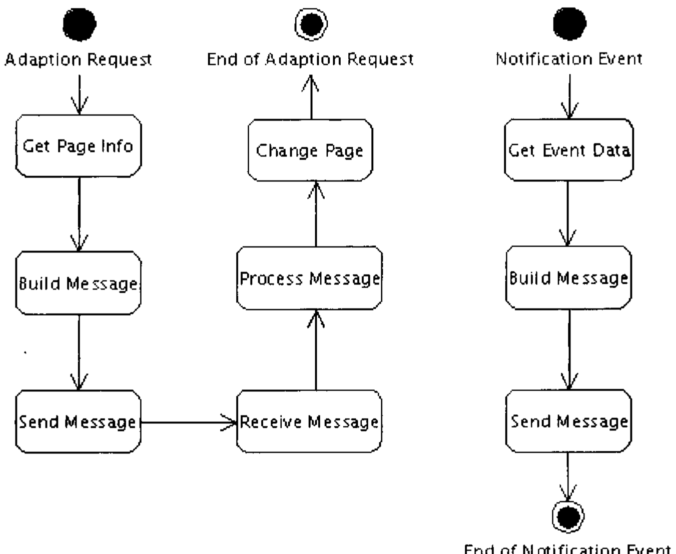


Figura 3.13: Diagrama de actividades dos processos de adaptação e notificação do Cliente.

como neste caso as notificações destinam-se apenas a registo, o processo termina tendo apenas um sentido.

3.3 *Broker*

O *Broker* assume o papel principal colocando-se como ponto central da arquitectura. Apesar do *Broker* não poder ser definido como um interveniente activo no processo de adaptação, isto é, não intervém nas mensagens onde estão os dados de adaptação em qualquer um dos sentidos, o seu papel de *proxy* é fundamental para a circulação de informação e registo de eventos.

O *Broker* encaminha as mensagens recebidas de qualquer um dos outros componentes fazendo-as seguir para o destinatário correcto, ou no caso dos eventos gerados pelo utilizador, passando-as ao *logger* que regista esses eventos. Esta funcionalidade mínima, apesar de vital mas sem qualquer influência directa nos processo de adaptação, confere ao *Broker* uma grande simplicidade e baixos requisitos de processamento. No pior dos

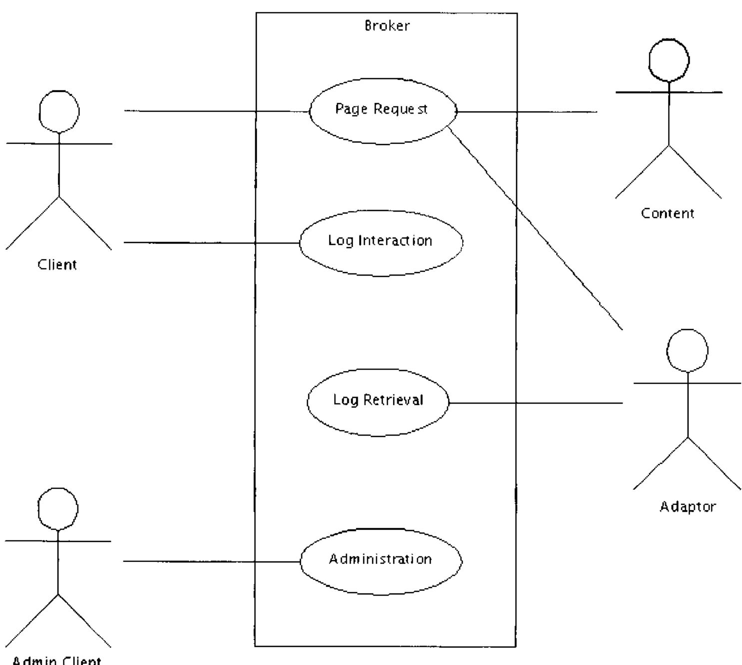


Figura 3.14: Diagrama de casos de utilização do *Broker.*

casos, o trabalho do *Broker* passa por receber uma mensagem do Cliente, descobrir a quem se destina essa mensagem e fazê-la seguir para um Adaptador, que irá devolver uma resposta a determinada altura que será encaminhada para o Cliente. Estas acções, são acções típicas de um *proxy*, isto é, não há interferência com a informação que circula, existe sim um processamento secundário em função dessa informação, caso do *logging,* e apenas se dá continuidade às comunicações.

Estas são as funções directamente relacionadas com o processo de adaptação, apesar de existirem outras que o *Broker* pode realizar como se pode ver na figura 3.14. Na realidade, os processos que influenciam directamente a navegação do utilizador são apenas dois: adaptação e notificação, no entanto, o *Broker* pode ainda receber pedidos de administração. Quando a *framework* está a ser usada e é feito um pedido a uma determinada página, duas coisas necessariamente ocorrem: o pedido da página e o pedido de adaptação sobre essa mesma página. O pedido da página é um pedido normal, como normalmente ocorre quando se acede a um determinado *site* e se deseja ver o conteúdo. Isto ocorre imediatamente para que o utilizador visualize os conteúdos

do seu pedido. Paralelamente a isso, um segundo pedido, o de adaptação, é feito para que a página pedida seja adaptada em função do utilizador. Assim, numa primeira fase, o utilizador recebe uma página original com os conteúdos do *site* em que se encontra e que poderá ser imediatamente adaptada, chegando a ser imperceptível a adaptação.
É precisamente devido a esta necessidade de resposta que a *framework* deve dar, que o *Broker* se pensou leve e simplificado, de forma a não introduzir nenhum tipo de latência ou degradação quer nas comunicações quer no processamento da informação. Desta forma cumprem-se os requisitos a que nos propusemos inicialmente.
Adicionalmente a este processo de adaptação, existe ainda a necessidade de registar os eventos do utilizador, que interessam para lhe dar contexto e informação.
São estes dois processos que podemos observar na parte superior da figura 3.14 onde o cliente comunica com o *Broker* de duas maneiras possíveis. Uma para pedir o conteúdo da página e respectiva adaptação, que por sua vez é passada ao Adaptador, e outra para registar as interacções do utilizador.
É importante referir que nesta arquitectura se assume que as ligações entre os diferentes actores são asseguradas pelos protocolos de comunicação a eles naturalmente associados, estando apenas acordado inicialmente que todos esperam as mensagens da forma que já referimos. Assim, temos que cada actor pode ser algo completamente diferente do *Broker,* pois não há dependências físicas entre eles. Nomeadamente, o conteúdo pode variar entre ficheiros de texto básicos, páginas dinâmicas, uma base de dados ou um gestor de conteúdos complexo. A única coisa importante é utilização das mensagens impostas.
Por último, o *Broker* pode receber comandos de administração também via mensagens, estes comandos permitem alterar ou definir configurações e receber informações sobre determinados aspectos que o *Broker* possa conter.

3.4 Adaptador

Na outra extremidade da arquitectura (Figura 3.1) o Adaptador realiza processos de complexidade variável para realizar adaptações sobre os dados recebidos. O Adaptador é um **fornecedor de adaptação**.

Os processos de adaptação são independentes de toda a arquitectura, podendo ter implementações específicas. Um Adaptador pode assumir diferentes perfis. Podemos ter um adaptador que é na realidade um agregador de adaptadores que podem ser externos, isto é, programas criados por terceiros que extendem o Adaptador ou pode ser um adaptador especializado num determinado tipo de adaptação. E como é isto possível?

O facto de utilizar *web services* para comunicar com os Adaptadores, e de haver um protocolo bem definido e com regras, torna-os independentes do resto dos componentes. Ao esperar uma mensagem com um determinado formato, que pode inclusive ser validada, e responder com uma mensagem que obedece às mesmas regras, permite explorar diversas possibilidades. Os Adaptadores podem implementar desde todo o processo de criação de mensagens com dados, ao protocolo de *web services* para a comunicação com o *Broker.* Como podem apenas ser um programa numa determinada linguagem e que produz como *output* valores pré-definidos, utilizados depois para construir as mensagens de resposta. Esta possibilidade tem duas vantagens:

- a utilização de programas anteriores à *framework* que podem já responder às necessidades de adaptação, podendo eventualmente terem que ser ligeiramente alterados para responder aos requisitos do sistema;
- a possibilidade dos criadores de adaptadores poderem rapidamente colocar em funcionamento um adaptador na sua linguagem favorita, ganhando assim tempo de desenvolvimento e estimulando a criação.

Deste modo, a *framework* assume um papel interessante para os investigadores, podendo ser uma ferramenta de suporte à investigação. Um investigador pode usufruir

do tipo de dados que a *framework* fornece para realizar testes e validações de ideias relacionadas com a adaptação.

Durante o processo de adaptação o conteúdo da mensagem criada no Cliente e que chegou ao Adaptador via *Broker* é fornecido a um ou vários processos que o Adaptador conhece, passando-lhes os dados necessários a uma computação que irá resultar numa adaptação. Estes processos podem estar no próprio Adaptador ou serem na realidade execuções de programas pré-existentes. O resultado de qualquer um deles é transformado novamente numa mensagem que respeite as regras e que é enviada novamente de volta. Isto permite uma separação total entre componentes, o que é estritamente necessário dada a diferença de contextos entre Cliente e Adaptador.

O contexto de um Cliente é bastante rico se tivermos em conta a presença do utilizador, as suas acções, toda a estrutura da página disponível e os conteúdos. Este contexto é bastante diminuto no Adaptador se pensarmos que este se encontra sempre a uma distância que não lhe permite conhecer directamente as acções do utilizador. Isso faz com que seja necessário dar ao Adaptador o maior contexto possível ou pelo menos, aquele que o Adaptador entender ser necessário para produzir os seus resultados. É aqui que a estrutura e definição das mensagens têm um papel fundamental para a manutenção desse contexto, quer permitindo a passagem de conteúdos, quer fazendo registos dos eventos que o utilizador gera para que o Adaptador possa ter essa informação.

Um ponto que vale a pena também referir, é que nesta arquitectura o *Logger* não está estritamente ligado ao *Broker* no que diz respeito à leitura de registos. É possível que o Adaptador aceda ao *Logger* directamente para obter as informações de que necessita para as sua operações. Evita-se assim introduzir carga no sistema e dando-lhe liberdade para realizar os seus processos paralelamente e sem restrições temporais. Com esta solução podemos ter vários Adaptadores em simultâneo mas com tempos de processamento diferentes. Se imaginarmos uma situação em que são feitos dois pedidos de adaptação, em que um deles é um simples processo de ordenação e outro envolve um processo complexo de *data minning*, facilmente se percebe a importância de poder realizar pesquisas sem recorrer aos restantes componentes da *framework*. Aliás, com

esta solução os adaptadores podem até realizar tarefas paralelamente ou quando não recebem pedidos por parte do *Broker*, isto é, quando estão em inactivos, acelerando eventuais respostas a novos pedidos.

A última questão a que os Adaptadores devem responder, e que está directamente relacionada com a discussão que já vimos sobre o que é uma adaptação, é o problema da adaptação que é uma recomendação e a adaptação que é uma reformatação. Esta dualidade a que os Adaptadores estão sujeitos traz implicações para as mensagens e para as tarefas que os Adaptadores vão realizar, aumentando mais tarde a dificuldade para o Cliente, que terá de interpretar cada linha da mensagem e executar as respectivas alterações. Apesar de ser possível optar por apenas uma solução, entendemos que seria mais interessante e enriquecedor se déssemos resposta às duas hipóteses, aumentando assim o domínio de acção da *framework*. Isto é mais importante ainda, dado que ao permitir utilizar qualquer tipo de Adaptador mantemos todas as hipóteses em aberto, não excluindo à partida Adaptadores interessantes.

3.5 Características SOA

Conhecendo agora a arquitectura dos vários componentes que constituem a *framework* e percebendo como se interligam, é mais fácil confirmar que realmente estamos perante uma arquitectura SOA. Basta que vejamos como a arquitectura segue algumas das suas directivas base:

- Os componentes relacionam-se entre si dependendo apenas das mensagens, desta forma não existem dependências de implementação;
- Há um pré-acordo no conteúdo da comunicação, optimizando os processos e diminuindo as possíveis falhas na interpretação;
- Os componentes são autónomos, isto é, apesar de terem que estar presentes e funcionais para que exista uma adaptação, as suas funções específicas estão contidas na sua lógica de operação;

- A utilização de serviços para os adaptadores mantêm escondidos os pormenores de implementação, havendo apenas preocupação com os dados comunicados.

Com estas características existe uma grande descentralização da *framework*. O facto de poderem existir diversos adaptadores que comunicam de uma forma perfeitamente definida, permite que existam adaptadores distribuídos. Ou seja, os adaptadores tanto podem estar no próprio servidor aplicacional do *Broker* como podem estar espalhados por diversos servidores heterogéneos. Aqui se mostra a diversidade que esta *framework* permite, quer ao nível dos seus próprios componentes, quer ao nível dos sistemas que a podem usar.
Esta possibilidade implica alguns cuidados que o sistema deverá ter em atenção. Ao ter Adaptadores descentralizados é necessário controlar de alguma forma o seu estado e localização, assim, os adaptadores devem registar-se no *Broker* não só para lhe dar a conhecer a sua existência mas também dar a conhecer o seu estado. É aqui que as mensagens de administração entram. Estas mensagens são mensagens iguais a todas as outras. A única diferença está na sua estrutura que faz uso de *tags* de administração que contêm comandos reconhecidos do sistema, como aliás já vimos na secção 3.1.1. Este tipo de controle aproxima-se um pouco das características de *reliability* que actualmente se definem como características de uma arquitectura orientada a serviços moderna.

Capítulo 4

Implementação do SOM

Neste capítulo apresentamos os detalhes de implementação duma *framework* para adaptação de *sites web* baseados em HTML seguindo a arquitectura exposta no capítulo anterior. É também descrita a implementação de um adaptador que permite a ligação de componentes externos escritos em linguagens de *scripting*.

A implementação baseada numa *framework* é constituída por pontos fixos - ***coldspots*** - e pontos de extensão - ***hotpsots***. Os pontos fixos apresentam-se como os componentes da *framework* que fornecem as capacidades de agregação da arquitectura e que são únicos e constantes durante qualquer utilização. Os pontos de extensão são os elementos que são mutáveis para uma qualquer utilização. Como pontos fixos temos o Cliente, o *Broker* e o *Logger*, assumindo uma posição central, e como pontos de extensão as diferentes páginas ou conteúdos e os diferentes Adaptadores.

Uma definição que surge deste desenho, e que vimos nos capítulos anteriores, é a noção de **clientes de adaptação** e **fornecedores de adaptação**. Estes consumidores e produtores estão ligados pelo *Broker* dando-lhes a capacidade de interagirem de forma transparente, graças à definição de uma linguagem de comunicação que representa a adaptação. Com o *Broker* estabelece-se uma ponte entre os pontos de extensão, através da qual fluem mensagens com informação com vários destinos e diferentes conteúdos. As mensagens podem ser destinadas ao *Broker*, mais concretamente ao *Logger*, ou

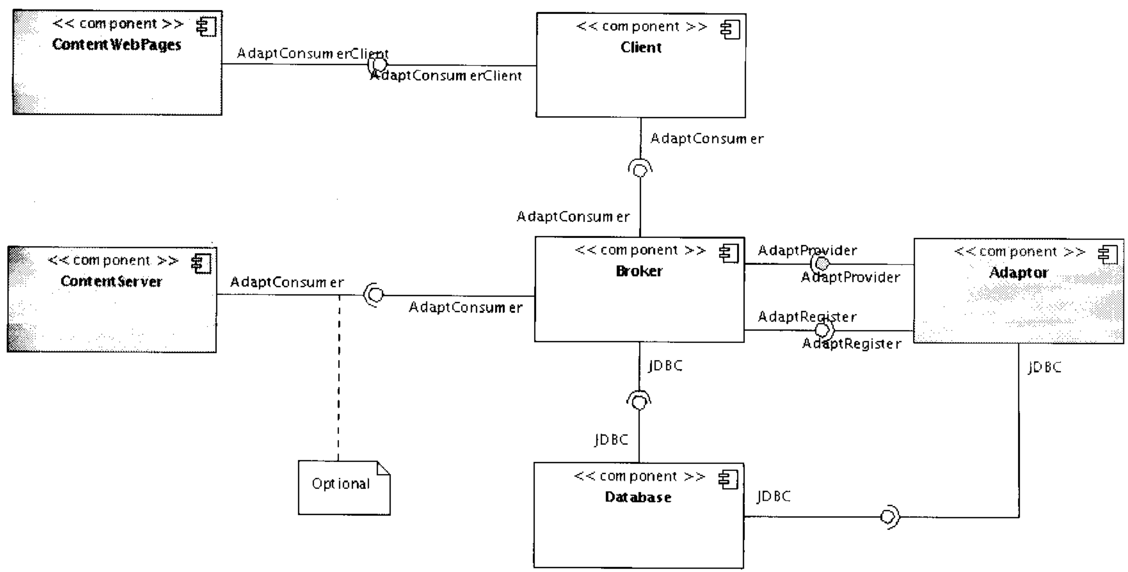


Figura 4.1: Componentes da *framework* e respectivos "interfaces funcionais".

aos Adaptadores e podem conter dados para adaptar ou já adaptados, bem como, informação que vai ser registada para mais tarde moldar a adaptação.

4.1 Infra-estrutura de adaptação

A disposição dos diferentes componentes da *framework* segue o seu papel na mesma. O Cliente encontra-se junto do utilizador, estando inserido nos conteúdos a que este tem acesso, de forma a compilar os dados necessários à adaptação. O *Broker* está num plano central onde medeia as comunicações e tem um contacto privilegiado com o sistema de gestão de base de dados. Usando o *Logger* guarda todas as informações necessárias ao funcionamento dos Adaptadores. É isso que podemos ver na figura 4.1 onde estão esquematizadas as interacções dos componentes, e onde se destacam os os *hotspots* nas extremidades, a cinzento, e num plano central os *coldspots*. Cada componente providencia um **"interface funcional"** criando vias de comunicação entre si. O Adaptador deve providenciar o interface `AdaptProvider` para poder ser invocado pelo *Broker*. Basicamente é a WSDL do *web service*. O *Broker* providencia

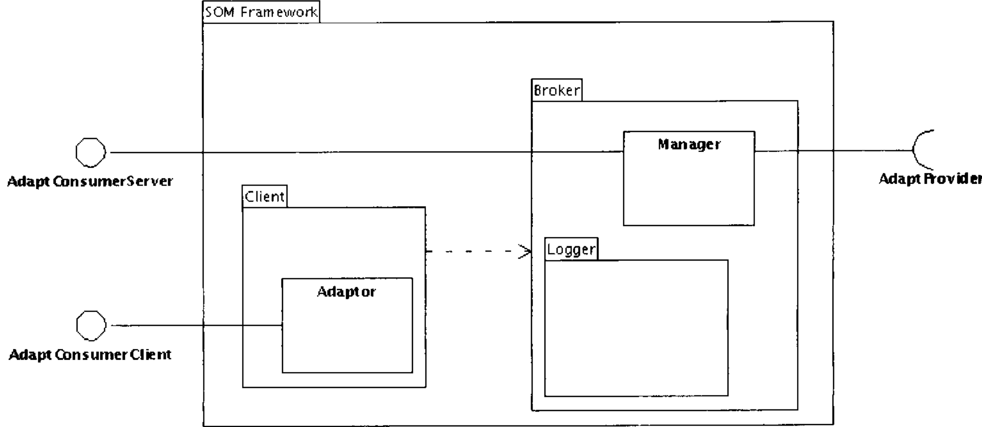


Figura 4.2: Estrutura e interfaces disponibilizados pela *framework*.

os interfaces `AdaptConsumer` e `AdaptRegister`. O interface `AdaptRegister` é usado inicialmente pelo Adaptador para se registar na *framework*. O *Broker* também usa um interface JDBC duma base de dados para guardar os eventos. O `AdaptConsumer` é usado para registar a interacção do utilizador e requerer a adaptação de páginas. Este interface pode ser usado opcionalmente pelo servidor de conteúdos para pedir adaptações a partir da camada de apresentação. No entanto, a função primordial deste interface é ser usado pelo componente Cliente para comunicar com o *Broker*. O componente Cliente disponibiliza o interface `AdaptConsumerClient` que é um conjunto de métodos JavaScript que podem ser invocados a partir das páginas do conteúdo *web* a adaptar. Do ponto de vista da implementação esta organização traduz-se na estrutura da figura 4.2. A implementação em Java da *framework* apresenta-se com uma estrutura em pacotes contendo as classes que compõem cada um dos componentes. Na figura estão representados apenas os pacotes que dizem respeito à *framework* e a este trabalho. Todas as possíveis implementações de Adaptadores ficam de fora, visto que não fazem parte da *framework*. Os dois pacotes, Cliente e *Broker*, este último com um sub-pacote responsável pela interacção com a base de dados, formam um pacote maior, que é o da *framework*. Este é o pacote que será distribuído e com ele ficará disponível a *framework* para que alguém implemente um ou mais Adaptadores completando o circuito de adaptação. Não devemos esquecer que uma *framework* é apenas um meio

de, não sendo só por si suficiente para funcionar como aplicação. A *framework* apenas fornece os mecanismos, neste caso os de recolha de dados, de registo de dados, de comunicação de dados e de manipulação de conteúdos. Tudo o resto, também pela própria independência da solução, é implementado à posterior.

4.2 Broker

O *Broker* é o ponto central da *framework*, por ele passam todas as comunicações tendo uma função assumidamente de *proxy* e *logger*. Mensagens de adaptação e notificação partem do Cliente, são analisadas e seguem os seus respectivos caminhos. Apesar desta simplicidade de funções e de não ser necessário um grande poder computacional para as realizar, é importante realizá-las com eficiência e rapidez, por isso a implementação do *Broker* tenta ser o mais simples possível tentando seguir estas directivas.
É importante perceber que esta *framework* fornece as capacidades para uma utilização distribuída, isto é, podem existir simultaneamente n clientes e n adaptadores, tudo isto graças ao *Broker* que organiza e distribui as comunicações. Por um lado isto confere-lhe alguma complexidade em determinados aspectos, como por exemplo, o conhecimento dos diversos adaptadores e a sua disponibilidade, ou a capacidade de receber notificações de n clientes e inseri-las numa base de dados. Num *site* que tenha um número razoável de acessos, se multiplicarmos por todos os utilizadores eventos de clique, *focus* ou selecção, conseguimos antever uma escalada das mensagens que chegam ao *Broker* e que necessitam de ser registadas imediatamente de forma segura e fiável. Isto faz com que possamos definir duas características do *Broker* que devem ser tomadas em conta na implementação:

- Manutenção de múltiplas ligações entre Cliente e Adaptadores;
- Elevada eficiência no processamento de mensagens e utilização da base de dados.

Estas características vão reflectir-se nas tarefas que o *Broker* deve desempenhar e que devem também ser levadas em conta na implementação:

- Envio e recepção de mensagens - *Proxying*;
- Registo de adaptações e eventos - *Logging*;
- Administração de configurações.

4.2.1 *Proxying*

As mensagens de adaptação passam pelo *Broker* com destino aos Adaptadores. Para que o *Broker* saiba como encaminhar estas mensagens tem de: analisar as mensagens para encontrar o identificador do Adaptador a usar e descobrir na sua própria configuração o endereço desse Adaptador.

As configurações do *Broker* permitem-lhe conhecer os nomes e endereços dos adaptadores. Quando um cliente envia uma mensagem, o *Broker* deve identificar o tipo de mensagem e procurar nela as informações de que necessita. Isto é feito recorrendo às classes `javax.xml.parsers.DocumentBuilder`[20] e `javax.xml.xpath.XPath`[20]. Estas classes possuem métodos que permitem, respectivamente, criar objectos a partir de documentos XML e processar esses mesmos objectos extraindo o seu conteúdo.

Encontrado o Adaptador, o *Broker* utiliza o canal de *web service* para enviar a mensagem, e quando recebe a respectiva resposta envia o resultado ao cliente pelo canal HTTP criado inicialmente e que se encontra em espera. A comunicação *web service* com os Adaptadores é assegurada pelo protocolo SOAP[28]. Este protocolo, usado em *web services*, assegura a comunicação entre os extremos de um serviço, criando toda a estrutura necessária para a circulação dos dados no canal de comunicação.

A comunicação usando *web services* recorre ao Axis[3] para integrar o protocolo SOAP no *Broker* e Adaptador.

O Axis permite quatro tipos de comunicação: RPC, *Document*, *Wrapped* e *Message*. Nesta implementação foram usados serviços do tipo **Message** que permitem deixar para o código contido na implementação dos serviços, a manipulação das mensagens contidas no envelope SOAP.

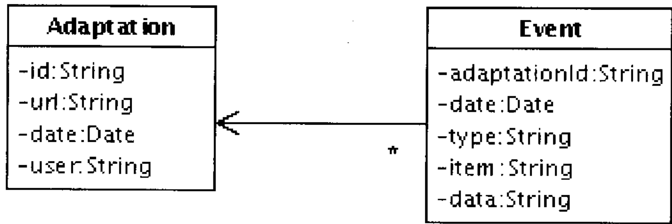


Figura 4.3: Esquema da base de dados.

4.2.2 *Logging*

A função que requer mais processamento, e que o *Broker* tem de fazer, é provavelmente o registo dos pedidos de adaptação e das notificações de eventos que são gerados pelo utilizador. O *Broker* analisa todas as mensagens que lhe chegam por parte dos Clientes e decide, em função do seu tipo, o que fazer. Quer as mensagens de adaptação quer as de notificação originam um registo na base de dados. Na recepção duma mensagem de adaptação é registado esse pedido e a informação associada. Na recepção duma mensagem de notificação é registado o evento que a gerou e a informação que lhe está associado.

Optamos por guardar esta informação numa base dados, apesar de ser possível fazê-lo usando outros métodos, com um ficheiro de *log*. No entanto, para conveniência dos Adaptadores é preferível a utilização de um sistema de gestão de base de dados. Assim, qualquer Adaptador pode fazer interrogações complexas sobre os registos que foram guardados.

A base de dados tem duas tabelas para guardar esta informação, as tabelas `Adaption` e `Event` esquematizadas na figura 4.3. A tabela `Adaption` guarda os dados relacionados com a adaptação, dados esse extraídos do cliente e representados nas mensagens. Os campos da tabela são:

`id` - identificação única da página actual, campo chave da tabela.

`url` - endereço *web* da página corrente, pode ser usado para mais tarde criar novos

links.

`date` - instante de tempo ao segundo do pedido de adaptação realizado, podendo ser usado para calcular tempos de visualização.

`user` - a identificação única do utilizador.

`ip_address` - o endereço IP do cliente.

`user_agent` - o *browser* usado pelo cliente, servindo como informação adicional para os adaptadores ou mesmo estatística.

A tabela `Event` guarda informação relacionada com eventos gerados no cliente. Quando um elemento está a ser monitorizado por eventos sempre que se registar um, uma mensagem de notificação é enviada ao *Broker* para que seja registada a acção. Estas informações são destinadas aos adaptadores fornecendo informações detalhadas sobre a navegação do utilizador. Vejamos também em detalhe os campos desta tabela:

`adaptionId` - identificador único da página adaptada onde se registaram os eventos. Chave externa para a tabela `Adaption`.

`date` - instante de tempo ao milissegundo do evento registado.

`type` - tipo do evento registado, *click*, *focus*, etc.

`item` - elemento HTML sobre o qual se registou o evento.

`data` - dados extra que foram recolhidos no evento, por exemplo os caracteres seleccionados do texto.

4.2.3 Implementação

O diagrama de classes da figura 4.4 representa a estrutura do *Broker*, que foi parcialmente implementado no protótipo da *framework*. Por exemplo, no protótipo da *framework*, a gestão de configurações, é estática, não há controle das excepções

nem existe listagem de adaptadores disponíveis. Existem dois canais de comunicação para as mensagens no *Broker*. Uma canal via HTTP e outro canal via *web services*. Olhando com maior atenção vemos que as classes responsáveis por esses canais são, respectivamente, a `RequestHandler`, que é uma *servlet*, e a `ServiceHandler`. Estas duas classes asseguram toda a comunicação, respectivamente pelo protocolo HTTP e SOAP. São elas que providenciam os "adaptadores funcionais" `AdaptConsumer` e `AdaptRegister`. Em ambos os casos o conteúdo XML das mensagens é passado à classe de controle `Manager`. Estas classes prevêem a possibilidade de comunicação com o *Broker* via HTTP ou via *web services*. Estas classes são também as responsáveis pela criação dos "interfaces funcionais" assinalados a cinzento e que vimos na figura 4.1.

A classe `Manager` é o núcleo do *Broker* e é um *singleton*, assegurando que todo o controle é realizado pelo mesmo objecto, evitando problemas de concorrência. Os objectos `MessageImpl` são implementações do interface `Message` que providencia uma série de métodos para manipulação das mensagens, como a leitura ou escrita de elementos e atributos e a serialização do conteúdo. Cada objecto irá representar uma mensagem com os seus métodos específicos, acelerando o seu processamento em qualquer ponto do *Broker*.

No ponto central do Broker, a classe `Manager`, baseada na análise da mensagem contida numa instância de `MessageImpl`, invoca uma das possíveis classes que implementam `Command`. Todas as classes recebem como argumentos um objecto `Message`. As classes que podem ser invocadas dependendo da mensagem recebida são: `Notify`, `Adapt` e `Admin`, cada uma delas com uma função diferente:

`Notify` Recebe as mensagens de notificação processando-as e passando ao *Logger* a informação a ser registada na base de dados.

`Adapt` Recebe as mensagens de adaptação, processa-as e envia-as ao respectivo Adaptador via *web services*.

`Admin` Recebe as mensagens de administração com informação relevante ao *Broker*

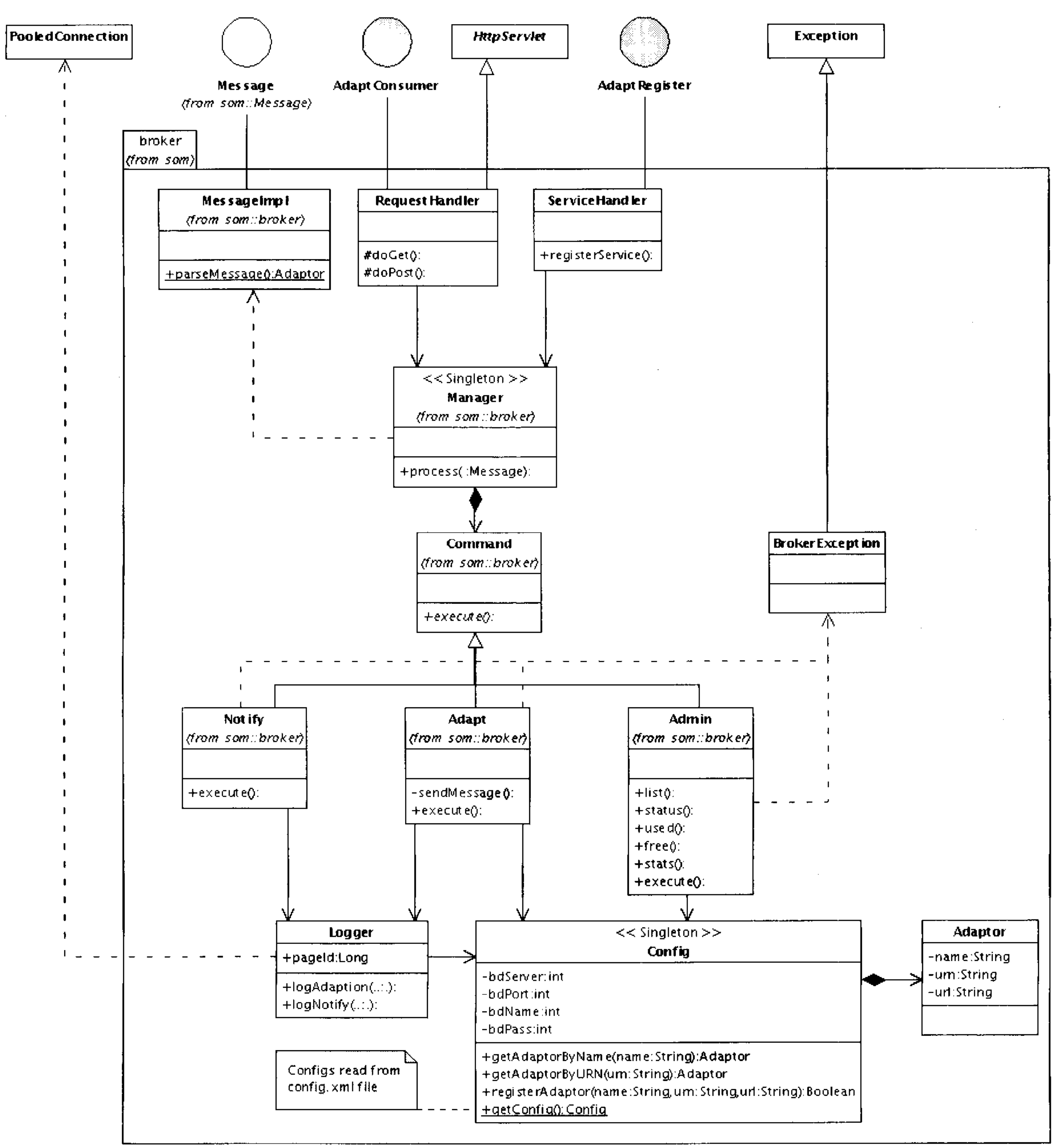


Figura 4.4: Diagrama de classes do *Broker*.

e/ou Adaptadores actualizando as informações locais de acordo com o indicado.

Estas três classes lançam excepções do tipo `BrokerException`, que contêm informações relevantes para os problemas que ocorram durante o processamento e que impeçam a continuidade do *Broker*, caso contrário os erros detectados são reportados nas mensagens de resposta.

Paralelamente e em função do tipo de mensagem recebida as classes `Notify` e `Adapt` encarregam-se de desencadear o processo de *logging* usando a classe *Logger*. Esta classe implementa uma série de métodos que recorre a conexões persistentes usando a API disponibilizado pelo *driver* do sistemas de gestão de base de dados usado. Este tipo de ligações à base de dados caracteriza-se por um conjunto (*pool*) de ligações em *standby* que são invocadas a pedido da aplicação, minimizando os tempos necessários para efectuar uma nova ligação. Deste modo, aceleram-se as inserções que o *Broker* tem de fazer.

Por fim as classes `Adapt`, `Admin` e `Logger` têm a particularidade de partilharem uma classe que contém todas as configurações do *Broker*, a classe `Config`. Esta classe mantêm todas as configurações relativas ao funcionamento do *Broker*, como se percebe pela figura. São guardadas definições para a base de dados, uma lista de adaptadores representada pela classe `Adaptor`, e métodos que permitem pesquisar e registar adaptadores.

4.3 Cliente

O Cliente tem duas funções primárias: realizar operações sobre mensagens e realizar a reformatação e transformação de conteúdos. As operações sobre as mensagens incluem: a criação de mensagens que representam os dados a serem adaptados no Adaptador e enviadas através do *Broker*, a interpretação de mensagens que chegam do *Broker* provenientes dos Adaptadores, com os resultados da adaptação e a criação de mensagens de notificação com os eventos gerados pelo utilizador. As operações

de reformatação ou transformação, são as operações sobre o conteúdo e que visam reflectir as adaptações realizadas. Nas operações sobre as mensagens o Cliente deve ter a capacidade de construir e processar as mensagens mas também de as enviar e receber, se possível sem prejudicar a navegação do utilizador. No que diz respeito à manipulação do conteúdo o Cliente deve ter a capacidade de interagir com o conteúdo, se possível de forma simples e sem introduzir efeitos secundários.

4.3.1 Utilização de Ajax

Do ponto de vista da implementação a utilização de Ajax é simples, bastando algumas linhas de código para o implementar. O primeiro passo é a criação de um objecto do tipo `XMLHttpRequest`[1]:

```
xmlObj = new XMLHttpRequest();
```

De seguida, para que a resposta ao pedido possa ser processada, deve ser indicada uma função que saberá o que fazer com a resposta recebida:

```
xmlObj.onreadystatechange = handleResponse;
```

Para definir os parâmetros do pedido a função `open` recebe três parâmetros: o método do pedido, `GET` ou `POST`, o URL para onde o pedido será feito e se o pedido é ou não assíncrono, `true` ou `false`. Com estes três parâmetros fica definido o novo pedido a fazer:

```
xmlObj.open("POST","http://localhost/Broker",true);
```

[1]Este exemplo de criação do objecto `XMLHttpRequest` é muito simplista, numa aplicação real seria necessário verificar qual o tipo de *browser* que está a ser usado pelo utilizador. Isso serviria para se verificar a compatibilidade com o JavaScript e, em caso afirmativo, criar o objecto de acordo com as especificações do *browser* em questão, visto que de *browser* para *browser* existem pequenas diferenças.

Para fazer o pedido propriamente dito existe a função **send** que recebe como argumento os dados a enviar[2]:

```
xmlObj.send(xmlDocument);
```

Quando uma resposta é recebida, o método **onreadystatechange** passa o controlo à função indicada para que possa lidar com a resposta. A resposta tem que ser processada de forma a extrair dela a informação necessária, utilizando para isso a DOM [6][5].

No que diz respeito ao processamento de documentos, em particular ao HTML, uma facilidade que a DOM fornece é a possibilidade de navegar directamente para um ponto através de um identificador no documento. O identificador, que é único para cada elemento do código, desambigua os elementos com os mesmos nomes evitando assim colisões. Esse identificador é o atributo **id**[3]. Com a utilização deste atributo é possível identificar um determinado elemento e aceder directamente a esse nó via DOM. A partir daí pode-se realizar alterações ou extrair informações relativas a esse nó.

A utilização deste atributo faz parte da especificação da W3C[32] para o HTML[12] e XHTML[34].

Com a DOM é possível realizar alterações em tempo real no código HTML que do ponto de vista do utilizador não implicam um redesenhar completo da página, pois podem ser realizadas em pontos específicos sem realizar um novo pedido. As funções de navegação pela estrutura em árvore, aliadas ao facto de esses nós conterem dados e guardarem em folhas os valores contidos em atributos, permitem a extracção ou colocação de dados de forma dinâmica.

As alterações introduzidas tem efeito imediato pois são realizadas na estrutura que o *browser* tem em memória para fazer o *render* da página.

[2]No caso do pedido ser um GET não é necessário enviar nenhum tipo de dados, passando o argumento da função `send` a ser `null`.

[3]Consideram-se elementos do HTML as *tags*. São exemplos de elementos: html, body, td, li e p.

4.3.2 Preparação dos conteúdos a adaptar

Para que um *site* possa utilizar a *framework* necessita de incluir a extensão que lhe vai permitir realizar pedidos de adaptação e coleccionar dados para adaptação. Essa inclusão deverá ser simples e deverá permitir realizar, de forma transparente, as duas operações da responsabilidade do Cliente. A ideia é utilizar as tecnologias já existentes e aproveitá-las para introduzir a extensão no cliente. Como assumimos que os *sites* que utilizarão a *framework* estarão construídos em HTML, escolhemos o JavaScript para criar a extensão. Construímos uma biblioteca que através de uma função introduz a extensão no cliente.
Esta função é incluída no elemento `<body>` de cada página recorrendo ao evento `onLoad`. Desta forma, cada vez que a página é carregada é realizado um processo de adaptação. Um exemplo da utilização da função `adapt()` em código HTML é:

```
...
<script type="text/javascript"src="somjslib.js" ></script>
...
<body onload="adapt('urn:som-adapter:recommend','','history');" >
...
```

A função `adapt()` pode ser utilizada múltiplas vezes, isto é, não há limitações para o número de adaptações a realizar desde que sejam realizadas sobre elementos diferentes. Um exemplo dessa situação seria:

```
...
<script type="text/javascript"src="somjslib.js" ></script>
...
<body onload="adapt('urn:som-adapter:a','pA pB','elementA');
adapt('urn:som-adapter:b','pA pC pD','elementB');" >
...
```

A chamada da função no código HTML do cliente dá início ao processo de adaptação. Esta função activa todo o processo e passa a informação necessária ao sistema. A função tem três argumentos:

```
adapt(adaptor_id,adaption_parameters,element_id);
```

- `adaptor id` - a identificação única do Adaptador a utilizar. A identificação permite ao *Broker* localizar adaptador. Este id é um URN[31] e pode ser opcional;
- `adaption parameters` - parâmetro opcional que permite passar ao Adaptador dados específicos que poderão influenciar o processo;
- `element id` - identificador único do elemento a adaptar na página. Este elemento é um valor de um atributo do tipo ID[13] do documento.

O facto de a identificação do adaptador ser opcional serve para os casos em que apenas se pretende a recolha de dados sobre um determinado elemento, isto é, como a função `adapt` indica um elemento através do atributo `id` sobre o qual se realizam as adaptações e recolhas de dados, se não for indicado nenhum adaptador, apenas são coleccionados dados. Esta opção é interessante se existir uma secção de uma página que não se quer adaptada mas que a sua utilização pode contribuir para a adaptação. A utilização do evento `onLoad` para activar a adaptação, deve-se ao facto de a adaptação se poder processar em paralelo com o carregamento da página. Como os pedidos são assíncronos, o processo de adaptação pode ser feito de imediato e alterar a página carregada sem obrigar a um novo pedido HTTP, não havendo assim um redesenho que o utilizador imediatamente notaria.
Para que esta extensão resulte é obrigatória a utilização do atributo `id`. Esta marcação que actualmente já existe em grande parte dos *sites*, é apenas uma recomendação do W3C. Isto significa que em *sites* mais antigos, será necessário algum trabalho de marcação dos elemento a adaptar. Este trabalho em certas situações poderá ser mais

fácil ou mais difícil de executar, dependendo da forma como *site* se organiza. Em situações em que são usadas *templates* para geração das páginas o trabalho poderá ser mais fácil, e noutras em que se está perante uma estrutura de directórios e ficheiros, mais demorada. No entanto em qualquer das situações apenas será necessário fazer essa marcação uma vez.

A necessidade de identificar os elementos a adaptar não se prende só com dar a conhecer a localização dos elementos. Prende-se também com a recolha dos dados da página, que devem ser incluídos na mensagem do pedido de adaptação, e com a maneira como devem ser alterados quando é necessário realizar reformatações. Isto significa que não basta saber localizar o ponto da página onde se quer realizar uma adaptação, mas também saber quais e como ler os dados aí presentes.

Há várias soluções para este problema. A primeira seria partir do princípio que apenas determinadas estruturas como listas ou tabelas seriam alvo de adaptação e assim ter um algoritmo fixo para a leitura e colocação de dados. Como é óbvio esta solução tem muitas limitações, existem muitas mais estruturas numa página HTML que ficariam de fora desta solução.

Outra solução seria a inclusão de *tags* específicas no código HTML que apenas a extensão JavaScript reconhecesse e soubesse interpretar. As *tags* indicariam quais os dados a extrair e como recolocá-los após a adaptação. Esta solução tem o problema de ser intrusiva, ou seja, obriga a colocar por cada secção a adaptar, linhas extras de código que podem ser bastante complexas para depois servirem de *template* ao processo de reformatação. Esta solução tem o problema de destruir as validações do código HTML, algo que é encorajado, por questões de compatibilidade entre *browsers*, pelo W3C.

Por fim, a última solução é a de colocar paralelamente, por exemplo num ficheiro, essas mesmas *templates* a que a extensão JavaScript teria acesso e a partir daí saber quais os dados a extrair e qual a forma de os voltar a colocar na página após a adaptação.

4.3.3 Comunicação com o *Broker*

Na comunicação entre o Cliente e o *Broker* circulam dois tipos de mensagens: adaptação e notificação. Ambas implicam a construção de mensagens com a informação a que a estrutura que vimos na secção 3.1.1 obriga. As mensagens de adaptação implicam conhecimento do conteúdo das páginas e no caso das notificações o processamento de eventos gerados pelo utilizador.
Apesar de terem diferentes funcionalidades, em ambas existe um conjunto de informações que lhes é comum, nomeadamente, as informações sobre o utilizador, a página e o cliente. Esta informação está sempre do lado do cliente e cabe à extensão descobri-la. Para as informações relacionadas com o cliente (*Browser*, IP, etc) podemos recorrer a valores que o próprio JavaScript tem acesso através de variáveis globais que estão definidas em objectos, como é o caso do objecto `document`. No caso mais específico do IP, não é possível obtê-lo localmente por limitações da própria especificação, mas podemos facilmente recorrer a um pedido assíncrono a uma página que nos devolva o IP de quem a contactou.
Num processo mais complexo está a identificação do utilizador e da página (ou sessão). Esta descoberta é um pouco mais complexa porque não faria sentido reinventar utilizadores ou sessões quando muitas vezes estão já definidos nos *sites*. Estas informações podem ser obtidas analisando locais comuns onde estes valores são guardados, como o URL ou uma *cookie*. Como alternativa pode ser indicado onde localizar esses valores por parte do administrador do *site*. Nos casos em que não existe qualquer tipo de informação sobre utilizadores ou sessões, a própria extensão define-os e guarda-os em *cookies* temporárias.
No que diz respeito à adaptação estes são os dados, para além dos que se encontram no próprio *site*, que é necessário obter para ser possível construir uma mensagem de adaptação. Ficam a faltar as mensagens de notificação.
As mensagens de notificação para além de terem de indicar os mesmos dados sobre o utilizador, página e cliente, por forma a ser possível o cruzamento de dados, necessitam de indicar os eventos gerados pelo o utilizador. A informação sobre o

evento inclui o local, o momento, o tipo de evento e alguma informação adicional proveniente do evento.

Quando se indica um elemento a ser adaptado através da função `adapt`, é feito um registo de um *eventhandler*[9] para esse elemento, associando-lhe uma função que recolhe estas informações. A definição da função `addEventListener` é a seguinte:

```
addEventListener(type,listener,useCapture);
```

type Tipo de evento que se deseja capturar, por exemplo, *click*, *scroll* ou *focus*.

listener Função que vai lidar com o evento quando este for capturado.

useCapture Valor booleano que no caso de ser *true* indica que o *listener* apenas deve ser invocado durante a fase de captura do evento, ou no caso de ser *false*, o mais comum, que o *listener* deve ser invocado quando o elemento é o real gerador do evento ou quando o mecanismo de propagação de eventos atinge o elemento.

Sempre que os eventos indicados ocorrerem, a função indicada como *listener* processa o evento construíndo uma mensagem de notificação que envia ao *Broker*. Um exemplo da aplicação da função `addEventListener` é:

```
...
element.addEventListener("scroll",register_event,false);
element.addEventListener("click",register_event,false);
element.addEventListener("focus",register_event,false);
...
```

Neste extracto é mostrado como são registados três *listeners* sobre o elemento `element`, que correspondem à função com o nome `register_event` para os eventos *scroll*, *click* e *focus*.

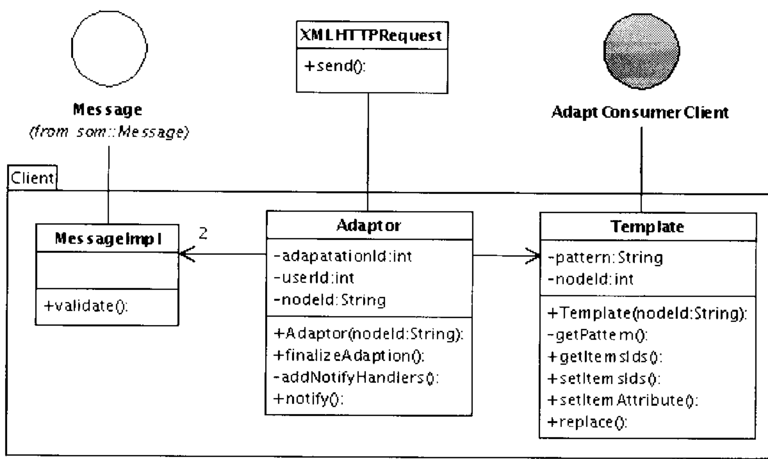


Figura 4.5: Diagrama de classes do Cliente.

4.3.4 Biblioteca de adaptação JavaScript

A biblioteca que forma a extensão a colocar no cliente assenta na estrutura que se pode ver na figura 4.5. Existem três classes, cada uma responsável por uma das tarefas que vimos: processamento de pedidos, criação de mensagens e manipulação da DOM. A classe `Adaptor`, é a responsável por receber os pedidos e respostas de adaptação e definir os eventos que serão usados como notificações. Usando a classe `MessageImpl` constrói as mensagens do tipo `adapt` e `notify` usando os métodos específicos para cada elemento das mensagens. Para a leitura de dados da página e colocação posterior dos resultados da adaptação usa a classe `Template` que contém métodos para processamento da *template*, que indica os dados a recolher e como os colocar no código da página novamente.

A criação do "interface funcional" `AdaptConsumerClient`, a cinzento, resulta da classe `Template` responsável por processar os conteúdos.

4.4 Adaptador

A criação de um Adaptador serviu principalmente para que fosse possível ter uma implementação funcional e de teste da *framework*. Para além disto serviu também como forma de obter um mecanismo de ligação com implementações diversificadas de Adaptadores, principalmente em linguagens de *scripting*. O resultado foi um **Meta-Adaptador** que agrega diversos adaptadores, sejam eles produzidos na mesma linguagem e embebidos - **adaptadores internos** - ou em linguagens de *script* - **adaptadores externos**. Em ambos os casos a intenção é sempre a mesma: processar os dados das mensagens que chegam via *Broker* e produzir mensagens de resposta. Estas duas hipóteses mantêm por um lado maior especificidade e por outro maior expansibilidade.

Como se tratava de um projecto de investigação, e os potenciais utilizadores, duma primeira versão, estarão interessados em aplicar programas já implementados ou em desenvolvimento, deu-se nesta fase maior importância à segunda possibilidade.

O Adaptador é responsável pela adaptação propriamente dita. É ele que tem acesso aos métodos que permitem produzir conteúdos adaptados ou adaptar os próprios conteúdos recebidos. Para realizar adaptações o Adaptador necessita de ter processos para fazer esse trabalho, seja através de código exclusivamente para esse propósito seja através da extensão do Adaptador para poder ter acesso a esses métodos.

4.4.1 Adaptadores internos

A ideia fundamental dos adaptadores internos, é permitir produzir adaptadores mais eficientes e dedicados e que estão integrados de alguma forma no Adaptador. Esta solução permite disponibilizar à partida adaptadores já funcionais e que realizam funções básicas, como por exemplo um adaptador *null*, ou identidade. O adaptador *null* permitiria realizar pedidos de adaptação vazios que não teriam resposta, ou seja, que não realizariam qualquer adaptação, e o adaptador identidade permitiria devolver uma adaptação igual ao próprio pedido, isto é, seriam devolvidos os dados exactamente

como foram enviados. Haveria comunicação, processamento mas nenhuma alteração. Estes adaptadores servem fundamentalmente para testes e para que seja possível verificar a funcionalidade de determinada instalação antes de definir adaptadores complexos.

A par destes, outros adaptadores podem ser incluídos criando as suas próprias classes e extendendo o modelo existente. Com esta solução não há limitações à criação e às possibilidades de adaptação que um *site* poderá usufruir.

Esta vertente faz sentido numa situação de utilização intensiva, onde a *performance* é importante para a qualidade do serviço e para a resposta em tempo útil que o utilizador espera.

Numa primeira versão de um protótipo que implementamos, este era o método escolhido para o Adaptador de teste. Um Adaptador simples que ordenava uma lista de itens em função da sua popularidade junto do utilizador. Esta noção de popularidade era conseguida pela contagem de cliques que o utilizador gerava sobre cada um dos itens, sendo depois a lista reordenada por ordem decrescente para reflectir essa popularidade. Assim, sempre que determinado utilizador voltava a esse *site* era-lhe apresentada a mesma lista, mas sempre pela ordem da sua preferência.

4.4.2 Adaptadores externos

Como os criadores de adaptadores são muitas vezes pessoas que não são programadores por definição mas sim investigadores, e como tal querem muitas vezes validar as suas teorias, seria penoso pedir que implementassem programas complexos para serem integrados no Adaptador. Para além dos eventuais atrasos que isso provocaria na verificação de um exemplo, rapidamente a vontade de criar novos adaptadores ficaria afectada.

Com o intuito de eliminar estes problemas e aumentar a vontade dos investigadores de criarem os seus próprios programas, criamos um mecanismo que permitisse carregar programas externos em qualquer linguagem. Os programas apenas necessitam de obedecer a determinadas regras, regras essas que servem para homogeneizar o compor-

tamento esperado dos *inputs* e *outputs*. Estas restrições não implicam complexidade na implementação dos programas, mas apenas definem um conjunto de acções, que são esperadas de um programa, e como o *input* e *output* deve ser feito. Ainda assim, foi dada alguma liberdade para essa implementação, sendo possível definir parâmetros de acordo com as necessidades de cada programa.

A base por trás deste método é o lançamento de um processo por parte do Adaptador que interage com o programa externo através dos canais de *input* e *output*. Na realidade, é uma simulação daquilo que aconteceria se executássemos o programa através da linha de comandos e interagíssemos com ele, enviando-lhe o *input* e recebendo o *output*. Isto é conseguido recorrendo à classe `Runtime`, que permite criar processos e aceder aos seus canais de comunicação.

Tendo esta interacção disponível, foi apenas necessário criar um protocolo para a comunicação com os processos, de modo a definir uma sequência de execução. Para isso, criou-se um ficheiro com toda a configuração que qualquer processo com funções de Adaptador deverá ter. O ficheiro está em XML e está estruturado em secções com informações que permitem ao Adaptador saber como executar o programa, quais os canais de leitura e escrita, quais os comandos de inicialização, quais os comandos de finalização e quais os comandos de pesquisa e iteração. Cada um deles com parâmetros que podem ser definidos como se achar adequado. Por exemplo, os parâmetros possíveis de definir são o que deve ser considerado como um valor de falso ou verdade para aquele determinado programa.

Tomemos então como exemplo o adaptador externo que foi criado na linguagem de programação *Perl*.

Este adaptador tinha como função consultar a base de dados, executar uma *query* SQL, onde era criado uma lista com os *links* visitados ordenados alfabeticamente e sem repetições, representando o histórico, e devolver o resultado dessa *query*.

A primeira definição é a da linha de comandos que deve ser usada para lançar o processo. Neste caso, e porque o adaptador foi implementado em *Perl*, foi necessário usar alguns parâmetros na linha de comandos para que fosse possível ter uma sessão

interactiva do programa[4].

```
<commandLine separator="," >
 perl, -e, $|=1; while(&lt;STDIN&gt;) { eval $_ ;}
</commandLine>
```

A segunda definição são os comandos de inicialização que devem ser feitos, que neste caso consistem em carregar um módulo que contém o programa a usar e realizar a ligação à base de dados.

```
<initCommands>
 use PerlAdapter;&amp;PerlAdapter::connect_db;
</initCommands>
```

Como os Adaptadores estarão vocacionados para a utilização de uma base de dados, pode-se especificar o processo a executar sobre esta, podendo passar ao programa o parâmetro de adaptação definido na função `adapt` no cliente. Neste caso:

```
<queryCommand name="queryString" >
 &amp;PerlAdapter::query_db(queryString);
</queryCommand>
```

Para terminar o processo e eventuais acções a tomar, devem por exemplo ser terminadas as ligações com a base de dados:

```
<termCommands>
 &amp;PerlAdapter::disconnect_db;exit;
</termCommands>
```

[4] Nestes extractos é possível observar alguns caracteres substituídos pelos seus códigos em XML, casos por exemplo do < e > que foram substituídos por `<` e `>` respectivamente ou do & que foi substituído por `&`.

A definição seguinte especifica as propriedades que são necessárias para construir a lista de itens da resposta. Para construir a mensagem de retorno é necessário saber quais os elementos que se devem colocar, logo, aqueles sobre os quais o adaptador deve devolver informação.

```
<getPropertyNames separator="," >
 page_id,url,label
</getPropertyNames>
```

Para construir n linhas da mensagem é necessário saber quando terminar, assim, é possível saber se existem mais linhas a processar ou se o adaptador já devolveu toda informação disponível. Para isso usa-se a seguinte definição:

```
<hasMoreItems true="1" false="0" >
 &amp;PerlAdapter::has_more_rows;
</hasMoreItems>
```

Em determinadas situações pode ser necessário saber como avançar nos dados, nomeadamente quando toda a informação de um elemento já foi recebida e é necessário recomeçar todo o processamento. No adaptador desenvolvido esta função não era necessária pelo que ficou vazia.

```
<selectNextItem>

</selectNextItem>
```

Finalmente, e depois de ter definido como lidar com as propriedades, é necessário obter os valores das propriedades que foram especificadas no elemento `getPropertyNames`:

```
<getItemProperty name="propertyName" >
 &amp;PerlAdapter::get_value_of(propertyName);
</getItemProperty>
```

Com estas definições é possível executar o adaptador em *Perl* e sequencialmente obter os valores das várias propriedades para ir construindo a mensagem de retorno com a adaptação.

Nos casos em que determinados elementos da configuração não são aplicáveis o seu conteúdo ficará vazio, como é o caso do elemento `selectNextItem`.

A partir deste momento qualquer investigador pode desenvolver um Adaptador na sua linguagem preferida para ser utilizado com a *framework*. Basta que o seu adaptador tenha funções que correspondam às acções previstas pelo Adaptador.

4.4.3 Implementação

A figura 4.6 mostra a estrutura do Adaptador que foi implementado de forma parcial tal como o *Broker*. Também aqui a gestão de configuração é estática e não há processamento de excepções. O Adaptador baseia-se na utilização de padrões de desenho de criação - ***factories*** - para obter classes que se responsabilizem pelos adaptadores externos e internos. Estas decisões são tomadas a nível da classe `Adaptor`, que recebe as mensagens via *web service* e que ao processá-las cria, através da classe `AdaptorFactory`, um Adaptador para lidar com o pedido. Mais uma vez está presente uma classe `MessageImpl` que representa cada mensagem e permite manipulá-la de acordo com as necessidades. A classe `Adaptor` lida também com as configurações de cada Adaptador, passando a cada Adaptador construído o seu respectivo ficheiro de configuração, isto no caso dos adaptadores externos.

O Adaptador pode ainda ser administrado via *web services*, enviando mensagens de administração. Isto permite, por exemplo, criar novos ficheiros de configuração para os Adaptadores. Existe também a possibilidade de utilização de uma página, onde algumas informações sobre os Adaptadores são apresentadas permitindo a sua

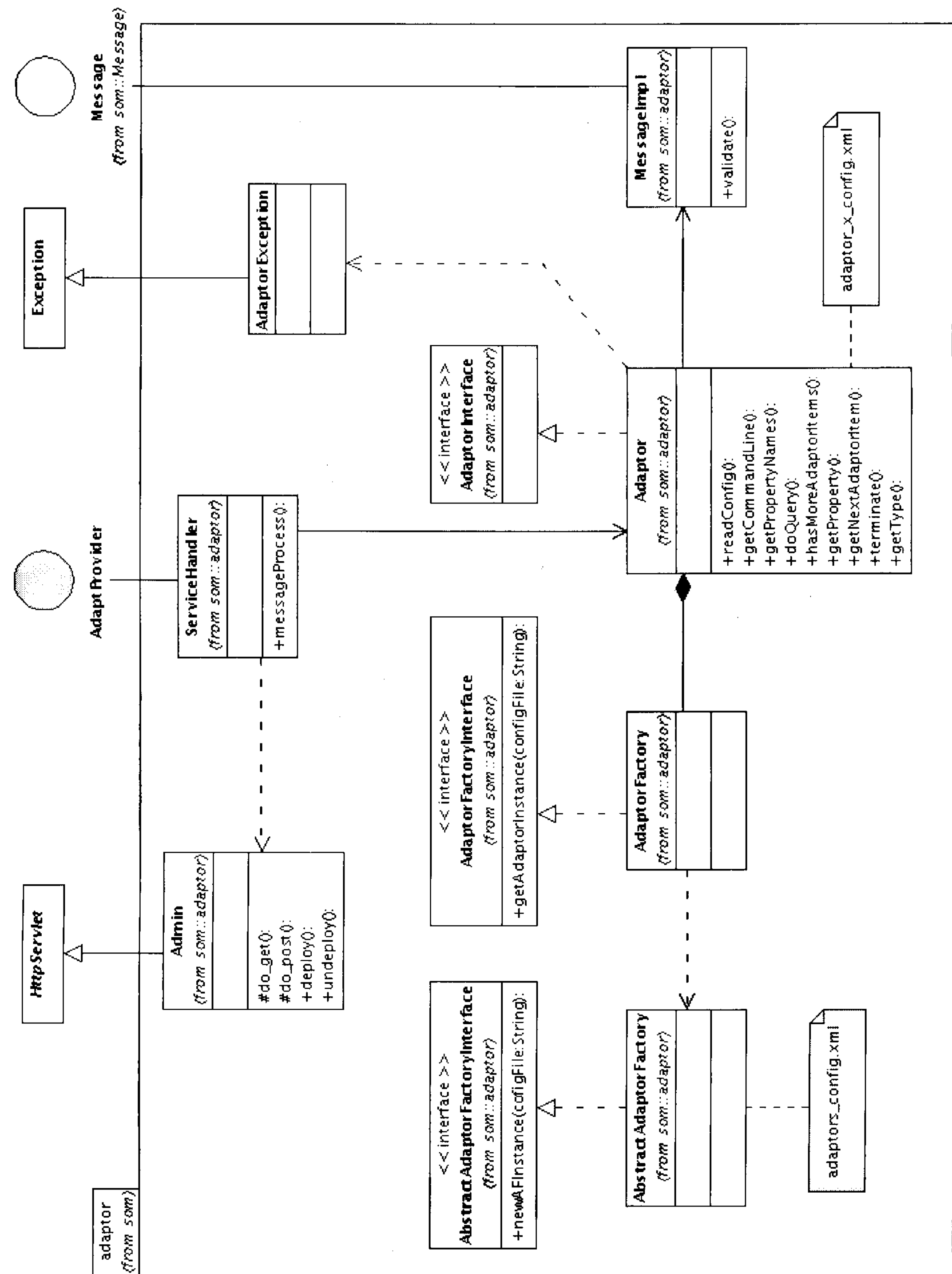


Figura 4.6: Diagrama de classes do Adaptador.

activação e desactivação.

No caso do "interface funcional" `AdaptProvider`, definido na figura 4.1, e apesar de no diagrama de classes este se encontrar ligado à classe `ServiceHandler`, ele existe pela definição de serviço WSDL que é gerada para este serviço. É o WSDL que providencia as ligações ao defini-las, permitindo que0 outras aplicações ou serviços o utilizem para obter as suas adaptações.

4.5 Mensagens

Na implementação que foi feita podemos ver, observando os diagramas 4.4, 4.5 e 4.6, que todos os componentes implementam um interface para representar as mensagens. Esse interface foi criado com o intuito de homogeneizar processos e acelerar a implementação de funções que são comuns aos três componentes. Assim, todas as classes `MessageImpl` que vimos até agora implementam o interface `Message` que define um conjunto de métodos necessários às mensagens. Este interface é altamente especializado na construção, leitura e serialização de mensagens XML de acordo com a especificação dada. Existem uma série de métodos, facilmente identificáveis pelos prefixos `create` e `get`, para a criação e leitura dos elementos pré-definidos para as mensagens. Existem ainda métodos de controlo para permitir por exemplo iterações. Como podemos ver na figura 4.7, os métodos para construção de mensagens são:

`createPage` - cria um elemento XML `page` com os respectivos atributos.

`createUser` - cria um elemento XML `user` com os atributos de identificação do cliente e informações relacionadas.

`createAdaptor` - cria um elemento XML `adaptor` com os atributos de identificação, estado e localização do adaptador.

`createQuery` - cria um elemento XML `query` que contém os dados passados no parâmetro `query` da função `adapt` no cliente.

```
<< interface >>
Message
(from som)
+createPage(id:String, url:String,label:String):
+createUser(id:String):
+createAdaptor(id:int, status:String, url:String):
+createQuery(queryString:String):
+createItem(id:String,url:String, label:String, order:int, higligth:int,visible:int):
+createEvent(source:String,type:String,value:String):
+createText(text:String):
+getMessage():String
+getPageAttr(attributeName:String):
+getUserId():String
+getAdaptorAttr(attrName:String):
+getQuery():String
+hasMoreItems():Boolean
+getNextItem
+getAdminAction():
-getXPath
```

Figura 4.7: Diagrama de classes do interface `Message`.

`createItem` - cria um elemento XML `item` com os atributos de identificação, URL, nome e características de estilos.

`createEvent` - cria um elemento XML `event` com os atributos do local de origem, tipo e valor.

`createText` - cria um nó XML de `text` com o conteúdo especificado.

E os métodos para processamento são:

`getMessage` - devolve uma mensagem como uma string, isto é, serializa-a.

`getPageAttr` - devolve o atributo especificado do elemento `page`.

`getUserId` - devolve a identificação do utilizador que se encontra no atributo `id` do elemento `user`.

`getAdaptorAttr` - devolve o atributo especificado do elemento `adaptor`.

`getQuery` - devolve o conteúdo do elemento `query`.

`hasMoreItems` - verifica se na mensagem ainda há mais itens a serem processados.

`getNextItem` - retorna a linha, nó XML, correspondente ao próximo item a processar.

`getAdminAction` - retorna a acção de administração a realizar no Adaptador.

Há no entanto algo a salientar quando dizemos que o interface é comum aos três componentes. Na realidade, isso não acontece porque os componentes *Broker* e Adaptador são implementados em Java e o Cliente é implementado em JavaScript que não possui realmente um objecto Interface. Apesar disso, a lógica de implementação é a mesma e a implementação em JavaScript e em Java seguiram os mesmos padrões, mantendo nomes de métodos e o modo como são usados, o que nos permite dizer no fim que ambos partilham um mesmo interface para as implementações dos objectos `MessageImpl`.

De qualquer forma, e dado que os *sites* podem estar implementados com outras tecnologias que não o JavaScript, definiu-se um interface que pode ser usado por outra linguagem, mantendo a homogeneidade da implementação a este nível.

Capítulo 5

Avaliação e testes

Para avaliar a solução proposta implementamos dois protótipos que utilizam a implementação da *framework* descrita no capítulo anterior.

O primeiro protótipo destinou-se exclusivamente ao estudo de algumas tecnologias e verificação das suas capacidades. Esse protótipo inicial foi completamente abandonado, apesar de ter sido talvez o protótipo mais fiel a um sistema de adaptação na vertente da reformatação. No entanto, e como era primordialmente um sistema de experimentação, não seguia totalmente a arquitectura aqui apresentada.

O segundo protótipo, já com a arquitectura final em mente, representa mais fielmente o que se pretende atingir com a *framework* a todos os níveis. Contudo, a abordagem escolhida centra-se na adaptação por recomendação. Este protótipo, do qual se pode ver um *screenshot* na figura 5.1 e que está disponível no endereço `http://www.niaad.liacc.up.pt/site-o-matic/demo`, trata-se de uma página para leitura de *man pages unix* em formato HTML. A página consiste numa *searchbox* que permite indicar a página a visualizar, uma frame onde é visualizada a página escolhida e uma coluna lateral à esquerda com um conjunto de informações: histórico e recomendações. O histórico mostra as páginas que o utilizador já viu permitindo uma navegação rápida para páginas anteriores. As recomendações apresentam uma lista de páginas que possam estar relacionadas com as visualizações já feitas.

Estas duas listas são os conteúdos que são gerados pelos Adaptadores e inseridos no cliente. No caso particular das recomendações não existe uma verdadeira recomendação, pois não foi produzido um adaptador mais complexo que criasse o relacionamento das páginas existentes com as páginas vistas. Para efeitos de teste e para ter uma simulação de dois adaptadores numa mesma página, aquilo que se fez foi usar o mesmo adaptador usado para o histórico mas com uma ordenação diferente. Com este protótipo foram realizados testes de *performance* e de *stress* num ambiente controlado e sempre com o mesmo estado inicial. Para criar este ambiente tomamos uma série de medidas:

- Desactivação da *cache* no *browser*, para que os pedidos fossem sempre efectuados na sua totalidade ao servidor, não podendo assim encurtar tempos de acesso;
- Limpeza da base de dados entre cada teste, para evitar que a acumulação de dados pudesse interferir com os tempos de acesso de teste para teste;
- Realização de um acesso inicial, para permitir a inicialização de todos os processos e eliminar o tempo gasto dos restantes tempos de utilização;
- Utilização de dois ou mais computadores para emular, do ponto de vista do cliente, a sua utilização real. Numa situação de aplicação real o Cliente e restantes componentes não estarão nas mesmas máquinas;
- Utilização de ferramentas automáticas, sempre que possível, para a realização dos testes, evitando assim possíveis efeitos paralelos e de modo a permitir repetições dos mesmos.

Nos testes foram usados três máquinas diferentes todas ligadas em rede numa LAN. O esquema assentou num computador fixo ligado por cabo a um *router wireless* que serviu de servidor, e dois portáteis com ligações *wireless* representando os clientes.
As especificações dos computadores podem ser vistas na tabela 5.1. A configuração do servidor incluía ainda a versão 5.5.17 do servidor aplicacional Apache Tomcat e a versão 1.5.0_07 do J2SE.
Para realização dos testes de *performance* foi instalado nos clientes uma ferramenta

Figura 5.1: *Screenshot* do segundo protótipo num momento de utilização.

Computador	CPU	RAM	Rede	*Browser*
Servidor	Athlon XP 2800+	1GB	100Mbps	-
Cliente 1	Pentium M 1,6 GHz	512MB	54Mbps	Firefox 1.5.0.6
Cliente 2	Pentium M 1,8 GHz	1GB	54Mbps	Firefox 1.5.0.6

Tabela 5.1: Especificações dos computadores usados para testes.

automática de nome Selenium[26] e para os testes de *stress* foi usado um *script* em *bash*.

5.1 Testes de *performance*

Os testes de *performance* consistiram na utilização de um cliente que efectuou n pedidos, como se de um utilizador real se tratasse. Para esse cliente foram variados o número de Adaptadores existentes na página de forma a perceber a sua influência no processo. As medições efectuadas foram os tempos gastos por cada componente nas suas funções.

Para os testes decidimos usar uma ferramenta que permitisse realizar de forma automática as n repetições, como as ferramentas de testes unitários. As ferramentas analisadas para esse efeito foram: JMeter[17], JWebUnit[19], JSUnit[18] e htmlunit[14] normalmente usadas para testes unitários e que permitem reproduzir o comportamento desejado. No entanto, devido a estarmos a usar AJAX e manipulação da DOM, não foi possível usá-las. Ou porque não suportavam JavaScript, logo não despoletando os eventos necessários para o pedido de adaptação, caso do JMeter, ou porque as implementações dos interpretadores de JavaScript eram limitadas deixando as funções usadas para o envio de pedidos ou manipulação da DOM sem possibilidades de utilização, casos do JWebUnit e JSUnit.

Finalmente decidimos usar a ferramenta Selenium. Enquanto as ferramentas usualmente utilizadas para os testes unitários emulam o *browser*, o Selenium utiliza o próprio *browser* para realizar os testes, deste modo o suporte dado do lado do cliente é o do *browser*, não ficando dependente da sua própria implementação.

Este software permite instalar no *browser* um *plugin* que grava as acções do utilizador e posteriormente reproduzi-las automaticamente. Além deste *plugin*, o Selenium tem uma versão Remote Control (Selenium-RC)[27] que permite através da programação, em várias linguagens, manipular um *browser* e o seu conteúdo. O Selenium-RC funciona lançando um servidor através do qual são feitos os pedidos (*proxy*) a uma

janela do *browser* com o endereço desejado. Depois, e através do programa criado, podem ser feitos, por exemplo, cliques nos botões e *links* ou preenchidos formulários presentes na página. É também possível executar uma série de testes que permitem inquirir e validar, à semelhança dos testes unitários, determinadas condições sobre a página.

O teste com o Selenium consistiu na criação de um pequeno programa em Java, que fazendo uso do Selenium-RC executa 20 pedidos a diferentes *man pages*. Isto funciona indicando qual o campo a preencher com o nome da *man page* e qual o botão a clicar para realizar o pedido, como se pode ver no seguinte extracto:

```
....
String[] pageList = { "ls", "bc", "man", "vi", "cat", "find",
                      "du", "dd", "login", "touch", "time",
                      "ln", "diff", "cp", "chmod", "chown",
                      "mv", "awk", "grep", "rmdir"};
....
selenium.type("page", pageList[i]);
selenium.click("submit");
....
```

Aqui `selenium` corresponde a um objecto do tipo `Selenium` que possui métodos `type` e `click`, para escrever e clicar respectivamente. Tudo o que é necessário fazer, é dizer ao primeiro qual o nome do elemento e o conteúdo a usar e ao segundo o nome do botão a clicar.

Com estes métodos podemos manipular da forma que entendermos um *site*, no nosso caso o visualizador *web* de *man pages*.

Nestes testes tivemos de fazer algumas opções devido ao tipo de página que estávamos a usar. Como utilizamos AJAX para criar dinamicamente o conteúdo dos menus de Histórico e Recomendação da nossa página é impossível realizar testes ou indicar cliques sobre algo que não existe na estrutura HTML, mas que é colocada à poste-

Universidade do Porto
Faculdade de Ciências
DEPARTAMENTO DE
CIÊNCIA DE COMPUTADORES
BIBLIOTECA

rior. Assim, optamos por realizar os testes apenas com consultas, ou seja como se o utilizador nunca usasse os *links* laterais. Isto tem dois efeitos. Por um lado nunca são geradas notificações, o que não é problemático, pois são comunicações num só sentido e para povoar base de dados. Por outro, as adaptações geradas pelo carregar de uma página contida num *link* nunca são feitas. No entanto, como isso corresponde exactamente ao mesmo que realizar um pedido de consulta é negligenciável.
Foi também necessário introduzir um espaçamento nos pedidos para os tornar mais reais. Como estamos a usar o AJAX para construir os menus laterais da aplicação, a sequência de pedidos ultrapassava rapidamente essa construção, não sendo possível ver os menus até ao fim dos testes, tal a rapidez com que eram feitos. A essa velocidade a página era obrigada a ser redesenhada constantemente nessas secções, acabando por aparecerem em branco até ao último pedido. Ao introduzir um intervalo de dois segundos entre pedidos, tornámos a simulação mais real, pois o utilizador por norma está interessado em ler a página e não faz pedidos a uma tão grande velocidade.
Os tempos e locais que escolhemos para fazer essas medições foram os seguintes:

- Tempo gasto pelo *Broker* para recepção de uma mensagem, envio para o Adaptador e depois de recebida a resposta, devolução ao Cliente;
- Tempo gasto entre a chegada de uma mensagem ao Adaptador, respectivo processamento, isto é, adaptação e devolução ao *Broker*;
- Tempo gasto na comunicação SOAP entre *Broker* e Adaptador;
- Tempo gasto para realizar inserções na base de dados.

Estes tempos correspondem ao tempo gasto dentro de cada um dos componentes. No caso particular do tempo gasto na base de dados devemos ter em conta que esse processo está dentro do *Broker*, pelo que os valores apresentados para o *Broker* excluem já o tempo gasto na inserção de dados.
Os tempos médios que obtivemos podem ser consultados na gráfico da figura 5.2. No gráfico podemos observar que o aumento do número de Adaptadores apenas tem in-

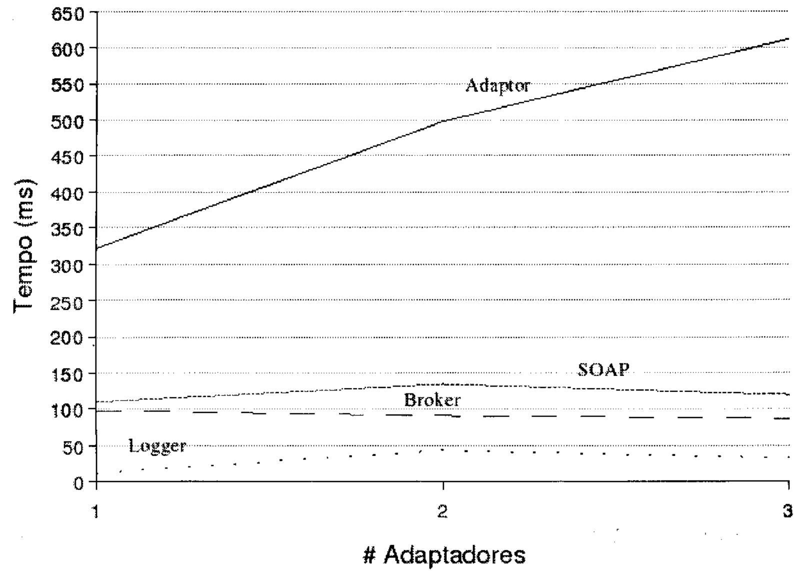


Figura 5.2: Tempos médios de acesso, em milissegundos, para um utilizador com um e dois adaptadores.

fluência sobre o tempo medido no Adaptador. Isto explica-se pelo facto do Adaptador estar a recorrer a um processo externo e usar o mecanismo que já vimos de *input-output* para a troca de dados. O aumento de Adaptadores aumenta o número de chamadas a estes processos e consequentemente o tempo de resposta de cada um deles. Os restantes tempos não têm alterações significativas, mantendo-se sensivelmente iguais à medida que o número de Adaptadores foi aumentando. Isto significa que do ponto de vista de *performance* da utilização, a *framework* consegue responder da mesma maneira para diferentes configurações de um *site.* O objectivo de ter um *Broker* capaz de realizar processos de modo eficiente, foi neste capítulo, conseguido.

5.2 Testes de carga

Os testes de carga serviram para perceber qual o comportamento dos componentes quando sujeitos a um elevado número de acessos. As medições são as mesmas que nos testes de *performance*, mas com a diferença de deixarmos de lado o cliente. Isto

faz sentido se pensarmos que o processamento de cada cliente é um exclusivo seu, ou seja, o processamento que for feito por cada cliente não vai ter influência sobre outros clientes que estejam a usar o sistema.

Assim, os testes focaram-se em fazer n pedidos com três computadores e registar os tempos do processamento do *Broker*, do *Logger*, do Adaptador e da comunicação entre *Broker* e Adaptador. Os locais de medição foram exactamente os mesmos que nos testes de *performance.*

Para realizar os testes usamos um *script* em *bash* com um ciclo que faz POST n vezes de uma mensagem, simulando assim n pedidos dos vários clientes. O teste realizado consistiu numa sequência incremental de pedidos. O primeiro teste foi feito com 100 pedidos, o segundo com 200 e assim sucessivamente, com acréscimos de 100, até atingir os 1000 pedidos. Cada iteração realizou dois `POST`s por forma a simular a existência de dois adaptadores configurados. Isto significa que na iteração de 100 pedidos foram feitos 600 `POST`s ($3clientes * 2adaptadores * 100pedidos$) e na iteracção 1000, 6000 ($3clientes * 2adaptadores * 1000pedidos$).

Importa referir que apesar de não existir cliente, o processamento feito e o circuito das mensagens é exactamente o mesmo do caso em que o cliente está presente. Os pedidos consecutivos são pedidos `POST` de uma mensagem escolhida aleatoriamente, mas que corresponde a uma mensagem igual à que um cliente faria quando faz um pedido de adaptação. Os valores do gráfico na figura 5.3, mostram de um modo geral que a *performance* se mantém idêntica nas várias situações. O aumento de pedidos apenas tem um impacto significativo nos tempos medidos no Adaptador. Aqui também relacionado com o facto de ser utilizado um processo mais complexo para interagir com os adaptadores externos.

Com estes dados podemos concluir que os componentes da *framework* têm um comportamento constante mesmo de baixo de maiores cargas, correspondendo à necessidade imposta pelos objectivos de não introduzir efeitos colaterais na utilização desta ferramenta num *site*, em particular nos tempos de reposta ao utilizador. Se observarmos no gráfico da figura 5.4 qual a distribuição percentual de processamento em todo o sistema vemos que a grande fatia está no Adaptador, pelas razões que já abordamos.

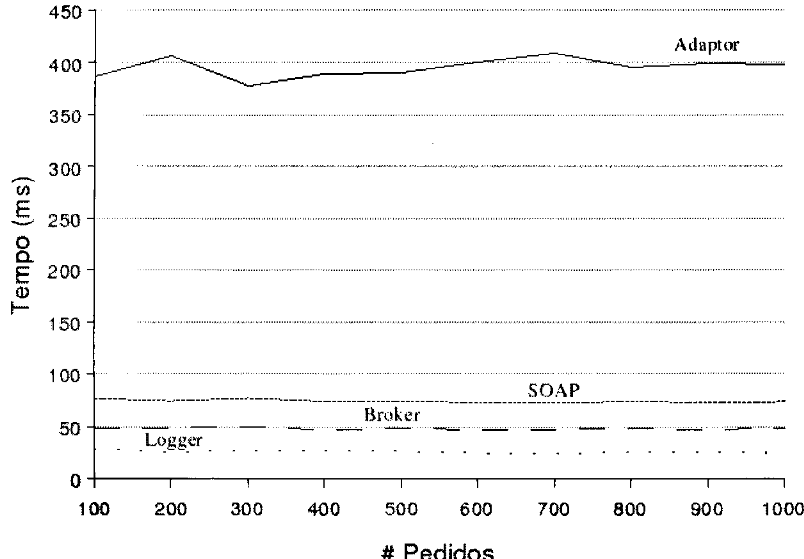


Figura 5.3: Tempos médios em milissegundos, para 3 clientes e uma configuração de página com 2 adaptadores, com o número de pedidos a variar entre 100 e 1000.

Vale a pena lembrar, face a esta distribuição, que o *Broker* para além de realizar as inserções na base de dados, que ocupa cerca de 5% do processamento, e fazer a comunicação SOAP com os Adaptadores, cerca de 14%, tem também de processar mensagens e validá-las, tudo incluído nos restantes 9%. Quer isto dizer que o Broker gasta apenas, neste protótipo, 9% do processamento total. As restantes tarefas ficam dependentes da eficiência dos Adaptadores, do sistema de gestão de base de dados e do *browser* do utilizador, responsável por realizar as alterações à página.

5.2.1 Avaliação qualitativa

Para podermos avaliar qualitativamente o protótipo seria necessário realizar algum tipo de estudo com utilizadores reais que usassem a aplicação num ambiente controlado e pudessem registar as sua opiniões de uma modo sistemático. Nessa avaliação as perguntas realizadas incidiriam sobre as sensações que puderam sentir com a utilização de um *site* não adaptado e adaptado, fossem elas de tempo ou qualidade de informação. Essa hipótese, apesar de interessante, não foi implementada pela dificuldade de ter um

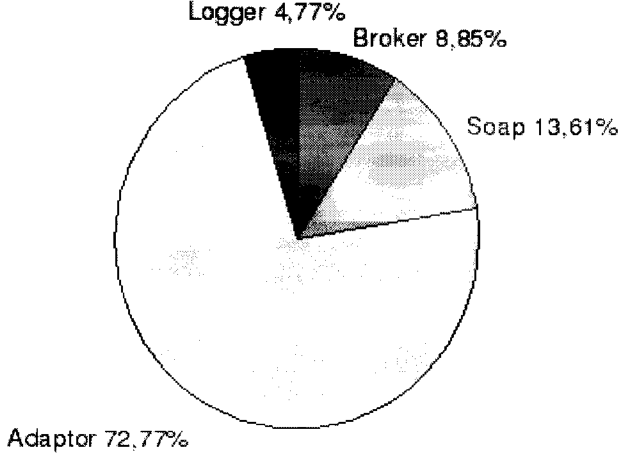


Figura 5.4: Percentagens de processamento para cada componente.

conjunto de utilizadores sujeitos a este teste, por existir um Adaptador muito básico que pode não ter grande utilidade e pelo facto de o próprio protótipo poder influenciar negativamente a opinião do utilizador devido à sua simplicidade.

Um utilizador que usasse o protótipo poderia responder negativamente a determinadas questões, pelo simples facto de a aplicação não lhe parecer interessante à primeira vista, mesmo pedindo-lhe para se focar apenas sobre aspectos relacionados com a adaptação.

Restou-nos fazer a avaliação qualitativa do protótipo tendo em conta a experiência que obtivemos da sua utilização. Como todos os *sites* que usam Ajax, a aplicação desta tecnologia nestas páginas está dependente de muitos factores. Está dependente da velocidade da máquina, da memória, do *browser* usado e até da complexidade da página. Existem muitos *sites* que usam esta tecnologia e que têm bastante sucesso. *Sites* como o Writely[33], Google Maps[10], Google Suggest[11], A9[1], Instant Domain Name Search[16], Netflix top 100[23], usam o Ajax para criar interfaces dinâmicos, carregar imagens parciais, fazer pré-validação das pesquisas ou inserção de conteúdo em função de um item seleccionado. É exactamente o mesmo que o protótipo faz e pelo que nos foi dado a observar com bastante destreza e sempre sem que o utilizador possa deixar de interagir com a página.

Durante os testes de *performance* que efectuamos foi possível observar que a página

era carregada com facilidade e os conteúdos preenchidos com recurso ao Ajax correctamente inseridos e sem causar nenhum tipo de efeito secundário na resto da página. Existe por vezes, caso o *browser* tenha já várias páginas abertas ou a própria máquina esteja sobrecarregada, algum atraso mais perceptível, mas ainda assim perfeitamente aceitável e nada de muito diferente daquilo que se pode observar em *sites* que usam esta tecnologia

Em termos de valores de tempo que foram possíveis apurar manualmente, dado que o facto de usar Ajax dificulta a compilação de valores entre pedidos, o cliente demora entre 20 a 15 milissegundos a fazer o *display* da adaptação que lhe chega. Este valor é bastante baixo o que mostra a capacidade do Ajax em realizar estas alterações.

Capítulo 6

Conclusão e trabalho futuro

O objectivo deste trabalho, como tarefa do projecto Site-O-Matic, era criar uma infra-estrutura que introduzisse adaptabilidade em *sites* existentes ou a desenvolver de raiz. A solução encontrada foi a criação de uma *framework* que permitiu introduzir meios de adaptação sem a necessidade de implementações específicas e dependentes das aplicações. Esta solução teve como base um conjunto de objectivos:

- Independência das tecnologias web;
- Abstracção das operações de adaptação;
- Expansibilidade dos módulos de adaptação.

Isto resultou na criação de uma arquitectura com inspiração SOA que permitiu responder aos objectivos.

A *framework* é independente de qualquer gestor de conteúdos e pode ser usada por qualquer tecnologia quer do lado do cliente quer do lado do servidor, pois é apenas necessário comunicar com um ponto central que se encarrega de orientar os pedidos feitos. Além disso, os clientes não estão obrigados a qualquer tecnologia, pois apenas precisam de enviar mensagens com um formato pré-determinado. Desde que uma determinada aplicação possa falar por HTTP com o *Broker* e consiga enviar mensagens

XML assincronamente, está apta a usar a *framework*.

No que diz respeito à abstracção dos processos de adaptação, a utilização das mensagens para conterem todo o contexto e sintaxe da adaptação, permite que se deixem para os extremos da arquitectura, Clientes e Adaptadores, a sua implementação. Deste modo a *framework* assume o papel de meio de transporte entre qualquer Cliente e qualquer Adaptador.

Quanto à expansibilidade dos módulos de adaptação, a utilização de *web services* e um método bem definido para a execução de programas que possam realizar tarefas de adaptação, tomam essa responsabilidade. A utilização dos *web services*, pela sua própria definição, define um *standard* para as comunicações e ao mesmo tempo aumenta o seu domínio de acção, permitindo a interoperação em qualquer momento com outros serviços conhecidos, enquanto que o método para execução de programas externos permite desenvolvimento e testes, e favorece a disponibilidade para novas soluções de adaptação em curto espaço de tempo.

Este trabalho permitiu também abordar novos conceitos, nomeadamente na parte de recolha de dados. Esta solução fornece informações e acima de tudo dá-lhes um contexto que até agora não existia. É possível extrair dos dados compilados, noções sobre a verdadeira utilização de uma página, que não era possível até agora pela simples análise dos *logs* de um servidor. O *feedback* dado pelos dados é mais completo e isso faz com que se possa também produzir melhores adaptações.

Assim, o sucesso desta ideia parece ser comprovado pelos resultados obtidos nos testes, que indicam que a utilização desta solução não será impeditiva nem para a navegação dos utilizadores, nem para os criadores de *sites*. Os tempos obtidos em testes, mostram que as respostas, mesmo para cargas elevadas, foram sempre dadas em tempo útil, nunca se colocando como concorrentes pelos recursos, logo não produzindo efeitos negativos sobre o sistema. Vantagens que advêm da especialização dos componentes e da comunicação assíncrona.

Ao longo do trabalho as escolhas feitas foram baseadas nas características funcionais e técnicas que correspondiam aos nossos objectivos.

As maiores dificuldades que encontrámos, mesmo fazendo as escolhas que mais se ade-

quavam às nossas necessidades, estiveram relacionadas com a biblioteca de adaptação JavaScript do cliente. A criação de uma *framework* com uma forte vertente sobre o cliente levantou algumas questões e problemas durante o estudo e desenvolvimento do protótipo. Ao usar JavaScript para recolher e manipular informação do cliente, para ser depois passada ao *Broker*, cria-se um laço estreito e dependente com este e com os dados que se estiverem a usar. Este não é um problema desta implementação em particular, mas um problema recorrente nas actuais implementações de aplicações *web*. A guerra de *browsers* e *standards* a que hoje se assiste, traz alguns problemas para as implementações de aplicações *web* e que nos obrigaram também a alguma reflexão. Problemas como a manipulação da DOM[6][5] ou a utilização de objectos para criação de documentos XML, são algumas das questões que encontrámos. As diferenças de implementação de *browser* para *browser* e o não cumprimento de *standards*, obrigaram mesmo ao abandono de soluções interessantes. Contudo, isto fez também com que se pudesse filtrar mais os problemas e aumentar as plataformas suportadas.

Para o futuro, fica ainda algum trabalho para realizar, principalmente a implementação de uma solução de produção seguindo todos os desenhos idealizados. Isto é, uma solução que tenha em conta a questão de administração prevista, o processamento de excepções e Adaptadores genéricos para testes e experimentação.

Outros pontos que ficam em aberto e que serão objecto de trabalho futuro são:

- A utilização de métodos e bibliotecas compatíveis em diversos *browsers*, para permitir uma fácil integração da biblioteca JavaScript fornecida com a *framework*;
- A utilização de *templates* para especificar o modo como os dados de e para adaptação devem ser tratados no cliente, mantendo as características da página original;
- A utilização de interfaces de administração para tarefas básicas de configuração do Broker ou Adaptadores;
- A utilização de métodos para o registo de Adaptadores, como por exemplo o UDDI;

- Definir métodos de teste apropriados para aplicações que façam uso do Ajax como é o caso do protótipo criado, permitindo assim obter dados sobre o cliente duma maneira mais sistemática e menos empírica;

- Realizar testes com um grupo de utilizadores para fazer uma avaliação qualitativa, usando um *site* de produção com utilização regular.

Glossário

Framework Estrutura de suporte sobre a qual se pode realizar a implementação de uma aplicação.

Adaptador Componente com capacidades funcionais para gerar adaptações recorrendo a algoritmos.

Broker Componente da *framework* que faz a ligação entre clientes e adaptadores e regista todos e eventos e adaptações.

Cliente Componente da *framework* que é colocado do lado do utilizador que permite a comunicação com o *Broker*.

adaptador Processo externo à framework. Desenvolvido por terceiros. Para ser usadado pelo Adaptador.

cliente Conjunto formado pelo *browser* e utilizador.

cliente de adaptação Componente que requer a adaptação.

fornecedor de adaptação Componente capaz de produzir adaptações.

Apêndice A

Especificação das mensagens

A.1 DTD

A DTD foi usada inicialmente para delinear a estrutura da linguagem. esta representação auxilia a descrição e é mais facilmente perceptível por terceiros.

```
<!-- SOM Markup Language -->
<!-- SOM message definition -->
<!ELEMENT som:message (adapt|notify|admin)>
<!ATTLIST som:message
mode (request|reply) #REQUIRED
>

<!ELEMENT adapt (user,client,page,adaptor,(query|item+))>
<!ELEMENT notify (user,client,page,event+)>
<!ELEMENT admin (user,client,adaptor+)>

<!ATTLIST admin
```

```
action (list|register|status|enable|disable) #REQUIRED
>

<!-- User -->
<!ELEMENT user EMPTY>
<!ATTLIST user
id CDATA #REQUIRED
>

<!-- Client -->
<!ELEMENT client EMPTY>
<!ATTLIST client
browser CDATA #REQUIRED
ip CDATA #IMPLIED
referer CDATA #IMPLIED
>

<!-- Page -->
<!ELEMENT page EMPTY>
<!ATTLIST page
id  CDATA #REQUIRED
url  CDATA  #REQUIRED
label CDATA #IMPLIED
>

<!-- Adaptor -->
<!ELEMENT adaptor EMPTY>
<!ATTLIST adaptor
id CDATA  #REQUIRED
status CDATA #IMPLIED
```

```
url CDATA #IMPLIED
>

<!-- Query -->
<!ELEMENT query (#PCDATA)>

<!-- Item -->
<!ELEMENT item (#PCDATA)>
<!ATTLIST item
id ID #REQUIRED
url CDATA #IMPLIED
label CDATA #IMPLIED
order CDATA #IMPLIED
highlight CDATA #IMPLIED
visible (yes | no) #IMPLIED
>

<!-- Event -->
<!ELEMENT event (#PCDATA)>
<!ATTLIST event
source CDATA #REQUIRED
type CDATA #REQUIRED
value CDATA   #IMPLIED
>
```

A.2 XSD

O XSD aqui representado foi usadoa já na fase de implementação para validar as mensagens no *Broker*.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:som="http://www.niaad.liacc.up.pt/SOM"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.niaad.liacc.up.pt/SOM"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
<xs:complexType name="message">
<xs:choice>
<xs:element ref="som:adapt"/>
<xs:element ref="som:notify"/>
<xs:element ref="som:admin"/>
</xs:choice>
<xs:attribute name="mode" use="required">
<xs:simpleType>
<xs:restriction base="xs:NMTOKEN">
<xs:enumeration value="reply"/>
<xs:enumeration value="request"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
<xs:element name="message" type="som:message"/>
<xs:complexType name="adapt">
<xs:sequence>
```

```
<xs:element ref="som:user"/>
<xs:element ref="som:client"/>
<xs:element ref="som:page"/>
<xs:element ref="som:adaptor"/>
<xs:choice>
<xs:element ref="som:query"/>
<xs:element ref="som:item" maxOccurs="unbounded"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
<xs:element name="adapt" type="som:adapt"/>
<xs:complexType name="notify">
<xs:sequence>
<xs:element ref="som:user"/>
<xs:element ref="som:client"/>
<xs:element ref="som:page"/>
<xs:element ref="som:event" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:element name="notify" type="som:notify"/>
<xs:complexType name="admin">
<xs:sequence>
<xs:element ref="som:user"/>
<xs:element ref="som:client"/>
<xs:element ref="som:adaptor" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="action" use="required">
<xs:simpleType>
<xs:restriction base="xs:NMTOKEN">
<xs:enumeration value="disable"/>
```

```
<xs:enumeration value="status"/>
<xs:enumeration value="list"/>
<xs:enumeration value="register"/>
<xs:enumeration value="enable"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
<xs:element name="admin" type="som:admin"/>
<xs:complexType name="user">
<xs:attribute name="id" type="xs:anySimpleType" use="required"/>
</xs:complexType>
<xs:element name="user" type="som:user"/>
<xs:complexType name="client">
<xs:attribute name="browser" type="xs:anySimpleType" use="required"/>
<xs:attribute name="ip" type="xs:anySimpleType"/>
<xs:attribute name="referer" type="xs:anySimpleType"/>
</xs:complexType>
<xs:element name="client" type="som:client"/>
<xs:complexType name="page">
<xs:attribute name="id" type="xs:anySimpleType" use="required"/>
<xs:attribute name="url" type="xs:anySimpleType" use="required"/>
<xs:attribute name="label" type="xs:anySimpleType"/>
</xs:complexType>
<xs:element name="page" type="som:page"/>
<xs:complexType name="adaptor">
<xs:attribute name="id" type="xs:anySimpleType" use="required"/>
<xs:attribute name="status" type="xs:anySimpleType"/>
<xs:attribute name="url" type="xs:anySimpleType"/>
</xs:complexType>
```

```
<xs:element name="adaptor" type="som:adaptor"/>
<xs:complexType name="query" mixed="true"/>
<xs:element name="query" type="som:query"/>
<xs:complexType name="item" mixed="true">
<xs:attribute name="id" type="xs:ID" use="required"/>
<xs:attribute name="url" type="xs:anySimpleType"/>
<xs:attribute name="label" type="xs:anySimpleType"/>
<xs:attribute name="order" type="xs:anySimpleType"/>
<xs:attribute name="highlight" type="xs:anySimpleType"/>
<xs:attribute name="visible">
<xs:simpleType>
<xs:restriction base="xs:NMTOKEN">
<xs:enumeration value="yes"/>
<xs:enumeration value="no"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
<xs:element name="item" type="som:item"/>
<xs:complexType name="event" mixed="true">
<xs:attribute name="source" type="xs:anySimpleType" use="required"/>
<xs:attribute name="type" type="xs:anySimpleType" use="required"/>
<xs:attribute name="value" type="xs:anySimpleType"/>
</xs:complexType>
<xs:element name="event" type="som:event"/>
</xs:schema>
```

Apêndice B

Especificação dos adaptadores externos

Este ficheiro serve de referência para qualquer utilizador que queira desenvolver um adaptador externo para incluir na *framework*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- This file contains the specifications for running an external adaptor. -->
<!-- Each entry matches a function or command that must be invoked for a     -->
<!-- given task. This command sequence must be usable like in a command      -->
<!-- prompt. -->
<configs>
<!-- Comand line to run the external adaptor.        -->
<!-- Same as invoking via comand line  if it has     -->
<!-- arguments use a separator that can be specified -->
<commandLine separator=",">
commandline
</commandLine>
<!-- Commands that have to be runned to initialize    -->
```

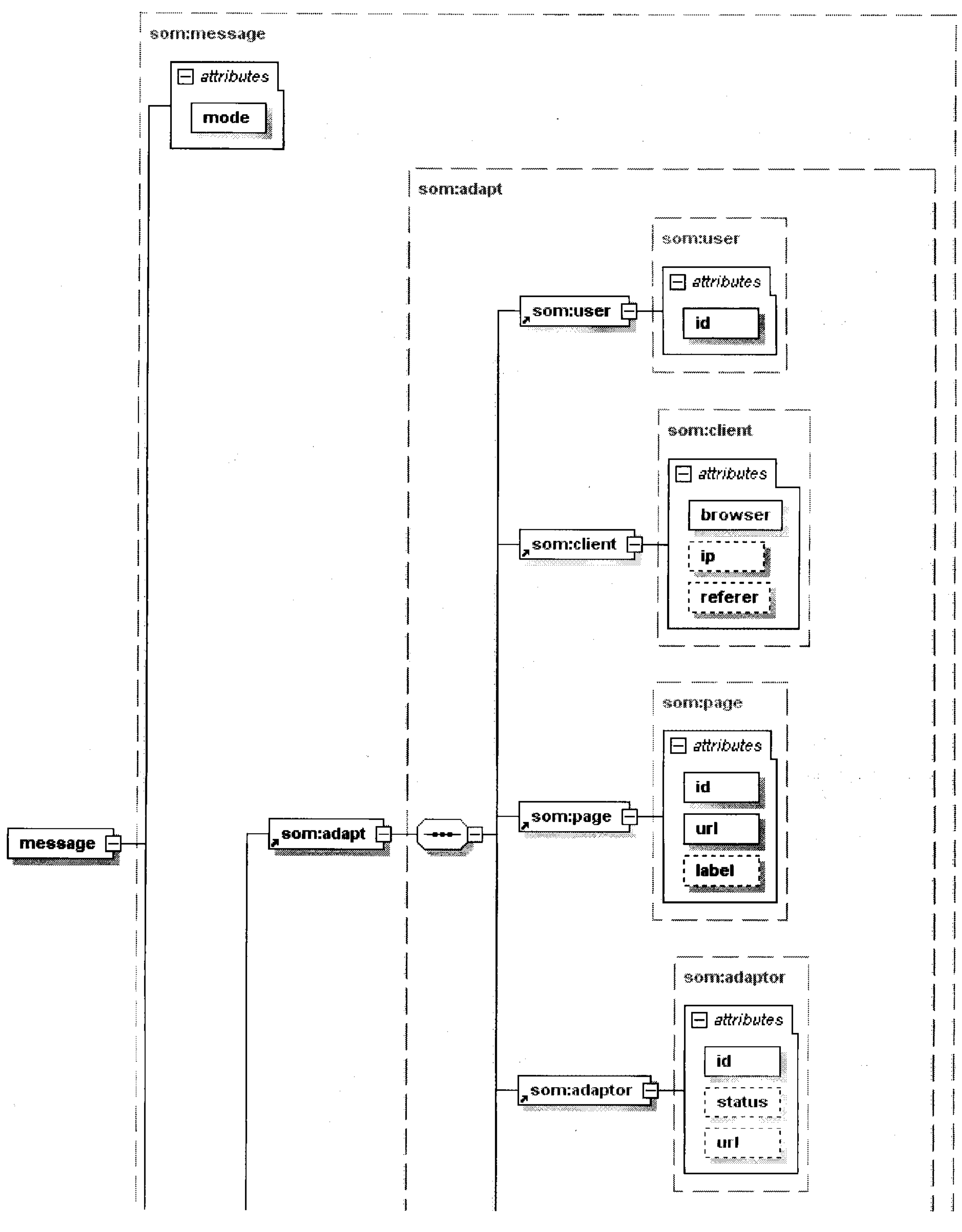
```
<!-- the adaptor for instance connect to the database -->
<initCommands>
connect()
</initCommands>
<!-- The "query" commmand of the adapter. Parameters  -->
<!-- specified in the client adapt function can be    -->
<!-- passed through a variable that can be sepcified  -->
<!-- with the attribute name                          -->
<queryCommand name="queryString">
query(querystring)
</queryCommand>
<!-- How to end the process. The comand tat should be issued -->
<!-- to terminate the process. For instance it is a bash     -->
<!-- script it should be "exit"                              -->
<termCommands>
exit()
</termCommands>
<!-- Specify the proprties that are necessary to build the  -->
<!-- items list in the reply message. Due to client         -->
<!-- limitations this cannot be changed for now. In this    -->
<!-- version the client allways expect to receive a message -->
<!-- with item entries with this properties as attributes   -->
<getPropertyNames separator=",">
page_id,url,label
</getPropertyNames>
<!-- How to know if there are more items (lines) to process. -->
<!-- Value of true and false can be specified with           -->
<!-- attributes "true" and "false"                           -->
<hasMoreItems true="1" false="0">
hasmoreitems()
```

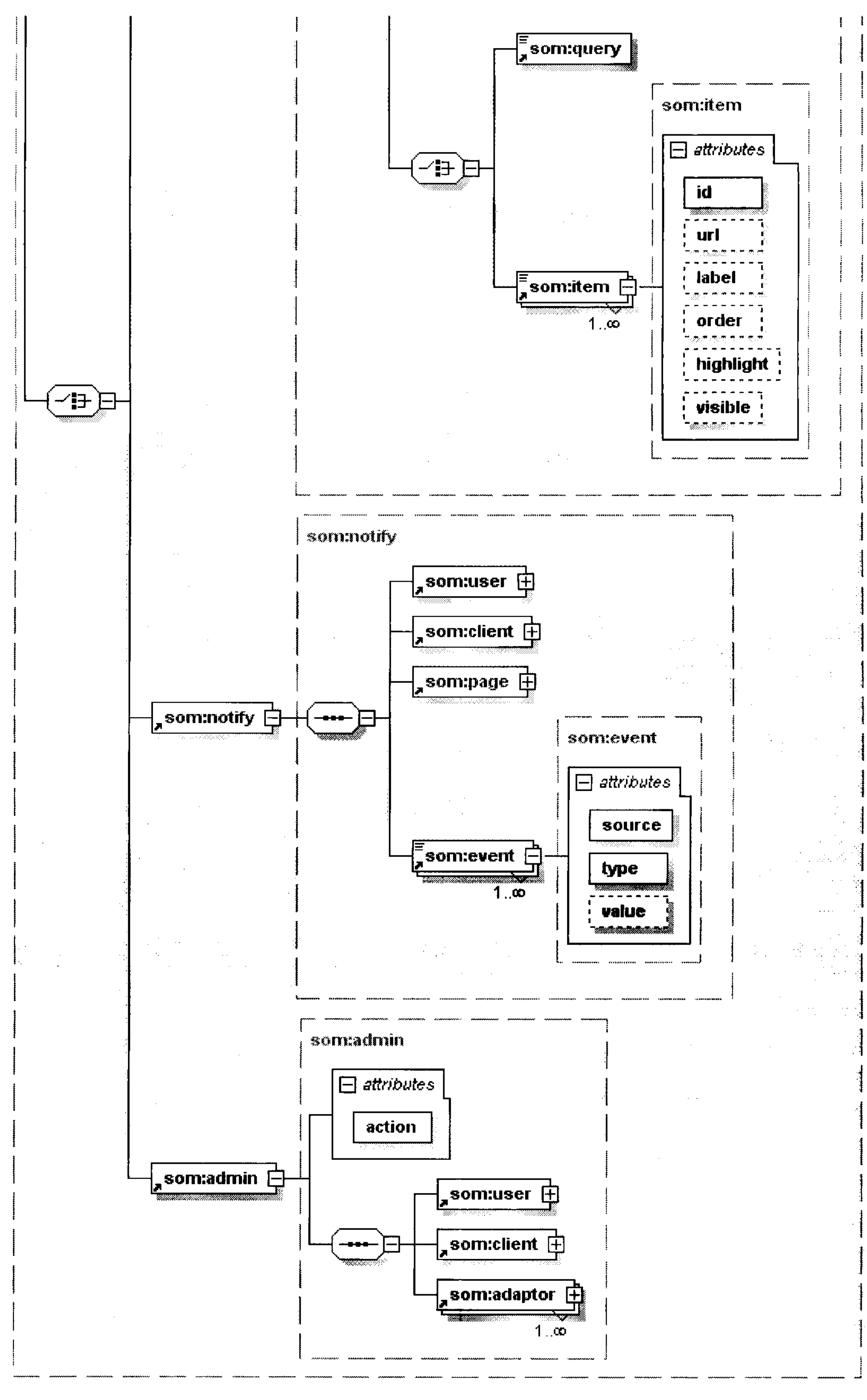
```
</hasMoreItems>
<!-- How to select the next item in the results pool. This -->
<!-- equivalent to shift the pointer of a database query.  -->
<!-- This is not mandatory but in this first version it    -->
<!-- has to have the default value of "1" if not used      -->
<selectNextItem>
1
</selectNextItem>
<!-- How to get the values for each of the specified       -->
<!-- properties. This is used to build each line line of   -->
<!-- items in the response message. Each property name     -->
<!-- can be passed using a variable that can be specified  -->
<!-- with the attribute "name"                             -->
<getItemProperty name="propertyName">
valueof(propertyName);
</getItemProperty>
</configs>
```

Apêndice C

Esquema das mensagens

Esta imagem esquematiza o XSD usado na construção das mensagens que circulam na *framework*.


som:message
attributes
mode
som:adapt
som:user
attributes
som:user
id
som:client
attributes
browser
som:client
ip
referer
som:page
attributes
id
som:page
url
message
som:adapt
label
som:adaptor
attributes
id
som:adaptor
status
url


som:query
som:item
attributes
id
url
label
order
highlight
visible
som:item
1..∞
som:notify
som:user
som:client
som:page
som:notify
som:event
attributes
source
type
value
som:event
1..∞
som:admin
attributes
action
som:admin
som:user
som:client
som:adaptor
1..∞

Bibliografia

[1] *Amazon Search Engine.* http://a9.com/.

[2] *Apache Software Foundation.* http://www.apache.org/.

[3] *Apache Axis.* http://ws.apache.org/axis/.

[4] *Axis API.* http://ws.apache.org/axis/java/apiDocs/index.html.

[5] *DOM Object Reference.* http://www.xulplanet.com/references/objref/#dom.

[6] *Document Object Model Specifications.* http://www.w3.org/DOM/DOMTR.

[7] *Introduction to Document Type Definition.* http://www.xmlfiles.com/dtd/dtd_intro.asp.

[8] Thomas Erl. *Service-Oriented Architecture - Concepts, Technology, and Design.* Prentice Hall, 2005.

[9] David Flanagan. *JavaScript The Definitive Guide.* O'Reilly, 2002.

[10] *Google Maps.* http://maps.google.com/.

[11] *Google Suggest.* http://www.google.com/webhp?complete=1.

[12] *XHTML 4.1 Specification.* http://www.w3.org/TR/html4/.

[13] *Global Structure of an HTML Document.* http://www.w3.org/TR/html4/struct/global.html.

[14] *htmlunit.* http://htmlunit.sourceforge.net/.

[15] *IBM Mastering Ajax.* http://www-128.ibm.com/developerworks/web/library/wa-ajaxintro1.html.

[16] *Instant Domain Search.* http://instantdomainsearch.com/.

[17] *JMeter.* http://jakarta.apache.org/jmeter/.

[18] *JSUnit.* http://www.jsunit.net/.

[19] *JWebUnit.* http://jwebunit.sourceforge.net/.

[20] Brett McLaughin. *Java and XML.* O'Reilly, 2000.

[21] *Mozilla Developer Center - Ajax.* http://developer.mozilla.org/en/docs/AJAX.

[22] *Mozilla Developer.* http://developer.mozilla.org/.

[23] *Netflix TOP 100.* http://www.netflix.com/Top100.

[24] *NIAAD.* http://www.niaad.liacc.up.pt/.

[25] Erik T. Ray. *Learning XML.* O'Reilly, 2003.

[26] *Selenium.* http://www.openqa.org/selenium/.

[27] *Selenium.* http://www.openqa.org/selenium-rc/.

[28] *SOAP.* http://www.w3.org/TR/soap/.

[29] *Site-O-Matic.* http://www.niaad.liacc.up.pt/Site-O-Matic.

[30] *Apache Tomcat.* http://tomcat.apache.org/.

[31] *Web Naming and Addressing.* http://www.w3.org/Addressing/.

[32] *W3C.* http://www.w3.org/.

[33] *The Web Word Processor.* http://www.writely.com/.

[34] *XHTML Specification.* http://www.w3.org/TR/xhtml1/.

[35] *XML Namespaces.* http://www.w3.org/TR/REC-xml-names/.

[36] *XML.* http://www.w3.org/XML/.

[37] *XMLSpy.* http://www.altova.com/products/xmlspy/xml_editor.html.

[38] *XML Schema Definition.* http://www.w3.org/XML/Schema.

Universidade do Porto
Faculdade de Ciências
DEPARTAMENTO DE
CIÊNCIA DE COMPUTADORES
BIBLIOTECA